

目 录

第一部分 C++ 语言

第一章 C++ 语言入门	3
第二章 顺序结构程序设计	7
第一节 运算符和表达式	7
第二节 常量和变量	11
第三节 标准数据类型	14
第四节 数据输入输出	19
第五节 顺序结构实例	22
第三章 程序的控制结构	28
第一节 if 选择结构	28
第二节 switch 语句	34
第四章 循环结构	40
第一节 for 语句	40
第二节 while 语句	58
第三节 do-while 语句	62
第四节 循环嵌套	67
第五章 数组	75
第一节 一维数组	75
第二节 二维数组	89
第三节 字符类型和字符数组	98
第六章 函数	124
第一节 函数	124
第二节 递归算法	146
第七章 指针及其应用	151
第八章 STL 模板库及其应用	158

第二部分 基础算法

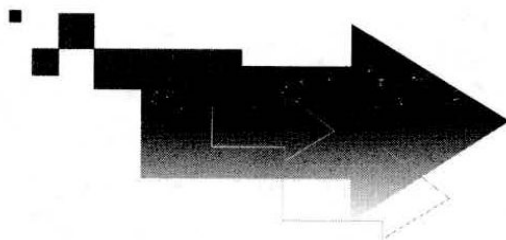
第一章 高精度计算	181
第二章 数据排序	190
第三章 递推算法	203
第四章 递归算法	212
第五章 搜索与回溯算法	227
第六章 贪心算法	245
第七章 分治算法	258
第八章 广度优先搜索算法	274
第九章 动态规划	296
第一节 动态规划的基本模型	296
第二节 背包问题	309
第三节 动态规划经典题	317

第三部分 数据结构

第一章 栈	330
第二章 队列	344
第三章 树	352
第一节 二叉树	352
第二节 堆及其应用	365
第四章 图论算法	374
第一节 图的遍历	374
第二节 最短路径算法	377
第三节 图的连通性问题	394
第四节 并查集	398
第五节 最小生成树	414
第六节 拓扑排序与关键路径	427

第一部分

C++语言



第一章 C++ 语言入门

本书训练在线评测系统 OJ:

【官方平台】<http://noi.openjudge.cn>

【备用平台】<http://ybt.ssoier.cn:8088>

1. Hello, World! 【1.1 编程基础之输入输出 01】

编写一个能够输出“Hello, World!”的程序,这个程序常常作为一个初学者接触一门新的编程语言所写的第一个程序,也经常用来测试开发、编译环境是否能够正常工作。

输入:无。

输出:一行,仅包含一个字符串:“Hello, World!”

样例输入:(无)

样例输出:Hello, World!

【参考程序 1】

```
#include<iostream>                //使用 cout,须调用 iostream 库
using namespace std;
int main()
{
    cout<<" Hello, World! "<<endl; //输出“Hello, World!”
    return 0;                      //结束整个程序
}
```

【参考程序 2】

```
#include<cstdio>                  //使用 printf,须调用 cstdio 库
using namespace std;
int main()
{
    printf(" Hello, World! ");     //输出“Hello, World!”
    return 0;                      //结束整个程序
}
```

2. 输出第二个整数【1.1 编程基础之输入输出 02】

输入三个整数,整数之间由一个空格分隔,整数是 32 位有符号整数。把第二个输入的整数输出。

输入:

只有一行,共三个整数,整数之间由一个空格分隔。整数是 32 位有符号整数。

输出:

只有一行,一个整数,即输入的第二个整数。

样例输入:

123 456 789

样例输出:

456

【参考程序 1】

```
#include<iostream>                //使用 cin,cout,须调用 iostream 库
using namespace std;
```

```
int main()
{
    int a,b,c;                //int 定义 a,b,c 为整型变量
    cin>>a>>b>>c;            //输入给定 3 个整数的值
    cout<<b<<endl;          //输出第 2 个整数的值
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>                //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b,c;                //int 定义 a,b,c 为整型变量
    scanf("%d%d%d",&a,&b,&c);    //输入给定 3 个整数的值,注意"&"不能漏
    printf("%d\n",b);          //输出第 2 个整数的值
    return 0;
}
```

3. 对齐输出【1.1 编程基础之输入输出 03】

读入三个整数,按每个整数占 8 个字符的宽度,右对齐输出它们,按照格式要求依次输出三个整数,之间以一个空格分开。

输入:

只有一行,包含三个整数,整数之间以一个空格分开。

输出:

只有一行,按照格式要求依次输出三个整数,之间以一个空格分开。

样例输入:

123456789 0 -1

样例输出:

123456789 0 -1

【参考程序 1】

```
#include<iostream>                //使用 cin,须调用 iostream 库
#include<cstdio>                //使用 printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b,c;                //int 定义 a,b,c 为整型变量
    cin>>a>>b>>c;            //输入给定 3 个整数的值
    printf("%8d %8d %8d",a,b,c); //输出 3 个整数的值,注意%8d %8d %8d 之间要有空格
    return 0;
}
```

【参考程序 2】

```
#include<iostream>                //使用 cin,cout,须调用 iostream 库
#include<cstdio>                //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
```

```
    int a,b,c;                //int 定义 a,b,c 为整型变量
    scanf("%d%d%d",&a,&b,&c);    //输入给定 3 个整数的值,注意"&"不能漏
```

```
printf("%8d %8d %8d ",a,b,c); //输出 3 个整数的值,注意%8d %8d %8d 之间要有空格
return 0;
```

4. 字符三角形【1.1 编程基础之输入输出 08】

给定一个字符,用它构造一个底边长 5 个字符,高 3 个字符的等腰字符三角形。

输入:

只有一行,包含一个字符。

输出:

该字符构成的等腰三角形,底边长 5 个字符,高 3 个字符。

样例输入:

*

样例输出:

*

* * *

* * * * *

【参考程序 1】

```
#include<iostream> //使用 cin,cout,须调用 iostream 库
using namespace std;
int main()
{
    char a; //定义字符变量 a
    cin>>a; //输入给定 1 个字符
    cout<<"  "<<a<<endl; //输出 1 个字符,先输出 2 个空格
    cout<<" "<<a<<a<<a<<endl; //输出 3 个字符,先输出 1 个空格
    cout<<a<<a<<a<<a<<a<<endl; //输出 5 个字符
    return 0;
}
```

【参考程序 2】

```
#include<iostream> //使用 cin,cout,须调用 iostream 库
#include<cstdio> //使用 scanf,printf,须调用 cstdio 库
using namespace std;
char a; //定义字符变量 a
int main()
{
    scanf("%c",&a); //输入给定 1 个字符,注意"&"不能漏
    printf(" %c\n",a); //输出 1 个字符,先输出 2 个空格
    printf(" %c%c%c\n",a,a,a); //输出 3 个字符,先输出 1 个空格
    printf("%c%c%c%c%c\n",a,a,a,a,a); //输出 5 个字符
    return 0;
}
```

5. 地球人口承载力估计【小学奥数 7653】

假设地球上的新生资源按恒定速度增长。照此测算,地球上现有资源加上新生资源可供 x 亿人生活 a 年,或供 y 亿人生活 b 年。

为了能够实现可持续发展,避免资源枯竭,地球最多能够养活多少亿人?

输入:

一行,包括四个正整数 x, a, y, b , 两个整数之间用单个空格隔开。 $x > y, a < b, ax < by$,

各整数均不大于 10,000。

输出：

一个实数 z ，表示地球最多养活 z 亿人，舍入到小数点后两位。

样例输入：

110 90 90 210

样例输出：

75.00

【参考程序 1】

```
#include<cstdio>           //使用 printf,须调用 cstdio 库
#include<iostream>         //使用 cin,须调用 iostream 库
using namespace std;
int main()
{
    double a,b,x,y,z;       //double 定义 a,b,x,y,z 为双精度实型变量
    cin>>x>>a>>y>>b;      //输入给定 4 个实数
    z=(y*b-x*a)/(b-a);      //计算地球最多能够养活的人数
    printf("%.2lf\n",z);     //输出地球最多能够养活的人数,精确到小数点后两位
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>           //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int x,a,y,b;               //int 定义 x,a,y,b 为整型变量
double s1,s2,s3;           //double 定义 s1,s2,s3 为双精度实型变量
int main()
{
    scanf("%d%d%d%d",&x,&a,&y,&b); //输入给定 4 个整数,注意"&"不能漏
    s1=x*a; s2=y*b; //s1,s2 分别储存 x 亿人生活 a 年,y 亿人生活 b 年各消耗的资源
    s3=(s2-s1)/(b-a);      //计算地球最多能够养活的人数
    printf("%.2lf\n",s3);   //输出地球最多能够养活的人数,精确到小数点后两位
    return 0;
}
```

第二章 顺序结构程序设计

第一节 运算符和表达式

1. A+B 问题【1.3 编程基础之算术表达式与顺序执行 01】

大部分的在线题库,都会将 A+B 问题作为第一题,以帮助新手熟悉平台的使用方法。

A+B 问题的题目描述如下:给定两个整数 A 和 B,输出 A+B 的值。保证 A、B 及结果均在整型范围内。现在请你解决这一问题。

输入:

一行,包含两个整数 A,B,中间用单个空格隔开。A 和 B 均在整型范围内。

输出:

一个整数,即 A+B 的值。保证结果在整型范围内。

样例输入:

1 2

样例输出:

3

【参考程序 1】

```
#include<iostream>           //使用 cin,cout,须调用 iostream 库
using namespace std;
int main()
{
    int a,b;                  //int 定义 a,b 为整型变量
    cin>>a>>b;               //输入给定 2 个整数
    cout<<a+b<<endl;         //输出 a+b 的值
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>             //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b;                  //int 定义 a,b 为整型变量
    scanf("%d%d",&a,&b);       //输入给定 2 个整数的值,注意“&”不能漏
    printf("%d\n",a+b);       //输出 a+b 的值
    return 0;
}
```

2. 计算(a+b)*c 的值【1.3 编程基础之算术表达式与顺序执行 02】

给定 3 个整数 a,b,c,计算表达式(a+b)*c 的值。

输入:

一行,包括三个整数 a,b,c($-10,000 < a,b,c < 10,000$),数与数之间以一个空格分开。

输出:

一行,即表达式的值。

样例输入:

2 3 5

样例输出:

25

【参考程序 1】

```
#include<iostream>                                //使用 cin,cout,须调用 iostream 库
using namespace std;
int main()
{
    int a,b,c;                                    //int 定义 a,b,c 为整型变量
    cin>>a>>b>>c;                                //输入给定 3 个整数
    cout<<(a+b)*c<<endl;                        //输出 (a+b)*c 的值
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>                                //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b,c;                                    //int 定义 a,b,c 为整型变量
    scanf("%d%d%d",&a,&b,&c);                    //输入给定 3 个整数,注意“&”不能漏
    printf("%d\n",(a+b)*c);                      //输出 (a+b)*c 的值
    return 0;
}
```

3. 计算 $(a+b)/c$ 的值【1.3 编程基础之算术表达式与顺序执行 03】

给定 3 个整数 a, b, c , 计算表达式 $(a+b)/c$ 的值, “/”是整除运算。

输入:

一行,包括三个整数 a, b, c , 数与数之间以一个空格分开。 $(-10,000 < a, b, c < 10,000, c \neq 0)$

输出:

输出一行,即表达式的值。

样例输入:

1 1 3

样例输出:

0

【参考程序 1】

```
#include<iostream>                                //使用 cin,cout,须调用 iostream 库
using namespace std;
int main()
{
    int a,b,c;                                    //int 定义 a,b,c 为整型变量
    cin>>a>>b>>c;                                //输入给定 3 个整数
    cout<<(a+b)/c<<endl;                        //输出 (a+b)/c 的值
    return 0;
}
```


【参考程序 2】

```
#include<stdio> //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b,c,s; //int 定义 a,b,c,s 为整型变量
    scanf("%d%d%d",&a,&b,&c); //输入给定 3 个整数,注意"&"不能漏
    s=(a+b)/c; //s 储存(a+b)/c 的值
    printf("%d\n",s); //输出 s 的值
    return 0;
}
```

4. 带余除法【1.3 编程基础之算术表达式与顺序执行 04】

给定被除数和除数,求整数商及余数。此题中请使用默认的整除和取余运算,无须对结果进行任何特殊处理。

输入:

一行,包含两个整数,依次为被除数和除数(除数非零),中间用一个空格隔开。

输出:

一行,包含两个整数,依次为整数商和余数,中间用一个空格隔开。

样例输入:

10 3

样例输出:

3 1

【参考程序 1】

```
#include<iostream> //使用 cin,cout,须调用 iostream 库
using namespace std;
int main()
{
    int a,b; //int 定义 a,b 为整型变量
    cin>>a>>b; //输入给定 2 个整数
    cout<<a/b<<" "<<a%b<<endl; //输出 a,b 的整数商和余数,中间用空格隔开
    return 0;
}
```

【参考程序 2】

```
#include<stdio> //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    int a,b,r,l; //int 定义 a,b,r,l 为整型变量
    scanf("%d%d",&a,&b); //输入给定 2 个整数的值
    r=a/b; //r 储存 a,b 的整数商
    l=a%b; //l 储存 a,b 的余数
    printf("%d %d\n",r,l); //输出 r 与 l 的值,注意%d %d 之间有空格
    return 0;
}
```

5. 计算分数的浮点数值【1.3 编程基础之算术表达式与顺序执行 05】

两个整数 a 和 b 分别作为分子和分母,即分数 a/b,求它的浮点数值(双精度浮点数,保

留小数点后 9 位)。

输入:

仅一行,包括两个整数 a 和 b。

输出:

仅一行,分数 a/b 的浮点数值(双精度浮点数,保留小数点后 9 位)。

样例输入:

5 7

样例输出:

0.714285714

【参考程序 1】

```
#include<iostream>                //使用 cin,须调用 iostream 库
#include<cstdio>                    //使用 printf,须调用 cstdio 库
using namespace std;
int main()
{
    double a,b;                    //double 定义 a,b 为双精度实型变量
    cin>>a>>b;                    //输入给定 2 个实数
    printf("%.9lf",a/b);           //输出分数 a/b 的浮点数值,精确到小数点后 9 位
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>                    //使用 scanf,printf,须调用 cstdio 库
using namespace std;
int main()
{
    double a,b;                    //double 定义 a,b 为双精度实型变量
    scanf("%lf%lf",&a,&b);         //输入给定 2 个实数,注意"&"不能漏
    printf("%.9lf\n",a/b);         //输出分数 a/b 的浮点数值,精确到小数点后 9 位
    return 0;
}
```

第二节 常量和变量

1. 甲流疫情死亡率【1.3 编程基础之算术表达式与顺序执行 06】

甲流并不可怕,在中国,它的死亡率并不是很高。请根据截至 2009 年 12 月 22 日各省报告的甲流确诊数和死亡数,计算甲流在各省的死亡率。

输入:

一行,有两个整数,第 1 个为确诊数,第 2 个为死亡数。

输出:

一行,甲流死亡率,以百分数形式输出,精确到小数点后 3 位。

样例输入:

10433 60

样例输出:

0.575%

【参考程序】

```
#include<iostream>                //使用 cin,须调用 iostream 库
#include<cstdio>                    //使用 printf,须调用 cstdio 库
using namespace std;
int main()
{
    double a,b;                    //double 定义 a,b 为双精度实型变量
    cin>>a>>b;                    //输入给定确诊数与死亡数
    printf("%.3lf%%",b/a*100);     //输出甲流死亡率,精确到小数点后 3 位,要输出一个百分号
    return 0;
}
```

2. 计算多项式的值【1.3 编程基础之算术表达式与顺序执行 07】

对于多项式 $f(x) = ax^3 + bx^2 + cx + d$ 和给定的 a, b, c, d, x , 计算 $f(x)$ 的值,保留到小数点后 7 位。

输入:

一行,包含五个实数,分别是 x , 参数 a, b, c, d 的值,每个数都是绝对值不超过 100 的双精度浮点数。数与数之间以一个空格分开。

输出:

一个实数,即 $f(x)$ 的值,保留到小数点后 7 位。

样例输入:

2.31 1.2 2 2 3

样例输出:

33.0838692

【参考程序】

```
#include<iostream>                //使用 cin,须调用 iostream 库
#include<cstdio>                    //使用 printf,须调用 cstdio 库
using namespace std;
int main()
{
    double x,a,b,c,d;              //double 定义 x,a,b,c,d 为双精度实型变量
```

```
cin>>x>>a>>b>>c>>d;    //输入给定 5 个实数
printf("%.7lf",a*x*x*x+b*x*x+c*x+d);
                                //输出 f(x) 的值
return 0;
}
```

3. 温度表达转化【1.3 编程基础之算术表达式与顺序执行 08】

利用公式 $C = 5 * (F - 32) / 9$ (其中 C 表示摄氏温度, F 表示华氏温度) 进行计算转化, 输入华氏温度 f , 输出摄氏温度 c , 要求精确到小数点后 5 位。

输入:

一行, 包含一个实数 f ($f \geq -459.67$), 表示华氏温度。

输出:

一行, 包含一个实数, 表示对应的摄氏温度, 要求精确到小数点后 5 位。

样例输入:

41

样例输出:

5.00000

【参考程序】

```
#include<iostream>    //使用 cin, 须调用 iostream 库
#include<cstdio>    //使用 printf, 须调用 cstdio 库
using namespace std;
int main()
{
    double f;    //double 定义 f 为双精度实型变量
    cin>>f;    //输入给定华氏温度
    printf("%.5lf",5*(f-32)/9);    //输出对应的摄氏温度, 精确到小数点后 5 位
    return 0;
}
```

4. 与圆相关的计算【1.3 编程基础之算术表达式与顺序执行 09】

给出圆的半径, 求圆的直径、周长和面积。输入圆的半径实数 r , 输出圆的直径、周长、面积, 每个数保留小数点后 4 位。

输入:

包含一个实数 r ($0 < r \leq 10,000$), 表示圆的半径。

输出:

一行, 包含三个数, 分别表示圆的直径、周长、面积, 数与数之间以一个空格分开, 每个数保留小数点后 4 位。

样例输入:

3.0

样例输出:

6.0000 18.8495 28.2743

【参考程序】

```
#include<iostream>    //使用 cin, 须调用 iostream 库
#include<cstdio>    //使用 printf, 须调用 cstdio 库
using namespace std;
const double pi=3.14159;    //const double 定义 pi 为双精度实型常量
int main()
```

```

{
    double r,d,c,s;           //double 定义 r,d,c,s 为双精度实型变量
    cin>>r;                   //输入圆的半径
    d=r*2;                    //计算圆的直径
    c=2*pi*r;                 //计算圆的周长
    s=pi*r*r;                 //计算圆的面积
    printf("%.4f %.4f %.4f ",d,c,s);
        //分别输出圆的直径、周长、面积,数与数之间以一个空格分开
    return 0;
}

```

5. 计算并联电阻的阻值【1.3 编程基础之算术表达式与顺序执行 10】

对于阻值为 r_1 和 r_2 的电阻,其并联电阻阻值公式计算如下: $R=1/(1/r_1+1/r_2)$ 。输入两个电阻阻抗大小,浮点型。输出并联之后的阻抗大小,结果保留小数点后 2 位。

输入:

两个电阻阻抗大小,浮点型,以一个空格分开。

输出:

并联之后的阻抗大小,结果保留小数点后 2 位。

样例输入:

1 2

样例输出:

0.67

【参考程序】

```

#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    float r1,r2;              //float 定义 r1,r2 为单精度实型变量
    cin>>r1>>r2;              //输入 2 个电阻阻抗大小
    printf("%.2f\n",1/(1/r1+1/r2)); //输出并联之后的阻抗大小,精确到小数点后 2 位
    return 0;
}

```

第三节 标准数据类型

1. 整型数据类型存储空间大小【1.2 编程基础之变量定义、赋值及转换 01】

分别定义 int, short 类型的变量各一个, 并依次输出它们的存储空间大小(单位: 字节)。

输入:

无。

输出:

一行, 包含两个整数, 分别是两个变量的存储空间大小, 用一个空格隔开。

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    int x; short y;                //定义整型变量 x, 短整型变量 y
    cout<<sizeof(x)<<" "<<sizeof(y)<<endl;
                                   //分别输出 2 个变量储存空间大小, 用 1 个空格隔开
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
{
    int a; short b;                //定义整型变量 x, 短整型变量 y
    printf("%d %d\n", sizeof(a), sizeof(b));
                                   //分别输出 2 个变量储存空间大小, 用 1 个空格隔开
    return 0;
}
```

2. 浮点型数据类型存储空间大小【1.2 编程基础之变量定义、赋值及转换 02】

分别定义 float, double 类型的变量各一个, 并依次输出它们的存储空间大小(单位: 字节)。

输入:

无。

输出:

一行, 包含两个整数, 分别是两个变量的存储空间大小, 用一个空格隔开。

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    float x; double y;            //定义单精度实型变量 x, 双精度实型变量 y
    cout<<sizeof(x)<<" "<<sizeof(y)<<endl;
                                   //分别输出 2 个变量储存空间大小, 用 1 个空格隔开
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
{
    float x; double y;           //定义单精度实型变量 x,双精度实型变量 y
    printf("%d %d ",sizeof(x),sizeof(y));
                                //分别输出 2 个变量储存空间大小,用 1 个空格隔开
    return 0;
}
```

3. 其他数据类型存储空间大小【1.2 编程基础之变量定义、赋值及转换 03】

分别定义 bool, char 类型的变量各一个,并依次输出它们的存储空间大小(单位:字节)。

输入:

无。

输出:

一行,包含两个整数,分别是两个变量的存储空间大小,用一个空格隔开。

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    bool x; char y;             //定义布尔型变量 x,字符型变量 y
    cout<<sizeof(x)<<" "<<sizeof(y)<<endl;
                                //分别输出 2 个变量储存空间大小,用 1 个空格隔开
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
{
    bool x; char y;             //定义布尔型变量 x,字符型变量 y
    printf("%d %d ",sizeof(x),sizeof(y));
                                //分别输出 2 个变量储存空间大小,用 1 个空格隔开
    return 0;
}
```

4. 浮点数向零舍入【1.2 编程基础之变量定义、赋值及转换 06】

输入一个单精度浮点数,将其向零舍入到整数。说明:向零舍入的含义是,正数向下舍入,负数向上舍入。提示:可以使用强制类型转换来实现。

输入:

一个单精度浮点数。

输出:

一个整数,即向零舍入到整数的结果。

样例输入:

2.3

样例输出:

2

【参考程序 1】

```
#include<cstdio>
#include<iostream>
using namespace std;
int main()
{
    float a;                //定义单精度实型变量 a
    cin>>a;                //输入给定 1 个实数
    cout<<(int)a<<endl;    //输出 a 的整数部分,即向零舍入到整数的结果
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
{
    float s; int a;          //定义单精度实型变量 s,整型变量 a
    scanf("%f",&s);          //输入给定 1 个实数
    a=s;                    //a 储存 s 的整数部分,即向零舍入到整数的结果
    printf("%d\n",a);        //输出 a 的值
    return 0;
}
```

5. 打印 ASCII 码【1.2 编程基础之变量定义、赋值及转换 07】

输入一个除空格以外的可见字符(保证在函数 scanf 中可使用格式说明符%c 读入),输出其 ASCII 码。

输入:

一个除空格以外的可见字符。

输出:

一个十进制整数,即该字符的 ASCII 码。

样例输入:

A

样例输出:

65

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    char a;                //定义字符型变量 a
    cin>>a;                //输入给定 1 个除空格以外的可见字符
    cout<<(int)a<<endl;    //输出 a 的 ASCII 码
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
```



```

{
    char a;                //定义字符型变量 a
    scanf("%c",&a);        //输入给定 1 个除空格以外的可见字符
    int k=a;                //k 储存 a 的 ASCII 码
    printf("%d\n",k);       //输出 k 的值
    return 0;
}

```

6. 打印字符【1.2 编程基础之变量定义、赋值及转换 08】

输入一个整数,即字符的 ASCII 码,保证存在对应的可见字符。输出相对应的字符。

输入:

一个整数,即字符的 ASCII 码,保证存在对应的可见字符。

输出:

一行,包含相应的字符。

样例输入:

65

样例输出:

A

【参考程序 1】

```

#include<iostream>
using namespace std;
int main()
{
    int a;                //定义整型变量 a
    cin>>a;                //输入给定 1 个整数,即字符的 ASCII 码
    cout<<(char)a;        //输出 a 相对应的字符
    return 0;
}

```

【参考程序 2】

```

#include<cstdio>
using namespace std;
int main()
{
    int n; char s;        //定义整型变量 n,字符变量 s
    scanf("%d",&n);        //输入给定 1 个整数,即字符的 ASCII 码
    s=n;                  //s 储存 n 相对应的字符
    printf("%c\n",s);     //输出 s 的值
    return 0;
}

```

7. 整型与布尔型的转换【1.2 编程基础之变量定义、赋值及转换 09】

将一个整型变量的值赋给一个布尔型变量,再将这个布尔型变量的值赋给一个整型变量,得到的值是多少?

输入:

一个整型范围内的整数,即初始时整型变量的值。

输出:

一个整数,经过上述过程后得到的结果。

样例输入:

3

样例输出:

1

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a;bool n;           //定义整型变量 a,布尔型变量 n
    cin>>a;                //输入给定 1 个整数
    n=a;                   //n 储存 a 相对应的布尔型
    cout<<n<<endl;        //输出 n 相对应的整数
}
```

8. Hello, World! 的大小【1.2 编程基础之变量定义、赋值及转换 10】

在上一章里,我们曾经输出过的“Hello, World!”它虽然不是本章所涉及的基本数据类型的数据,但我们同样可以用 sizeof 函数获得它所占用的空间大小。

输入:

无。

输出:

一个整数,即“Hello, World!”的大小。

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    cout<<sizeof(" Hello, World! "); //输出“Hello, World!”的大小
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>
using namespace std;
int main()
{
    printf("%d\n",sizeof(" Hello, World! ")); //输出“Hello, World!”的大小
    return 0;
}
```

第四节 数据输入输出

1. 输出保留 3 位小数的浮点数【1.1 编程基础之输入输出 04】

读入一个单精度浮点数,保留 3 位小数输出这个浮点数。

输入:

只有一行,一个单精度浮点数。

输出:

也只有一行,读入的单精度浮点数保留 3 位小数的结果。

样例输入:

12.34521

样例输出:

12.345

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    float x;                //定义单精度实型变量 x
    scanf("%f",&x);          //输入给定 1 个单精度浮点数
    printf("%.3f\n",x);      //输出 x 的值,精确到小数点后 3 位
    return 0;
}
```

2. 输出保留 12 位小数的浮点数【1.1 编程基础之输入输出 05】

读入一个双精度浮点数,保留 12 位小数,输出这个浮点数。

输入:

只有一行,一个双精度浮点数。

输出:

也只有一行,保留 12 位小数的浮点数。

样例输入:

3.1415926535798932

样例输出:

3.141592653580

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    double x;                //定义双精度实型变量 x
    scanf("%lf",&x);          //输入给定 1 个双精度浮点数
    printf("%.12lf",x);      //输出 x 的值,精确到小数点后 12 位
}
```

3. 空格分隔输出【1.1 编程基础之输入输出 06】

读入一个字符、一个整数、一个单精度浮点数、一个双精度浮点数,然后按顺序输出它们,并且要求在它们之间用一个空格分隔。输出浮点数时保留 6 位小数。

输入:

- 第 1 行是一个字符;
- 第 2 行是一个整数;
- 第 3 行是一个单精度浮点数;
- 第 4 行是一个双精度浮点数。

输出:

输出字符、整数、单精度浮点数和双精度浮点数,用空格分隔。

样例输入:

```
a
12
2.3
3.2
```

样例输出:

```
a 12 2.300000 3.200000
```

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    char a; int b; float c; double d;
    //定义字符变量 a,整型变量 b,单精度实型变量 c,双精度实型变量 d
    scanf("%c%d%f%lf",&a,&b,&c,&d);
    //输入给定 1 个字符,1 个整数,1 个单精度浮点数,1 个双精度浮点数
    printf("%c %d %.6f %.6lf\n",a,b,c,d);
    //输出 a,b,c,d,中间用空格隔开,c,d 精确到小数点后 6 位
    return 0;
}
```

4. 输出浮点数【1.1 编程基础之输入输出 07】

读入一个双精度浮点数,分别按输出格式“%f”,“%f”保留 5 位小数,“%e”和“%g”的形式输出这个整数,每次在单独一行上输出。

输入:

一个双精度浮点数。

输出:

- 第 1 行是按“%f”输出的双精度浮点数;
- 第 2 行是按“%f”保留 5 位小数输出的双精度浮点数;
- 第 3 行是按“%e”输出的双精度浮点数;
- 第 4 行是按“%g”输出的双精度浮点数。

样例输入:

```
12.3456789
```

样例输出:

```
12.345679
12.34568
1.234568e+001
12.3457
```

【参考程序】

```
#include<cstdio>
using namespace std;
```

```

int main()
{
    double a;                //定义双精度实型变量 a
    scanf("%lf",&a);         //输入给定 1 个双精度浮点数
    printf("%lf\n",a);        //按“%lf”输出 a
    printf("%.5lf\n",a);      //按“%lf”输出 a,精确到小数点后 5 位
    printf("%e\n",a);         //按“%e”输出 a
    printf("%g\n",a);         //按“%g”输出 a
}

```

5. 字符菱形【1.1 编程基础之输入输出 09】

给定一个字符,用它构造一个对角线长 5 个字符,倾斜放置的菱形。

输入:

输入只有一行,包含一个字符。

输出:

该字符构成的菱形。

样例输入:

*

样例输出:

```

    *
  * * *
* * * * *
  * * *
    *

```

【参考程序】

```

#include<iostream>
#include<cstdio>
using namespace std;
char a;                //定义字符型变量 a
int main()
{
    scanf("%c",&a);      //输入给定 1 个字符
    printf("  %c\n",a);  //输出 1 个 a,先输出 2 个空格
    printf(" %c%c%c\n",a,a,a); //输出 3 个 a,先输出 1 个空格
    printf("%c%c%c%c%c\n",a,a,a,a,a); //输出 5 个 a
    printf(" %c%c%c\n",a,a,a); //输出 3 个 a,先输出 1 个空格
    printf("  %c\n",a);      //输出 1 个 a,先输出 2 个空格
    return 0;
}

```

第五节 顺序结构实例

1. 计算浮点数相除的余数【1.3 编程基础之算术表达式与顺序执行 11】

计算两个双精度浮点数 a 和 b 相除的余数, a 和 b 都是双精度浮点数。这里余数(r)的定义是: $a = k * b + r$, 其中 k 是整数, $0 \leq r < b$ 。

输入:

一行, 包括两个双精度浮点数 a 和 b 。

输出:

一行, $a \div b$ 的余数。

样例输入:

73.263 0.9973

样例输出:

0.4601

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    double a,b,r;           //定义双精度实型变量 a,b,r
    int k;                   //定义整型变量 k
    scanf("%lf%lf",&a,&b);   //输入给定 2 个双精度浮点数
    k=a/b;                   //k 储存 a/b 的商
    r=a-k*b;                 //r 储存 a/b 的余数
    printf("%g",r);          //输出 r
    return 0;
}
```

2. 计算球的体积【1.3 编程基础之算术表达式与顺序执行 12】

对于半径为 r 的球, 其体积的计算公式为 $V = 4/3 * \pi r^3$, 这里取 $\pi = 3.14$ 。现给定 r , 即球半径, 类型为 `double`, 求球的体积 V , 保留到小数点后 2 位。

输入:

一个不超过 100 的非负实数, 即球半径, 类型为 `double`。

输出:

一个实数, 即球的体积, 保留到小数点后 2 位。

样例输入:

4

样例输出:

267.95

【参考程序】

```
#include<cstdio>
double r;           //定义双精度实型变量 r
int main()
{
    scanf("%lf",&r);   //输入球的半径
```

```
printf("%.2lf", 4.0/3 * 3.14 * r * r * r); //输出球的体积
return 0;
```

3. 反向输出一个三位数【1.3 编程基础之算术表达式与顺序执行 13】

将一个三位数反向输出,例如输入 358,反向输出 853。

输入:

一个三位数 n 。

输出:

反向输出 n 。

样例输入:

100

样例输出:

001

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n,b,c,d; //定义整型变量 n,b,c,d
    scanf("%d",&n); //输入给定 1 个整数 n
    b=n/100; //b 储存 n 的百位
    c=(n%100)/10; //c 储存 n 的十位
    d=n%10; //d 储存 n 的个位
    printf("%d%d%d",d,c,b); //反向输出 n
    return 0;
}
```

4. 大象喝水【1.3 编程基础之算术表达式与顺序执行 14】

一只大象口渴了,要喝 20 升水才能解渴,但现在只有一个深 h 厘米,底面半径为 r 厘米的小圆桶(h 和 r 都是整数)。问大象至少要喝多少桶水才会解渴。

输入:

一行,包含两个整数,以一个空格分开,分别表示小圆桶的深 h 和底面半径 r ,单位都是厘米。

输出:

一行,包含一个整数,表示大象至少要喝水的桶数。

样例输入:

23 11

样例输出:

3

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int h,r,a,b; //定义整型变量 h,r,a,b
    scanf("%d%d",&h,&r); //输入小圆桶的深与底面半径
    a=3.14159 * r * r * h; //a 储存一个小圆桶的蓄水量
```

```

b=20000/a+1;           //b 储存大象至少要喝水的桶数
printf("%d ",b);        //输出 b 的值
return 0;
}

```

5. 计算线段长度【1.3 编程基础之算术表达式与顺序执行 16】

已知线段的两个端点的坐标 $A(x_a, y_a)$, $B(x_b, y_b)$, 求线段 AB 的长度, 保留到小数点后 3 位。

输入:

第 1 行是两个实数 x_a, y_a , 即 A 的坐标;
 第 2 行是两个实数 x_b, y_b , 即 B 的坐标。
 输入中所有实数的绝对值均不超过 10000。

输出:

一个实数, 即线段 AB 的长度, 保留到小数点后 3 位。

样例输入:

1 1
 2 2

样例输出:

1.414

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<math.h>           //使用 sqrt, 须调用 math.h 库
using namespace std;
int main()
{
    double xa,ya,xb,yb,n;   //定义双精度实型变量 xa,ya,xb,yb,n
    scanf("%lf%lf%lf%lf",&xa,&ya,&xb,&yb); //输入 A,B 的坐标
    n=sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb)); //求线段 AB 的长度
    printf("%.3lf",n);      //输出 n, 即线段 AB 的长度
    return 0;
}

```

6. 计算三角形面积【1.3 编程基础之算术表达式与顺序执行 17】

平面上有一个三角形, 它的三个顶点坐标分别为 (x_a, y_a) , (x_b, y_b) , (x_c, y_c) , 那么请问这个三角形的面积是多少, 精确到小数点后两位。

输入:

一行, 包括 6 个单精度浮点数, 分别对应 $x_a, y_a, x_b, y_b, x_c, y_c$ 。

输出:

一行, 输出三角形的面积, 精确到小数点后两位。

样例输入:

0 0 4 0 0 3

样例输出:

6.00

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cmath>

```



```
#include<math.h>
using namespace std;
int main()
{
    float xa,ya,xb,yb,xc,yc,n,a,b,c,r,s;
    //定义单精度实型变量 xa,ya,xb,yb,xc,yc,n,a,b,c,r,s
    scanf("%f%f%f%f%f%f", &xa, &ya, &xb, &yb, &xc, &yc);
    //输入三角形三个角的坐标
    a=sqrt((xa-xb)*(xa-xb)+(ya-yb)*(ya-yb));
    b=sqrt((xb-xc)*(xb-xc)+(yb-yc)*(yb-yc));
    c=sqrt((xc-xa)*(xc-xa)+(yc-ya)*(yc-ya));
    //a,b,c 储存三角形三条边的长度
    r=(a+b+c)/2; //r 储存(a+b+c)/2 的值
    s=sqrt(r*(r-a)*(r-b)*(r-c));
    //s 储存三角形的面积
    printf("%.2f",s); //输出 s
    return 0;
}
```

7. 等差数列末项计算【1.3 编程基础之算术表达式与顺序执行 18】

给出一个等差数列的前两项 a_1, a_2 , 求第 n 项是多少。

输入:

一行, 包含三个整数 a_1, a_2, n ($-100 \leq a_1, a_2 \leq 100, 0 < n \leq 1000$)。

输出:

一个整数, 即第 n 项的值。

样例输入:

1 4 100

样例输出:

298

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a1,a2,n; //定义整型变量 a1,a2,n
    cin>>a1>>a2>>n; //读入这个等差数列的前两项及 n 的值
    cout<<a1+(a2-a1)*(n-1)<<endl; //输出第 n 项的值
    return 0;
}
```

8. A * B 问题【1.3 编程基础之算术表达式与顺序执行 19】

输入两个正整数 A 和 B , 求 $A * B$ 的值。注意乘积的范围和数据类型的选择。

输入:

一行, 包含两个正整数 A 和 B , 中间用单个空格隔开。 $1 \leq A, B \leq 50,000$ 。

输出:

一个整数, 即 $A * B$ 的值。

样例输入:

3 4

样例输出：

12

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
int main()
{
    long long a,b,c;
    //定义超长整型变量 a,b,c,注意变量定义为 long long,否则超出范围
    scanf("%lld%lld",&a,&b); //输入给定 2 个正整数,注意输入超长整型数时用%lld
    c=a*b; //c 储存两个正整数的积
    printf("%lld\n",c);
    //输出 c,即 a*b 的值,注意输出超长整型数时用%lld,详见教程第四章第一节说明
    return 0;
}
```

9. 计算 2 的幂【1.3 编程基础之算术表达式与顺序执行 20】

给定非负整数 n , 求 2^n 的值, 即 2 的 n 次方。

输入：

一个整数 $n(0 \leq n < 31)$ 。

输出：

一个整数, 即 2 的 n 次方。

样例输入：

3

样例输出：

8

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cmath> //使用 pow 须调用 cmath 库
using namespace std;
int main()
{
    double n,s; //定义双精度实型变量 n,s
    scanf("%lf",&n); //输入给定 1 个整数 n
    s=pow(2,n); //计算  $2^n$ , 结果为双精度实数, 储存在 s 里
    printf("%.0lf\n",s); //输出 s, 即 2 的 n 次方, 精确到个位
    return 0;
}
```

10. 苹果和虫子【1.3 编程基础之算术表达式与顺序执行 15】

你买了一箱 n 个苹果, 很不幸的是买完时箱子里混进了一条虫子。虫子每 x 小时能吃掉一个苹果, 假设虫子在吃完一个苹果之前不会吃另一个, 那么经过 y 小时你还有多少个完整的苹果?

输入：

一行, 三个整数 n, x 和 y 。

输出:

一行,剩下的苹果个数

样例输入:

10 4 9

样例输出:

7

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cmath>           //使用 ceil()须调用 cmath 库
using namespace std;
int main()
{
    double n,x,y,a,b,c;    //定义双精度实型变量 n,x,y,a,b,c
    cin>>n>>x>>y;        //输入给定 3 个整数
    a=y/x; b=n-ceil(a);

    //a 储存在 y 小时中,虫子能吃完的苹果数;
    //b 储存 y 小时后剩下的苹果数。
    //ceil(a)是求不小于实数 a 的最小整数
    //处理 b<0 的情况
    if (b<0) b=0;
    cout<<b<<endl;
    return 0;
}
```

第三章 程序的控制结构

第一节 if 选择结构

1. 判断数的正负【1.4 编程基础之逻辑表达式与条件分支 01】

给定一个整数 n , 判断其正负。如果 $n > 0$, 输出 positive; 如果 $n = 0$, 输出 zero; 如果 $n < 0$, 输出 negative。

输入:

一个整数 $n (-10^9 \leq n \leq 10^9)$ 。

输出:

如果 $n > 0$, 输出 positive;
如果 $n = 0$, 输出 zero;
如果 $n < 0$, 输出 negative。

样例输入:

1

样例输出:

positive

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    long n;                //定义长整型变量 n
    scanf("%ld",&n);        //输入给定 1 个整数
    if (n>0) printf("positive ");
    else if (n==0) printf("zero ");
    else printf("negative ");    //判断 n 的正负
    return 0;
}
```

2. 输出绝对值【1.4 编程基础之逻辑表达式与条件分支 02】

输入一个浮点数, 输出这个浮点数的绝对值, 保留到小数点后两位。

输入:

一个浮点数, 其绝对值不超过 10,000。

输出:

这个浮点数的绝对值, 保留到小数点后两位。

样例输入:

-3.14

样例输出:

3.14

【参考程序】

```
#include<cstdio>
```

```
#include<cmath>                //使用 abs 须调用 cmath 库
using namespace std;
int main()
{
    double n,s;                //定义双精度实型变量 n,s
    scanf("%lf",&n);           //输入给定 1 个双精度浮点数
    s=abs(n);                  //s 储存 n 的绝对值
    printf("%.2lf\n",s);       //输出 s,精确到小数点后两位
    return 0;
}
```

3. 奇偶数判断【1.4 编程基础之逻辑表达式与条件分支 03】

给定一个整数,判断该数是奇数还是偶数。如果 n 是奇数,输出 odd;如果 n 是偶数,输出 even。

输入:

一行,一个大于零的正整数 n。

输出:

一行,如果 n 是奇数,输出 odd;如果 n 是偶数,输出 even。

样例输入:

5

样例输出:

odd

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n;                    //定义整型变量 n
    scanf("%d",&n);          //输入给定 1 个整数
    if (n%2==0) printf(" even ");
    else printf(" odd ");     //判断 n 是奇数还是偶数
    return 0;
}
```

4. 奇偶 ASCII 值判断【1.4 编程基础之逻辑表达式与条件分支 04】

任意输入一个字符,判断其 ASCII 是否是奇数,若是,输出 YES,否则,输出 NO。例如,字符 A 的 ASCII 值是 65,则输出 YES,若输入字符 B(ASCII 值是 66),则输出 NO。

输入:

一个字符。

输出:

如果这个字符的 ASCII 值为奇数,则输出 YES,否则,输出 NO。

样例输入:

A

样例输出:

YES

【参考程序】

```
#include<iostream>
#include<cstdio>
```

```
using namespace std;
int main()
{
    int a;           //定义整型变量 a
    char n;          //定义字符型变量 n
    scanf("%c", &n); //输入给定 1 个字符
    a=n;             //a 储存 n 的 ASCII 码
    if(a%2==0) printf(" NO\n");
    else printf(" YES\n"); //判断 a 是奇数还是偶数
    return 0;
}
```

5. 整数大小比较【1.4 编程基础之逻辑表达式与条件分支 05】

输入两个整数 x, y , 比较它们的大小。若 $x > y$, 输出 $>$; 若 $x = y$, 输出 $=$; 若 $x < y$, 输出 $<$ 。

输入:

一行, 包含两个整数 x 和 y ($0 \leq x < 2^{32}, -2^{31} \leq y < 2^{31}$), 中间用单个空格隔开。

输出:

一个字符。若 $x > y$, 输出 $>$; 若 $x = y$, 输出 $=$; 若 $x < y$, 输出 $<$;

样例输入:

1000 100

样例输出:

$>$

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    long long x, y; //定义超长整型变量 x, y
    scanf("%d%d", &x, &y); //输入给定 2 个整数
    if (x > y) printf(">\n");
    else if (x == y) printf("=\n");
    else printf("<\n"); //比较 x 与 y 的大小
    return 0;
}
```

6. 判断是否为两位数【1.4 编程基础之逻辑表达式与条件分支 06】

判断一个正整数是否是两位数(即大于等于 10 且小于等于 99)。若该正整数是两位数, 则输出 1, 否则输出 0。

输入:

一个正整数, 不超过 1000。

输出:

一行。若该正整数是两位数, 输出 1, 否则输出 0。

样例输入:

54

样例输出:

1

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    int n;                //定义整型变量 n
    scanf("%d",&n);        //输入给定 1 个整数
    if ((n>=10)&&(n<=99)) printf("1 ");
    else printf("0 ");      //判断 n 是否是两位数
    return 0;
}
```

7. 收集瓶盖赢大奖【1.4 编程基础之逻辑表达式与条件分支 07】

某饮料公司最近推出了一个“收集瓶盖赢大奖”的活动:如果你拥有 10 个印有“幸运”或 20 个印有“鼓励”的瓶盖,就可以兑换一个神秘大奖。现分别给出你拥有的印有“幸运”和“鼓励”的瓶盖数,判断是否可以兑换大奖。若可以兑换大奖,输出 1,否则输出 0。

输入:

一行,包含两个整数,分别是印有“幸运”和“鼓励”的瓶盖数,用一个空格隔开。

输出:

一行。若可以兑换大奖,输出 1,否则输出 0。

样例输入:

11 19

样例输出:

1

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int x,y;                //定义整型变量 x,y
    scanf("%d%d",&x,&y);    //输入 2 个印有“幸运”与“鼓励”的瓶盖数
    if (x>=10 || y>=20) printf("1 ");
    else printf("0 ");      //判断是否可以兑换大奖
    return 0;
}
```

8. 判断一个数能否同时被 3 和 5 整除【1.4 编程基础之逻辑表达式与条件分支 08】

判断一个数 n 能否同时被 3 和 5 整除,如果能同时被 3 和 5 整除输出 YES,否则输出 NO。

输入:

一行,包含一个整数 n($-1,000,000 < n < 1,000,000$)。

输出:

一行,如果能同时被 3 和 5 整除,那么输出 YES,否则输出 NO。

样例输入:

15

样例输出:

YES

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    long n;                //定义长整型变量 n
    scanf("%ld",&n);        //输入给定 1 个整数
    if ((n%3==0)&&(n%5==0)) printf(" YES ");
    else printf(" NO ");    //判断 n 能否同时被 3 和 5 整除
    return 0;
}
```

9. 判断一个数能否被 3,5,7 整除【1.4 编程基础之逻辑表达式与条件分支 09】

给定一个整数,判断它能否被 3,5,7 整除,并输出以下信息:

- (1)能同时被 3,5,7 整除(直接输出 3 5 7,每个数中间一个空格);
- (2)只能被其中两个数整除(输出两个数,小的在前,大的在后。例如:3 5 或者 3 7 或者 5 7,中间用空格分隔);
- (3)只能被其中一个数整除(输出这个除数);
- (4)不能被任何数整除,输出小写字符“n”,不包括引号。

输入:

一行,一个整数。

输出:

一行,按照描述要求给出整数被 3,5,7 整除的情况。

样例输入:

105

样例输出:

3 5 7

【参考程序】

```
#include<cstdio>
#include<cmath>
using namespace std;
int main()
{
    int n;                //定义整型变量 n
    scanf("%d",&n);        //输入给定 1 个整数
    if ((n%3==0)&&(n%5==0)&&(n%7==0)) printf("3 5 7"); else
    if ((n%3==0)&&(n%5==0)) printf("3 5"); else
    if ((n%3==0)&&(n%7==0)) printf("3 7"); else
    if ((n%5==0)&&(n%7==0)) printf("5 7"); else
    if (n%3==0) printf("3"); else
    if (n%5==0) printf("5"); else
    if (n%7==0) printf("7"); else
    printf("n");          //分别判断 n 被 3,5,7 整除的情况
}
```

10. 有一门课不及格的学生【1.4 编程基础之逻辑表达式与条件分支 10】

给出一名学生的语文和数学成绩,判断他是否恰好有一门课不及格(成绩小于 60 分)。若该生恰好有一门课不及格,输出 1;否则输出 0。

输入:

一行,包含两个在 0 到 100 之间的整数,分别是该生的语文成绩和数学成绩。

输出:

若该生恰好有一门课不及格,输出 1;否则输出 0。

样例输入:

50 80

样例输出:

1

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    long a,b;                //定义长整型变量 a,b
    scanf("%ld%ld",&a,&b);    //输入 2 个整数,是语文成绩和数学成绩
    if ((a>=60)&&(b>=60)) printf("0 ");
    else if ((a<60)&&(b<60)) printf("0 ");
    else printf("1 ");        //判断该生是否恰好有一门课不及格
    return 0;
}
```

第二节 switch 语句

1. 晶晶赴约会【1.4 编程基础之逻辑表达式与条件分支 11】

晶晶的朋友贝贝约晶晶下周一起去看展览,但晶晶每周的一、三、五有课必须上课,请帮晶晶判断她能否接受贝贝的邀请。如果能,输出 YES;如果不能,则输出 NO。注意 YES 和 NO 都是大写字母!

输入:

一行,贝贝邀请晶晶去看展览的日期,用数字 1~7 表示从星期一到星期日。

输出:

一行,若晶晶可以接受贝贝的邀请,则输出 YES,否则输出 NO。注意 YES 和 NO 都是大写字母!

样例输入:

2

样例输出:

YES

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n;
    scanf("%d",&n);           //输入 1 个整数,即贝贝邀请晶晶去看展览的日期
    if (n!=1 && n!=3 && n!=5) printf(" YES "); //若 n=1,3 或 5,则晶晶不接受邀请
    else printf(" NO ");       //否则晶晶可以接受贝贝的邀请
    return 0;                  //注意 YES 和 NO 都是大写字母!
}
```

2. 骑车与走路【1.4 编程基础之逻辑表达式与条件分支 12】

在清华校园里,若没有自行车,上课办事会很不方便。但实际上,并非去办任何事情都是骑车快,因为骑车总要找车、开锁、停车、锁车等,这要耽误一些时间。假设找到自行车,开锁并骑上自行车的时间为 27 秒;停车锁车的时间为 23 秒;步行每秒行走 1.2 米,骑车每秒行走 3.0 米。请判断走不同的距离去办事是骑车快还是走路快。如果骑车快,那么输出“Bike”;如果走路快,那么输出“Walk”;如果一样快,那么输出“All”。

输入:

一行,包含一个整数,表示一次办事要行走的距离,单位为米。

输出:

一行,如果骑车快,那么输出“Bike”;如果走路快,那么输出“Walk”;如果一样快,那么输出“All”。

样例输入:

120

样例输出:

Bike

【参考程序】

```
#include<cstdio>
using namespace std;
```

```

int main()
{
    double a,b,n;
    scanf("%lf",&n);           //输入给定 1 个整数,即一次办事要行走的距离
    a=n/3.0+27+23; b=n/1.2;    //a 与 b 分别储存骑车与走路的所需时间
    if (a<b) printf(" Bike\n"); //若 a<b,则骑车快,输出"Bike"
    if (a==b) printf(" All\n"); //若 a=b,则一样快,输出"All"
    if (a>b) printf(" Walk\n"); //若 a>b,则走路快,输出"Walk"
    return 0;
}

```

3. 分段函数【1.4 编程基础之逻辑表达式与条件分支 13】

编写程序,计算下列分段函数 $y=f(x)$ 的值。结果保留到小数点后三位。

$y = -x + 2.5;$	$0 \leq x < 5$
$y = 2 - 1.5(x-3)(x-3);$	$5 \leq x < 10$
$y = x/2 - 1.5;$	$10 \leq x < 20$

输入:

一个浮点数 $n(0 \leq n < 20)$ 。

输出:

n 对应的分段函数值: $f(n)$ 。结果保留到小数点后三位。

样例输入:

1.0

样例输出:

1.500

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    double x,y;
    scanf("%lf",&x);           //输入给定 1 个浮点数 n
    if (0<=x&& x<5) y=-x+2.5; //如果 0<=x<5,则 y=-x+2.5
    else if (5<=x&& x<10) y=2-1.5*(x-3)*(x-3);
                                //否则如果 5<=x<10,则 y=2-1.5*(x-3)*(x-3)
    else if (10<=x&& x<20) y=x/2.0-1.5;
                                //否则如果 10<=x<20,则 y=x/2.0-1.5
    printf("%.3lf",y);          // 结果保留到小数点后三位
    return 0;
}

```

4. 计算邮资【1.4 编程基础之逻辑表达式与条件分支 14】

根据邮件的质量和用户是否选择加急计算邮费。计算规则:质量在 1000 克以内(包括 1000 克),基本费 8 元。超过 1000 克的部分,每 500 克加收超重费 4 元,不足 500 克部分按 500 克计算;如果用户选择加急,多收 5 元。

输入:

一行,包含整数和一个字符,以一个空格分开,分别表示质量(单位为克)和是否加急。如果字符是 y,说明选择加急;如果字符是 n,说明不加急。

输出:

一行,包含一个整数,表示邮费。

样例输入:

1200 y

样例输出:

17

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    int k,s=8; char w;           //int s=8;等同于 int s;s=8;
    cin>>k>>w;                  //输入给定整数与字符,即质量和是否加急
    if (k>1000)
        if ((k-1000)%500!=0) s+=((k-1000)/500+1)*4;
        else s+=(k-1000)/500*4;  //计算邮件质量的费用,储存于 s
    if (w=='y') s+=5;            //判断用户是否加急
    printf("%d\n",s);
    return 0;
}
```

5. 最大数输出【1.4 编程基础之逻辑表达式与条件分支 15】

输入三个整数,数与数之间以一个空格分开。输出一个整数,即最大的整数。

输入:

一行,包含三个整数,数与数之间以一个空格分开。

输出:

一行,包含一个整数,即最大的整数。

样例输入:

10 20 56

样例输出:

56

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    if ((a>=b)&&(a>=c)) printf("%d",a);    //如果 a 最大,则输出 a
    else if ((b>=a)&&(b>=c)) printf("%d",b); //否则如果 b 最大,则输出 b
    else printf("%d",c);                  //否则输出 c
    return 0;
}
```

6. 三角形判断【1.4 编程基础之逻辑表达式与条件分支 16】

给定三个正整数,分别表示三条线段的长度,判断这三条线段能否构成一个三角形。如果能构成三角形,则输出“yes”,否则输出“no”。

输入:

一行,包含三个正整数,分别表示三条线段的长度,数与数之间以一个空格分开。

输出:

如果能构成三角形,则输出“yes”,否则输出“no”。

样例输入:

3 4 5

样例输出:

yes

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c); //输入给定 3 个整数,即三条线段的长度
    if (a+b>c && a+c>b && b+c>a) printf(" yes ");
    else printf(" no ");      //如果三条线段能构成三角形,则输出 yes,否则输出 no
    return 0;
}
```

7. 判断闰年【1.4 编程基础之逻辑表达式与条件分支 17】

判断某年是否是闰年。如果公元 a 年是闰年输出 Y,否则输出 N。

输入:

一行,包含一个整数 a($0 < a < 3000$)。

输出:

一行,如果公元 a 年是闰年输出 Y,否则输出 N。

样例输入:

2006

样例输出:

N

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int n;
int main()
{
    scanf("%d",&n); //输入给定一个整数,即公元 n 年
    if (n%4==0&& n%100!=0|| n%400==0&& n%3200!=0) printf(" Y ");
    //如果公元 n 年是闰年,则输出“Y”
    else printf(" N "); //否则输出“N”
    return 0;
}
```

8. 点和正方形的关系【1.4 编程基础之逻辑表达式与条件分支 18】

有一个正方形,四个角的坐标(x,y)分别是(1,-1),(1,1),(-1,-1),(-1,1),x 是横坐标,y 是纵坐标。写一个程序,判断一个给定的点是否在这个正方形内(包括正方形边界)。如果点在正方形内,则输出 yes,否则输出 no。

输入:

一行,包含两个整数 x,y,以一个空格分开,表示坐标(x,y)。

输出:

一行,如果点在正方形内,则输出 yes,否则输出 no。

样例输入:

1 1

样例输出:

yes

【参考程序】

```
#include<iostream>
#include<cstdio>
int main()
{
    int x,y;
    scanf("%d%d",&x,&y);           //输入给定 2 个整数,即点的坐标(x,y)
    if (x>=-1 && x<=1 && y>=-1 && y<=1) printf("yes");
                                   //如果点在正方形内,输出“yes”
    else printf("no");              //否则输出“no”
    return 0;
}
```

9. 简单计算器【1.4 编程基础之逻辑表达式与条件分支 19】

一个最简单的计算器,支持+,−,*,/ 四种运算。仅需考虑输入输出为整数的情况,数据和运算结果不会超过 int 表示的范围。然而:

(1)如果出现除数为 0 的情况,则输出:Divided by zero!;

(2)如果出现无效的操作符(即不为+,-,*,/之一),则输出:Invalid operator!。

输入:

一行,共有 3 个参数,其中第 1、2 个参数为整数,第 3 个参数为操作符(+,-,*,/)。

输出:

一行,一个整数,是运算结果。然而:

(1)如果出现除数为 0 的情况,则输出:Divided by zero!;

(2)如果出现无效的操作符(即不为+,-,*,/之一),则输出:Invalid operator!。

样例输入:

1 2 +

样例输出

3

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int a,b,n; char x;
    cin>>a>>b>>x;
    if ((b==0)&&(x=='/')) //如果出现除数为 0 的情况,输出“Divided by zero!”
        cout<<" Divided by zero! "<<endl;
    else if ((x=='+')||(x=='-')||(x=='*')||(x=='/')) //否则如果 x 为有效操作符
    {
        switch (x) //判断 x 的值
        {
            case '+':n=a+b;break; //如果 x="+",n=a+b,退出
            case '-':n=a-b;break; //如果 x="-",n=a-b,退出
            case '*':n=a*b;break; //如果 x="*",n=a*b,退出
            case '/':n=a/b;break; //如果 x="/",n=a/b,退出
        }
    }
}
```

```

    }
    cout<<n<<endl;
}
else if ((x! = '+') || (x! = '/') || (x! = '-') || (x! = '*'))
    cout<<" Invalid operator! "<<endl;
    // 否则如果 x 为无效操作符,则输出“Invalid operator!”
}

```

10. 求一元二次方程的根【1.4 编程基础之逻辑表达式与条件分支 20】

利用公式 $x_1 = (-b + \sqrt{b^2 - 4ac}) / (2a)$, $x_2 = (-b - \sqrt{b^2 - 4ac}) / (2a)$, 求一元二次方程 $ax^2 + bx + c = 0$ 的根, 其中 a 不等于 0。结果要求精确到小数点后 5 位。

输入:

一行, 包含三个浮点数 a , b , c (它们之间以一个空格分开), 分别表示方程 $ax^2 + bx + c = 0$ 的系数。

输出:

一行, 是方程的根。

若两个实根相等, 则输出形式为: $x_1 = x_2 = \dots$

若两个实根不等, 则输出形式为: $x_1 = \dots; x_2 = \dots$, 其中若是两个虚根, 则输出: $x_1 = \text{实部} + \text{虚部}i$; $x_2 = \text{实部} - \text{虚部}i$, 其中 x_1, x_2 满足以下两个条件中的一个:

(1) x_1 的实部大于 x_2 的实部;

(2) x_1 的实部等于 x_2 的实部且 x_1 的虚部大于等于 x_2 的虚部。

所有实数部分要求精确到小数点后 5 位, 数字、符号之间没有空格。

样例输入:

1.0 2.0 8.0

样例输出:

$x_1 = -1.00000 + 2.64575i; x_2 = -1.00000 - 2.64575i$

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<math.h>
using namespace std;
double a,b,c,x1,x2,s,x;
int main()
{
    cin>>a>>b>>c;
    x1=(-b+sqrt(b*b-4*a*c))/(2*a); //求第一个实根的值
    x2=(-b-sqrt(b*b-4*a*c))/(2*a); //求第二个实根的值
    s=-b/(2*a); //求实部的值
    if(-b==0) s=0; //若被除数等于0,需特判 s=-0 的情况,将 s 的值改为 0
    x=sqrt(4*a*c-b*b)/(2*a); //求虚部的值
    if(b*b==4*a*c) printf("x1=x2=%.5f",x1);
        //若 b*b=4*a*c,则两个实根相等
    if(b*b>4*a*c) printf("x1=%.5f;x2=%.5f",x1,x2);
        //若 b*b>4*a*c,则输出两个实根
    if(b*b<4*a*c) printf("x1=%.5f+%.5fi;x2=%.5f-%.5fi",s,x,s,x);
        //若 b*b<4*a*c,则输出实部+虚部 i 和实部-虚部 i
}

```

第四章 循环结构

第一节 for 语句

1. 求平均年龄【1.5 编程基础之循环控制 01】

班上有学生若干名,给出每名学生的年龄(整数),求班上所有学生的平均年龄,保留到小数点后两位。

输入:

第 1 行有一个整数 $n(1 \leq n \leq 100)$,表示学生的人数;其后 n 行每行有 1 个整数,表示每个学生的年龄,取值为 15~25。

输出:

一行,包含一个浮点数,为所求的平均年龄,保留到小数点后两位。

样例输入:

2
18
17

样例输出:

17.50

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n,t;
    double s;
    s=0; //s 储存班上同学年龄之和,初始值赋为 0
    scanf("%d",&n);
    for (int i=1;i<=n;++i) //循环累加班上的 n 名学生的年龄
    {
        scanf("%d",&t);
        s+=t;
    }
    printf("%.2lf",s/n); //输出班上同学年龄的平均值,精确到小数点后两位
    return 0;
}
```

2. 均值【1.5 编程基础之循环控制 03】

给出一组样本数据,包含 n 个浮点数,计算其均值,精确到小数点后 4 位。

输入:

有两行,第 1 行包含一个整数 $n(n$ 小于 100),代表样本容量;第 2 行包含 n 个绝对值不超过 1000 的浮点数,代表各个样本数据。

输出:

一行,包含一个浮点数,表示均值,精确到小数点后 4 位。

样例输入:

2

1.0 3.0

样例输出:

2.0000

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n;
    double s,t;
    s=0; //s 储存一组浮点数之和,初始值赋为 0
    scanf("%d",&n);
    for (int i=1;i<=n;++i) //循环累加这组样本的 n 个浮点数
    {
        scanf("%lf",&t);
        s+=t;
    }
    printf("%.4lf",s/n); //输出这组浮点数的均值,精确到小数点后 4 位
    return 0;
}
```

3. 求整数的和与均值【1.5 编程基础之循环控制 04】

读入 $n(1 \leq n \leq 10000)$ 个整数,求它们的和与均值。

输入:

第 1 行是一个整数 n ,表示有 n 个整数;第 2~ $n+1$ 行每行包含 1 个整数。每个整数的绝对值均不超过 10,000。

输出:

一行,先输出和,再输出平均值(保留到小数点后 5 位),两个数间用单个空格分隔。

样例输入:

4

344

222

343

222

样例输出:

1131 282.75000

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    double n,s,t;
    scanf("%lf",&n);
    s=0; //s 储存 n 个整数之和,初始值赋为 0
```

```

for (int i=1;i<=n;++i)    //循环累加 n 个整数
{
    scanf("%lf",&t);
    s+=t;
}
printf("%.0lf %.5lf\n",s,s/n);
//输出 n 个整数之和与平均值,整数之和保留整数,平均值保留到小数点后 5 位
return 0;
}

```

4. 最高的分数【1.5 编程基础之循环控制 05】

孙老师讲授的《计算概论》这门课期中考试刚刚结束,他想知道考试中取得的最高分数。因为人数比较多,他觉得这件事情交给计算机来做比较方便。你能帮孙老师解决这个问题吗?

输入:

两行,第 1 行为整数 n ($1 \leq n \leq 100$),表示参加这次考试的人数;第 2 行是这 n 个学生的成绩,相邻两个数之间用单个空格隔开。所有成绩均为 $0 \sim 100$ 之间的整数。

输出:

输出一个整数,即最高的成绩。

样例输入:

```

5
85 78 90 99 60

```

样例输出:

```

99

```

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    long a,n,max=-1000000;//max 储存考试中取得的最高成绩,初始值赋为-1000000
    scanf("%ld",&n);
    for (int i=1; i<=n; i++)//输入所有成绩,找最高成绩
    {
        scanf("%ld",&a);
        if (a>max) max=a;    //如果当前成绩最高,更新 max 中储存的最高成绩
    }
    printf("%ld",max);
    return 0;
}

```

5. 最大跨度值【1.5 编程基础之循环控制 06】

给定一个长度为 n 的非负整数序列,请计算序列的最大跨度值(最大跨度值=最大值减去最小值)。

输入:

一共两行,第 1 行为序列的个数 n ($1 \leq n \leq 1000$),第 2 行为序列的 n 个不超过 1000 的非负整数,整数之间以一个空格分隔。

输出:

一行,表示序列的最大跨度值。

样例输入:

```
6
3 0 8 7 5 9
```

样例输出:

```
9
```

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int n,i,x,max=-1000000,min=1000000;//max 储存序列中的最大值,min 储存最小值
    scanf("%d",&n);
    for(i=1;i<=n;++i)                //循环寻找序列中的最大值与最小值
    {
        scanf("%d",&x);
        if (x>max) max=x; //如果当前值大于之前的最大值,更新 max 中储存的最大值
        if (x<min) min=x; //如果当前值小于之前的最小值,更新 min 中储存的最小值
    }
    printf("%d\n",max-min);          //输出序列的最大跨度值
    return 0;
}
```

6. 奥运奖牌计数【1.5 编程基础之循环控制 07】

2008 年北京奥运会,A 国的运动员参与了 n 天的决赛项目($1 \leq n \leq 17$)。现在要统计一下 A 国所获得的金、银、铜牌数目及总奖牌数。输入第 1 行是 A 国参与决赛项目的天数 n , 其后 n 行,每一行是该国某一天获得的金、银、铜牌数目。输出 4 个整数,为 A 国所获得的金、银、铜牌总数及总奖牌数。

输入:

有 $n+1$ 行,第 1 行是 A 国参与决赛项目的天数 n ,其后 n 行,每一行是该国某一天获得的金、银、铜牌数目,数目之间以一个空格分开。

输出:

只有一行,包括 4 个整数,为 A 国所获得的金、银、铜牌总数及总奖牌数,数目之间以一个空格分开。

样例输入:

```
3
1 0 3
3 1 0
0 3 0
```

样例输出:

```
4 4 3 11
```

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int n,i,a,b,c,x=0,y=0,z=0,s=0;
    //x 存 A 国所获得的金牌数,y 存银牌数,z 存铜牌数,s 存总奖牌数,初始值赋为 0
    scanf("%d",&n);
```

```

for(i=1;i<=n;i++)          //循环累加 A 国每天所获的金、银、铜牌数
{
    scanf("%d%d%d",&a,&b,&c);
    x+=a; y+=b; z+=c;
}
s=x+y+z;                    //s 储存 A 国所获得的总奖牌数
printf("%d %d %d %d\n",x,y,z,s);
return 0;
}

```

7. 奇数求和【1.5 编程基础之循环控制 09】

计算非负整数 m 到 n (包括 m 和 n) 之间的所有奇数的和, 其中, m 不大于 n , 且 n 不大于 300。例如 $m=3, n=12$, 其和则为: $3+5+7+9+11=35$ 。

输入:

两个数 m 和 n ($0 \leq m \leq n \leq 300$), 两个数以一个空格分开。

输出:

一行, 包含一个整数, 表示 m 到 n (包括 m 和 n) 之间的所有奇数的和。

样例输入:

7 15

样例输出:

55

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    long m,n,s=0;                //s 储存 m 到 n 之间所有奇数之和, 初始值赋为 0
    scanf("%ld%ld",&m,&n);
    for (int i=m; i<=n; i++)      //循环寻找 m 到 n 之间所有的奇数
        if (i%2!=0) s=s+i;       //如果 i 为奇数, 则累加
    printf("%ld",s);
    return 0;
}

```

8. 满足条件的数累加【1.5 编程基础之循环控制 10】

将正整数 m 和 n 之间 (包括 m 和 n) 能被 17 整除的数累加, 其中 $0 < m < n < 1000$ 。

输入:

一行, 包含两个整数 m 和 n , 其间以一个空格间隔。

输出:

一行, 包含一个整数, 表示累加的结果。

样例输入:

50 85

样例输出:

204

【参考程序】

```

#include <cstdio>
using namespace std;
int main()

```

```

{
    int m,n,i,s=0;           //s 储存能被 17 整除的数之和,初始值赋为 0
    scanf("%d%d",&m,&n);
    for (i=m;i<=n;++i)       //循环寻找 m 与 n 之间所有能被 17 整除的数
        if (i%17==0) s+=i;   //如果 i 能被 17 整除,则累加
    printf("%d\n",s);        //输出累加的结果
    return 0;
}

```

9. 整数的个数【1.5 编程基础之循环控制 11】

给定 $k(1 < k < 100)$ 个正整数,其中每个数都是大于等于 1,小于等于 10 的数。写程序计算给定的 k 个正整数中,1,5 和 10 出现的次数。

输入:

有两行,第 1 行包含一个正整数 k ,第 2 行包含 k 个正整数,每两个正整数用一个空格分开。

输出:

输出有三行,第 1 行为 1 出现的次数,第 2 行为 5 出现的次数,第 3 行为 10 出现的次数。

样例输入:

```

5
1 5 8 10 5

```

样例输出:

```

1
2
1

```

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    //y 储存 n 个正整数中 1 出现的次数,w 储存 5 的次数,s 储存 10 的次数
    int a,y=0,w=0,s=0,n;
    scanf("%d",&n);
    for (int i=1;i<=n;++i)    //读入 n 个正整数
    {
        scanf("%d",&a);
        if (a==1) ++y;        //如果 a=1,y 累加
        if (a==5) ++w;        //如果 a=5,w 累加
        if (a==10) ++s;       //如果 a=10,s 累加
    }
    printf("%d\n%d\n%d\n",y,w,s);
    return 0;
}

```

10. 与指定数字相同的数的个数【1.5 编程基础之循环控制 12】

输出一个整数序列中与指定数字相同的数的个数。输入包含 2 行:第 1 行为 n 和 m ,表示整数序列的长度($n \leq 100$)和指定的数字;第 2 行为 n 个整数,整数之间以一个空格分开。输出为 n 个数中与 m 相同的数的个数。

输入:

第 1 行为 $n(n \leq 100)$ 和 m , 表示整数序列的长度和指定的数字, 中间用一个空格分开;

第 2 行为 n 个整数, 整数之间以一个空格分开。

输出:

只有一行, 为 n 个数中与 m 相同的数的个数。

样例输入:

3 2
2 3 2

样例输出:

2

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int n, m, i, k = 0, x;           //k 储存与 m 相同的数的个数, 初始值赋为 0
    scanf("%d%d", &n, &m);
    for(i = 1; i <= n; ++i)         //循环寻找这个整数序列中与 m 相同的数
    {
        scanf("%d", &x);
        if (x == m) k++;           //如果 x 与 m 相等, 则 k 累加
    }
    printf("%d\n", k);
    return 0;
}
```

11. 乘方计算【1.5 编程基础之循环控制 13】

给出一个整数 a 和一个正整数 $n(-1000000 \leq a \leq 1000000, 1 \leq n \leq 10000)$, 求乘方 a^n , 即乘方结果。最终结果的绝对值不超过 1000000。

输入:

一行, 包含两个整数 a 和 $n(-1000000 \leq a \leq 1000000, 1 \leq n \leq 10000)$, 两个数之间以单个空格隔开。

输出:

一个整数, 即乘方结果。题目保证最终结果的绝对值不超过 1,000,000。

样例输入:

2 3

样例输出:

8

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int a, n, s = 1;                //s 储存乘方结果, 初始值赋为 1
    scanf("%d%d", &a, &n);
    for (int i = 1; i <= n; ++i)    //循环累乘  $a^n$ 
    {
        s *= a;                    //s 储存  $a^i$ 
    }
    printf("%d", s);
    return 0;
}
```

12. 人口增长【1.5 编程基础之循环控制 14】

我国现有 x 亿人口,按照每年 0.1% 的增长速度, n 年后将有多少人? 结果保留小数点后四位。

输入:

一行,包含两个整数 x 和 n ($1 \leq x \leq 100, 1 \leq n \leq 100$),分别是人口基数和年数,以单个空格分隔。

输出:

输出最后的人口数,以亿为单位,保留到小数点后四位。

样例输入:

13 10

样例输出:

13.1306

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    double x,n;
    scanf("%lf%lf",&x,&n);
    double s=x;           //s 储存 n 年后的人口数,初始值赋为当前人口数
    for (int i=1; i<=n; ++i) //循环计算 n 年后的人口数
        s *= 1.001;         //s 储存到 i 年为止的人口数
    printf("%.4lf",s);
    return 0;
}
```

13. 菲波那契数【1.5 编程基础之循环控制 17】

菲波那契数列是指这样的数列:数列的第一个和第二个数都为 1,接下来每个数都等于前面两个数之和。给出一个正整数 k ,要求菲波那契数列中第 k 个数是多少。

输入:

一行,包含一个正整数 k ($1 \leq k \leq 46$)。

输出:

一行,包含一个正整数,表示菲波那契数列中第 k 个数的大小。

样例输入:

19

样例输出:

4181

【参考程序】

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    int a,b,c,k,s;
    scanf("%d",&k);
    a=1; b=1;           //a,b 初始值赋为数列的第 1、2 个数
    for(int i=3; i<=k; ++i) //循环计算到第 k 个数
        c=a+b;
    printf("%d",c);
}
```

```

{
    c=a+b; a=b; b=c;          //c 储存第 i 个数,每个数都等于前面 2 个数之和
}
printf("%d\n",b);
return 0;
}

```

14. 鸡尾酒疗法【1.5 编程基础之循环控制 18】

鸡尾酒疗法,指“高效抗逆转录病毒治疗”。人们在鸡尾酒疗法的基础上又提出了很多种改进的疗法。为了验证这些治疗方法是否在疗效上比鸡尾酒疗法更好,可用通过临床对照实验的方式进行。假设鸡尾酒疗法的有效率为 x ,新疗法的有效率为 y ,如果 $y-x$ 大于 5%,则效果更好,如果 $x-y$ 大于 5%,则效果更差,否则称为效果差不多。下面给出 n 组临床对照实验,其中第一组采用鸡尾酒疗法,其他 $n-1$ 组为各种不同的改进疗法。请写程序判定各种改进疗法效果如何。

输入:

第一行为整数 $n(1 \leq n \leq 20)$;其余 n 行每行两个整数,第一个整数是临床实验的总病例数(小于等于 10,000),第二个疗效有效的病例数。这 n 行数据中,第一行为鸡尾酒疗法的数据,其余各行为各种改进疗法的数据。

输出:

有 $n-1$ 行,分别表示对应改进疗法的效果:如果效果更好,输出 better;如果效果更差,输出 worse;否则输出 same。

样例输入:

```

5
125 99
112 89
145 99
99 97
123 98

```

样例输出:

```

same
worse
better
same

```

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    long a,b,n;
    double x,y;
    scanf("%ld",&n);
    scanf("%ld%ld",&a,&b);
    x=b/(a/1.0);          //x 储存原鸡尾酒疗法的有效率
    for (int i=2; i<=n; i++) //循环判断其他 n-1 种改进疗法与原鸡尾酒疗法的差距
    {
        scanf("%ld%ld",&a,&b);
        y=b/(a/1.0);      //y 储存第 i 种改进疗法的有效率
        if (y-x>0.05) printf(" better\n");
    }
}

```



```

    else if (x-y>0.05) printf(" worse\n ");
    else printf(" same\n "); //判断第 i 种改进疗法与原鸡尾酒疗法的差距
}
return 0;
}

```

15. 救援【1.5 编程基础之循环控制 19】

救生船从大本营出发,营救若干屋顶上的人回到大本营,屋顶数目以及每个屋顶的坐标和人数都将由输入决定,求出所有人都到达大本营并登陆所用的时间。

直角坐标系的原点是大本营,救生船每次从大本营出发,救了人之后将人送回大本营。坐标系中的点代表屋顶,每个屋顶由其位置坐标和其上的人数表示。救生船每次从大本营出发,以速度 50 米/分钟驶向下一个屋顶,达到一个屋顶后,救下其上的所有人,每人上船 1 分钟,船原路返回,达到大本营,每人下船 0.5 分钟。假设原点与任意一个屋顶的连线不穿过其他屋顶。

输入:

第 1 行,一个整数,表示屋顶数 n 。接下来依次有 n 行输入,每一行中包含两个表示屋顶相对于大本营的平面坐标位置的实数(单位是米)、一个表示人数的整数。

输出:

救援需要的总时间,精确到分钟(向上取整)。

样例输入:

```

1
30 40 3

```

样例输出:

```

7

```

【参考程序】

```

#include <stdio>
#include <cmath>
using namespace std;
int main()
{
    double a,b,s=0,n,c,l; //s 储存救援需要的总时间,初始值赋为 0
    scanf("%lf",&n);
    for(int i=1;i<=n;i++) //循环每一个屋顶
    {
        scanf("%lf%lf%lf",&a,&b,&c);
        l=sqrt(1.0*(a*a)+1.0*(b*b));
        s=s+2*l/50.0+c*1.5; //s 累加救援这个屋顶所需要的时间
    }
    printf("%.0lf\n",ceil(s));
    return 0;
}

```

16. 津津的储蓄计划【1.5 编程基础之循环控制 22】Noip2012 提高组第 1 题

津津的零花钱一直都是自己管理。每个月的月初妈妈给津津 300 元钱,津津会预算这个月的花销,并且总能做到实际花销和预算的相同。

为了让津津学习如何储蓄,妈妈提出,津津可以随时把整百的钱存在她那里,到了年末她会加上 20%还给津津。因此津津制订了一个储蓄计划:每个月的月初,在得到妈妈给的零花钱后,如果她预计到这个月的月末手中还会有多于 100 元或恰好 100 元,她就会把整百的

钱存在妈妈那里,剩余的钱留在自己手中。

例如,11月初津津手中还有83元,妈妈给了津津300元。津津预计11月的花销是180元,那么她就会在妈妈那里存200元,自己留下183元。到了11月月末,津津手中会剩下3元钱。

现在请你根据2004年1月到12月每个月津津的预算,判断会不会出现这种情况。如果不会,计算到2004年年末,妈妈将津津平常存的钱加上20%还给津津之后,津津手中会有多少钱。

输入:

包括12行数据,每行包含一个小于350的非负整数,分别表示1月到12月津津的预算。

输出:

只包含一个整数。如果储蓄计划实施过程中出现某个月钱不够用的情况,输出一X,X表示出现这种情况的第1个月;否则输出到2004年年末津津手中会有多少钱。

样例输入:

样例 #1:

290
230
280
200
300
170
340
50
90
80
200
60

样例 #2:

290
230
280
200
300
170
330
50
90
80
200
60

样例输出:

样例 #1:

-7

样例 #2:

1580

【参考程序】

```
#include <iostream>
```

```

#include <cstdio>
#include <cstdlib>
using namespace std;
int x; int i, n, t; bool f=1;
int main()
{
    t=0; n=0; //t 存津津留在手中的钱, n 存津津存在妈妈手上的钱, 初始值均赋为 0
    for(i=1; i<=12; ++i) //循环计算这一年中钱的变化
    {
        scanf("%d", &x);
        t=t+300-x;
        if (t<0) //如果在第 i 个月津津手上的钱不够了, 输出“-i”
        {
            printf("-%d\n", i);
            f=0;
            break;
        }
        if (t>=100) //剩余的钱不少于 100 元
        {
            n=n+t/100; t%=100;
        } //把整百的钱存在妈妈那里, 剩余的钱留在自己手中
    }
    if (f==1) printf("%d\n", n*120+t); //妈妈将津津平常存的钱加上 20% 还给津津
    return 0;
}

```

17. 药房管理【1.5 编程基础之循环控制 23】

随着信息技术的蓬勃发展, 医疗信息化已经成为医院建设中必不可少的一部分。计算机可以很好地辅助医院管理医生信息、病人信息、药品信息等海量数据, 使工作人员能够从这些机械的工作中解放出来, 将更多精力投入真正的医疗过程中, 从而极大地提高了医院整体的工作效率。

对药品的管理是其中的一项重要内容。现在药房的管理员希望使用计算机来帮助他管理。假设对于任意一种药品, 每天开始工作时的库存总量已知, 并且一天之内不会通过进货的方式增加。每天会有很多病人前来取药, 每个病人希望取走不同数量的药品。如果病人需要的数量超过了当时的库存量, 药房会拒绝该病人的请求。管理员希望知道每天会有多少病人没有取上药。

输入:

共三行, 第 1 行是每天开始时的药品总量 m ; 第 2 行是这一天取药的人数 n ($0 < n \leq 100$); 第 3 行共有 n 个数, 分别记录了每个病人希望取走的药品数量 (按照时间先后的顺序)。

输出:

只有一行, 为这一天没有取上药品的人数。

样例输入:

```

30
6
10 5 20 6 7 8

```

样例输出:

```

2

```

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int m,n,i,x,k=0;           //k 储存这一天没有取上药品的人数,初始值赋为 0
    scanf("%d",&m);
    scanf("%d",&n);
    for (i=1; i<=n; ++i)       //循环判断每个病人取药的情况
    {
        scanf("%d",&x);
        if (m>=x) m-=x; else ++k; //判断第 i 个人能否取上药
    }
    printf("%d",k);
    return 0;
}
```

18. 正常血压【1.5 编程基础之循环控制 24】

监护室每小时测量一次病人的血压,若收缩压在 90~140 并且舒张压在 60~90(包含端点值)则称之为正常,现给出某病人若干次测量的血压值,计算病人保持正常血压的最长小时数。

输入:

第 1 行为一个正整数 $n(n < 100)$,其后有 n 行,每行 2 个正整数,分别为一次测量的收缩压和舒张压。

输出:

仅一行,血压连续正常的最长小时数。

样例输入:

```
4
100 80
90 50
120 60
140 90
```

样例输出:

```
2
```

【参考程序】

```
#include <cstdio>
using namespace std;
int main()
{
    int a,b,i,n,t,m,j,max=0;   //max 储存病人血压连续正常的最长小时数
    scanf("%d",&n); t=0;
    for(i=1;i<=n;++i)         //循环判断每个小时中,病人的血压是否正常
    {
        scanf("%d%d",&a,&b);
        if (a>=90&&a<=140&&b>=60&&b<=90)
            //判断在第 i 个小时中,病人的血压是否正常
        {
            t++;               //如果大于目前病人血压连续正常的最长小时数,则替换 max 的值
        }
    }
}
```

```

        if (t > max) max = t;
    }
    else t = 0;           //这个小时病人血压不正常,则清零 t
}
printf("%d\n", max);
return 0;
}

```

19. 统计满足条件的 4 位数【1.5 编程基础之循环控制 26】

给定若干个四位数,求出其中满足以下条件的数的个数:个位上的数字减去千位上的数字,再减去百位上的数字,再减去十位上的数字的结果大于零。

输入:

两行,第 1 行为四位数的个数 n ($n \leq 100$),第 2 行为 n 个的四位数。

输出:

一行,包含一个整数,表示满足条件的四位数的个数。

样例输入:

```

5
1234 1349 6119 2123 5017

```

样例输出:

```

3

```

【参考程序】

```

#include <cstdio>
using namespace std;
int main()
{
    int a, b, c, d, n, m, i, s = 0; //s 为这 n 个四位数中满足条件的数的总数,初始值赋为 0
    scanf("%d", &n);
    for(i = 1; i <= n; ++i) //循环判断每个四位数是否满足条件
    {
        scanf("%d", &m);
        a = m / 1000;
        b = (m / 100) % 10;
        c = (m / 10) % 10;
        d = m % 10;
        if (d - a - b - c > 0) s++; //如果这个四位数满足条件,s 累加
    }
    printf("%d\n", s);
    return 0;
}

```

20. 求分数序列和【1.5 编程基础之循环控制 32】

有一个分数序列 $q_1/p_1, q_2/p_2, q_3/p_3, q_4/p_4, q_5/p_5, \dots$, 其中 $q(i+1) = q_i + p_i$, $p(i+1) = q_i$, $p_1 = 1$, $q_1 = 2$ 。比如这个序列前 6 项分别是 $2/1, 3/2, 5/3, 8/5, 13/8, 21/13$ 。求这个分数序列的前 n 项之和。

输入:

一行,包含一个正整数 n ($n \leq 30$)。

输出:

一行,包含一个浮点数,表示分数序列前 n 项的和,精确到小数点后 4 位。

样例输入：

2

样例输出：

3.5000

【参考程序】

```
#include<cstdio>
#include<cmath>
using namespace std;
int main()
{
    double a=1,b=1,c,i,j,s=0; int n; //s 储存分数序列前 n 项之和,初始值赋为 0
    scanf("%d",&n);
    for(i=1;i<=n;++i) //循环累加前 n 项分数的和
    {
        c=a+b; a=b; b=c;
        s=s+(double)b/a; //s 储存前 i 项分数之和
    }
    printf("%.4lf\n",s);
    return 0;
}
```

21. 计算分数加减表达式的值【1.5 编程基础之循环控制 33】

编写程序,输入 n 的值,求算式 $1/1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 - 1/8 + \dots + (-1)^{n-1} \cdot 1/n$ 的值。

输入：

一个正整数 n ($1 \leq n \leq 1000$)。

输出：

一个实数,为表达式的值,保留到小数点后四位。

样例输入：

2

样例输出：

0.5000

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n; double s=0; //s 储存表达式的值,初始值赋为 0
    scanf("%d",&n);
    for (int i=1;i<=n;++i) //循环计算表达式的值
    {
        if (i%2==0) s-=1.0/i;
        else s+=1.0/i; //s 储存 n=i 时表达式的值
    }
    printf("%.4f",s);
    return 0;
}
```

22. 余数相同问题【小学奥数 7647】

已知三个正整数 a, b, c 。现有一个大于 1 的整数 x , 将其作为除数分别除 a, b, c , 得到的余数相同。请问满足上述条件的 x 的最小值是多少? 数据保证 x 有解。

输入:

一行, 三个不大于 1,000,000 的正整数 a, b, c , 每个整数之间用一个空格隔开。

输出:

一个整数, 即满足条件的 x 的最小值。

样例输入:

300 262 205

样例输出:

19

【参考程序】

```
#include<cstdio>
#include<cstdlib> //使用 exit 须调用 cstdlib 库
using namespace std;
int main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    for (int i=2; i<=1000000; i++) //循环寻找满足条件的 x 的最小值
        if ((a%i==b%i)&&(b%i==c%i)&&(c%i==a%i)) //判断 i 是否满足条件
        {
            printf("%d",i);
            exit(0); //满足条件则输出,结束
        }
    return 0;
}
```

23. 分苹果【小学奥数 7826】

把一堆苹果分给 n 个小朋友, 要使每个人都能拿到苹果, 而且每个人拿到的苹果数都不同的话, 这堆苹果至少应该有多少个?

输入:

一个不大于 1000 的正整数 n , 代表小朋友人数。

输出:

一个整数, 表示满足条件的最少苹果个数。

样例输入:

8

样例输出:

36

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n,s=0; //s 储存满足条件的最少苹果个数, 初始值赋为 0
    scanf("%d",&n);
    for (int i=1; i<=n; ++i) //循环计算满足条件的最少苹果个数
```

```

    s += i; //s 储存当 n=i 时, 满足条件的最少苹果个数
    printf("%d\n", s);
    return 0;
}

```

24. 求小数的某一位数字【小学奥数 7830】

分数 a/b 化为小数后, 小数点后第 n 位的数字是多少?

输入:

三个正整数 a, b, n ($0 < a < b < 100, 1 \leq n \leq 10,000$), 相邻两个数之间用单个空格隔开。

输出:

一个数字。

样例输入:

1 2 1

样例输出:

5

【参考程序】

```

#include <iostream>
using namespace std;
int main()
{
    int a, b, n, t;
    cin >> a >> b >> n;
    for (int i = 1; i <= n; ++i) //循环计算分数 a/b 化为小数后, 小数点后第 n 位的数字
    {
        a *= 10; //模拟除法竖式计算
        t = a / b; //t 储存分数 a/b 化为小数后, 小数点后第 i 位的数字
        a %= b;
    }
    cout << t << endl;
    return 0;
}

```

25. 计算星期几【小学奥数 7831】

假设今天是星期日, 那么过 a^b 天之后是星期几?

输入:

两个正整数 a, b ($0 < a \leq 100, 0 < b \leq 10,000$), 中间用单个空格隔开。

输出:

一个字符串, 代表过 a^b 天之后是星期几。其中, Monday 是星期一, Tuesday 是星期二, Wednesday 是星期三, Thursday 是星期四, Friday 是星期五, Saturday 是星期六, Sunday 是星期日。

样例输入:

3 2000

样例输出:

Tuesday

【参考程序】

```

#include <iostream>
using namespace std;
int main()

```



```

{
    int a,b,s=1; //s 储存过  $a^b$  天后的天数,初始值赋为 1
    cin>>a>>b;
    for (int i=1; i<=b; ++i) //计算过  $a^b$  天后的天数
        s=(s*a)%7;
    if (s%7==0) cout<<" Sunday "<<endl;
    if (s%7==1) cout<<" Monday "<<endl;
    if (s%7==2) cout<<" Tuesday "<<endl;
    if (s%7==3) cout<<" Wednesday "<<endl;
    if (s%7==4) cout<<" Thursday "<<endl;
    if (s%7==5) cout<<" Friday "<<endl;
    if (s%7==6) cout<<" Saturday "<<endl; //判断过  $a^b$  天后是星期几
    return 0;
}

```

26. 幂的末尾【小学奥数 7833】

幂 a^b 的末 3 位数是多少?

输入:

两个正整数 $a, b (1 \leq a \leq 100, 1 \leq b \leq 10000)$ 。

输出:

从高位到低位输出幂的末三位数字,中间无分隔符。若幂本身不足三位,在前面补零。

样例输入:

7 2011

样例输出:

743

【参考程序】

```

#include<iostream>
#include<cstdio>
using namespace std;
int a,b,i,k=1;
int main()
{
    cin>>a>>b;
    for(i=1; i<=b; i++) //循环计算幂的末三位数字
    {
        k=k*a;
        k=k%1000; //k 储存幂  $a^i$  的后三位数字
    }
    if(k>=100)cout<<k;
    else if(k>=10)cout<<" 0 "<<k;
    else cout<<" 00 "<<k; //处理幂本身不足三位的情况
    return 0;
}

```

第二节 while 语句

1. 球弹跳高度的计算【1.5 编程基础之循环控制 20】

一个球从某一高度 h 落下(单位:米),每次落地后反弹回原来高度的一半,再落下。编程计算气球在第 10 次落地时,共经过多少米?第 10 次反弹多高?

输入:

输入一个整数 h ,表示球的初始高度。

输出:

第 1 行:到球第 10 次落地时,一共经过的米数;

第 2 行:第 10 次弹跳的高度。

注意:结果可能是实数,结果用 double 类型保存。

提示:输出时不需要对精度特殊控制,用 `cout << ANSWER`,或者 `printf("%g", ANSWER)` 即可。

样例输入:

20

样例输出:

59.9219

0.0195313

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    double h; int i=1;
    scanf("%lf",&h);
    double s=h; //第一次落地经过的米数,是实数,所以用 double
    while (i<=9) //9 次反弹
    {
        h/=2; //每次落地后反弹回原来高度的一半
        s+=h*2; //累加反弹和接下来落地的米数
        ++i; //次数+1
    }
    printf("%g\n%g\n",s,h/2); //最后一次弹跳的高度还是前一次的一半
    return 0;
}
```

2. 角谷猜想【1.5 编程基础之循环控制 21】

所谓角谷猜想,是指对于任意一个正整数,如果是奇数,则乘 3 加 1,如果是偶数,则除以 2,得到的结果再按照上述规则重复处理,最终总能够得到 1。如:假定初始整数为 5,计算过程分别为 16,8,4,2,1。程序要求输入一个整数,将经过处理得到 1 的过程输出来。

输入:

一个正整数 $n(n \leq 2,000,000)$ 。

输出:

从输入整数到 1 的步骤,每一步为一行,每一步中描述计算过程。最后一行输出“End”。如果输入为 1,直接输出“End”。

样例输入:

5

样例输出:

$5 * 3 + 1 = 16$

16/2=8

8/2=4

4/2=2

2/2=1

End

【参考程序】

```
#include <iostream>
using namespace std;
int main()
{
    long long n,a;
    cin>>n;
    while(n!=1)                //还没得到 1 便继续循环
    {
        if(n%2==1)             //判定是否为奇数
        {
            cout<<n<<" * 3+1="<<n * 3+1<<endl; //先输出这一步计算过程
            n=n * 3+1;          //将奇数乘 3 加 1
        }
        else if(n%2==0)         //判定是否为偶数
        {
            cout<<n<<" /2="<<n/2<<endl; //先输出这一步计算过程
            n/=2;                //将偶数除以 2
        }
    }
    cout<<" End ";
}
```

3. 级数求和【1.5 编程基础之循环控制 27】Noip2002 普及组第 1 题

已知： $S_n=1+1/2+1/3+\dots+1/n$ 。显然对于任意一个整数 k ，当 n 足够大的时候， S_n 大于 k 。现给出一个整数 $k(1 \leq k \leq 15)$ ，要求计算出一个最小的 n ，使得 $S_n > k$ 。

输入：

一个整数 $k(1 \leq k \leq 15)$ 。

输出：

一个整数 n 。

样例输入：

1

样例输出：

2

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n=1; double s=1,k;      //实数,用 double 类型
    scanf("%lf",&k);
    while (s<=k)                //Sn>k 条件还没达成便继续循环
    {
        ++n;                    //累加 n 的值
        s+=1.00/n;              //实数除法用 1.00
    }
}
```

```
printf("%d",n);
return 0;
}
```

4. 分离整数的各个数【1.5 编程基础之循环控制 28】

给定一个整数 $n(1 \leq n \leq 100,000,000)$ ，要求从个位开始分离出它的每一位数字。从个位开始按照从低位到高位顺序依次输出每一位数字。

输入：

一个整数，整数在 $1 \sim 100,000,000$ 之间。

输出：

从个位开始按照从低位到高位顺序依次输出每一位数字。数字之间以一个空格分开。

样例输入：

123

样例输出：

3 2 1

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int n,b,c;
    cin>>n;
    while(n!=0)                //还未全部分离便继续循环
    {
        cout<<n%10<<" ";      //输出个位数,用空格分隔
        n=n/10;                //整除 10,也相当于删除个位,之后原来的十位成为个位
    }
    cout<<endl;
    return 0;
}
```

5. 数字反转【1.5 编程基础之循环控制 29】Noip2011 普及组第 1 题

给定一个整数，请将该数各位上数字反转得到一个新数。新数也应满足整数的常见形式，即除非给定的原数为零，否则反转后得到的新数的最高位数字不应为零，例如输入 -380，反转后得到的新数为 -83。

输入：

一行，一个整数 $n(-1,000,000,000 \leq n \leq 1,000,000,000)$ 。

输出：

一行，一个整数，表示反转后的新数。

样例输入：

样例 #1：

123

样例 #2：

-380

样例输出

样例 #1：

321

样例 #2：

-83

【参考程序】

```

#include<cstdio>
#include<cstdlib>
using namespace std;
int main()
{
    int n,s=0;
    scanf("%d",&n);
    if (n<0) printf("-"); n=abs(n); //是负数,先输出负号,取绝对值
    if (n==0) {printf("0");exit(0);} //等于0则直接输出,结束
    while (n!=0) //还没完成反转便继续循环
    {
        s=s*10+n%10; //乘10,空出个位,原数的个位即为新数的最高位
        n=n/10; //整除10,也相当于删除个位
    }
    printf("%d\n",s);
    return 0;
}

```

6. 含 k 个 3 的数【1.5 编程基础之循环控制 30】

输入两个正整数 m 和 k,其中 $1 < m < 100000$, $1 < k < 5$,判断 m 能否被 19 整除,且恰好含有 k 个 3,如果满足条件,则输出 YES,否则,输出 NO。例如,输入:43833 3,满足条件,输出 YES。如果输入:39331 3,尽管有 3 个 3,但不能被 19 整除,也不满足条件,应输出 NO。

输入:

m 和 k 的值,中间用单个空格间隔。

输出:

满足条件时输出 YES,不满足时输出 NO。

样例输入:

43833 3

样例输出:

YES

【参考程序】

```

#include<iostream>
using namespace std;
int main()
{
    int m,n,k,s=0,t;
    cin>>m>>k;
    n=m; //用 n 保存 m 的值,进行分离,m 用于判断是否 19 的倍数
    while (n!=0) //还没分离完便继续循环
    {
        t=n%10; //从低位往高位分离各位
        if (t==3) s++; //若该位为 3,计数器 s 累加
        n/=10; //整除 10,相当于删除个位
    }
    if (m%19==0 && s==k) cout<<" YES "<<endl;
    //判断 m 能否被 19 整除,且恰好含有 k 个 3
    else cout<<" NO "<<endl;
    return 0;
}

```

第三节 do-while 语句

1. 球弹跳高度的计算【1.5 编程基础之循环控制 20】

一球从某一高度 h 落下(单位米),每次落地后弹跳回原来高度的一半,再落下。编程计算气球在第 10 次落地时,共经过多少米?第 10 次反弹多高?

输入:

输入一个整数 h ,表示球的初始高度。

输出:

第 1 行:到球第 10 次落地时,一共经过的米数;

第 2 行:第 10 次弹跳的高度。

注意:结果可能是实数,结果用 double 类型保存。

提示:输出时不需要对精度特殊控制,用 `cout << ANSWER`,或者 `printf("%g", ANSWER)` 即可。

样例输入:

20

样例输出:

59.9219

0.0195313

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    double h; int i=1;
    scanf("%lf",&h);           //h 存储每次落地后弹跳的高度
    double s=h;                 //s 先记录下第一次落地经过的米数
    do
    {
        h/=2;                   //每次落地后弹跳回原来高度的一半
        s+=h*2;                  //累加经过的米数
        ++i;                     //累加反弹次数
    }while(i<=9);               //反弹 9 次
    printf("%g\n%g\n",s,h/2);   //最后一次反弹是第 10 次落地的高度的一半
    return 0;
}
```

2. 角谷猜想【1.5 编程基础之循环控制 21】

所谓角谷猜想,是指对于任意一个正整数,如果是奇数,则乘 3 加 1,如果是偶数,则除以 2,得到的结果再按照上述规则重复处理,最终总能够得到 1。如:假定初始整数为 5,计算过程分别为 16,8,4,2,1。程序要求输入一个整数,将经过处理得到 1 的过程输出来。

输入:

一个正整数 $n(n \leq 2,000,000)$ 。

输出:

从输入整数到 1 的步骤, 每一步为一行, 每一步中描述计算过程。最后一行输出“End”。如果输入为 1, 直接输出“End”。

样例输入:

5

样例输出:

$5 * 3 + 1 = 16$

$16 / 2 = 8$

$8 / 2 = 4$

$4 / 2 = 2$

$2 / 2 = 1$

End

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    long long n;
    cin>>n;
    if (n==1) cout<<" End "<<endl; //如果等于 1,无须循环
    else
    {
        do
        {
            if (n%2!=0) //判断是否为奇数
            {
                cout<<n<<" * 3 + 1 = "; //输出计算过程
                n=n * 3 + 1;
                cout<<n<<endl; //奇数乘 3 加 1
            }
            else //否则是偶数
            {
                cout<<n<<" / 2 = "; //输出计算过程
                n=n/2; //偶数除以 2
                cout<<n<<endl;
            }
        }while(n!=1); //还没得到 1 便继续循环
        cout<<" End "<<endl;
    }
    return 0;
}
```

3. 级数求和【1.5 编程基础之循环控制 27】Noip2002 普及组第 1 题

已知: $S_n = 1 + 1/2 + 1/3 + \dots + 1/n$ 。显然对于任意一个整数 k , 当 n 足够大的时候, S_n 大于 k 。现给出一个整数 $k(1 \leq k \leq 15)$, 要求计算出一个最小的 n , 使得 $S_n > k$ 。

输入:

一个整数 $k(1 \leq k \leq 15)$ 。

输出:

一个整数 n 。

样例输入:

1

样例输出:

2

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n=1; double s=1,k;           //实数,用 double 类型
    scanf("%lf",&k);
    do
    {
        ++n;                          //累加 n 的值
        s+=1.00/n;                    //累加 1/n,实数,用 1.00
    }while (s<=k);                    //Sn>k 条件未达成便继续循环
    printf("%d",n);
    return 0;
}
```

4. 分离整数的各个数【1.5 编程基础之循环控制 28】

给定一个整数 $n(1 \leq n \leq 100,000,000)$,要求从个位开始分离出它的每一位数字。从个位开始按照从低位到高位顺序依次输出每一位数字。

输入:

一个整数,整数在 $1 \sim 100,000,000$ 之间。

输出:

从个位开始按照从低位到高位顺序依次输出每一位数字。数字之间以一个空格分开。

样例输入:

123

样例输出:

3 2 1

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int n,x;
    scanf("%d",&n);
    do
    {
        x=n%10;           //分离个位
        n/=10;             //整除 10,也相当于删除个位,原来的十位现在为个位
        printf("%d ",x);   //输出个位数,用空格分隔
    }while(n>0);          //还未全部分离便继续循环
    return 0;
}
```


5. 数字反转【1.5 编程基础之循环控制 29】Noip2011 普及组第 1 题

给定一个整数, 请将该数各位上数字反转得到一个新数。新数也应满足整数的常见形式, 即除非给定的原数为零, 否则反转后得到的新数的最高位数字不应为零, 例如输入 -380, 反转后得到的新数为 -83。

输入:

一行, 一个整数 n ($-1,000,000,000 \leq n \leq 1,000,000,000$)。

输出:

一行, 一个整数, 表示反转后的新数。

样例输入:

样例 #1:

123

样例 #2:

-380

样例输出:

样例 #1:

321

样例 #2:

-83

【参考程序】

```
#include<cstdio>
#include<cmath>
using namespace std;
int main()
{
    int n,s=0;
    scanf("%d",&n);
    if(n<0) printf("-");
    n=abs(n);          //若为负数,先输出负号,再取绝对值
    do
    {
        s=s*10+n%10;    //乘 10,空出个位,原数的个位即为新数的最高位
        n/=10;          //整除 10,也相当于删除个位
    }while(n!=0);      //还没完成反转便继续循环
    printf("%d\n",s);
    return 0;
}
```

6. 含 k 个 3 的数【1.5 编程基础之循环控制 30】

输入两个正整数 m 和 k , 其中 $1 < m < 100000$, $1 < k < 5$, 判断 m 能否被 19 整除, 且恰好含有 k 个 3, 如果满足条件, 则输出 YES, 否则, 输出 NO。例如, 输入: 43833 3, 满足条件, 输出 YES。如果输入: 39331 3, 尽管有 3 个 3, 但不能被 19 整除, 也不满足条件, 应输出 NO。

输入:

m 和 k 的值, 中间用单个空格间隔。

输出:

满足条件时输出 YES, 不满足时输出 NO。

样例输入:

43833 3

样例输出:

YES

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int k,m,w=0,x; bool f;
    f=false;
    scanf("%d%d",&m,&k);
    if (m%19==0) f=true; //判断是否为 19 的倍数
    do
    {
        x=m%10;           //分离各位
        m/=10;             //整除 10,也相当于删除个位
        if (x==3) ++w;      //若该位为 3,计数器+1
    } while(m>0);          //还未全部分离便继续循环
    if (w==k && f==true) printf(" YES "); //若是 19 的倍数且含 k 个 3,满足条件
    else printf(" NO ");
    return 0;
}
```

第四节 循环嵌套

1. 求阶乘的和【1.5 编程基础之循环控制 34】

给定正整数 n , 求不大于 n 的正整数的阶乘的和(即求 $1! + 2! + 3! + \dots + n!$), 输出阶乘的和。

输入:

一行, 包含一个正整数 $n(1 < n < 12)$ 。

输出:

一行, 阶乘的和。

样例输入:

5

样例输出:

153

【参考程序】

```
#include<stdio>           //程序为优化程序,只用一重循环
using namespace std;
int main()
{
    int n;
    long long s=0,t=1;      //乘法运算初始值为1,这个数可能很大,用 long long
    scanf("%d",&n);
    for (int i=1;i<=n;++i)
    {
        t*=i;              //计算 i!, 只要用之前存储的(i-1)! * i
        s+=t;              //累加 i!
    }
    printf("%lld\n",s);
    return 0;
}
```

2. 求出 e 的值【1.5 编程基础之循环控制 35】

利用公式 $e = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$, 求 e 的值, 要求保留小数点后 10 位。

输入:

只有一行, 该行包含一个整数 $n(2 \leq n \leq 15)$, 表示计算 e 时累加到 $1/n!$ 。

输出:

只有一行, 该行包含计算出来的 e 的值, 要求打印小数点后 10 位。

样例输入:

10

样例输出:

2.7182818011

【参考程序】

```
#include<stdio>           //程序为优化程序,只用一重循环
using namespace std;
int main()
```

```
{
    int n;
    long long t=1;           //乘法运算初始值要等于 1
    double e=1;              //e 为实数
    scanf("%d",&n);
    for (int i=1;i<=n;++i)
    {
        t*=i;                //计算 i!, 只要用(i-1)! * i
        e+=1.0/t;             //累加 1/i!, 实数计算用 1.0
    }
    printf("%.10lf\n",e);    //保留 10 位小数
    return 0;
}
```

3. 计算多项式的值【1.5 编程基础之循环控制 36】

假定多项式的形式为 $x^n + x^{n-1} + \dots + x^2 + x + 1$, 请计算给定单精度浮点数 x 和正整数 n 的情况下这个多项式的值。 x 在 float 范围内, $n \leq 1,000,000$ 。多项式的值精确到小数点后两位, 保证最终结果在 float 范围内。

输入:

一行, 包括 x 和 n 的值, 用单个空格隔开。 x 在 float 范围内, $n \leq 1,000,000$ 。

输出:

一个实数, 即多项式的值, 精确到小数点后两位。保证最终结果在 float 范围内。

样例输入:

2.0 4

样例输出:

31.00

【参考程序】

```
#include<stdio>           //程序为优化程序, 只用一重循环
using namespace std;
int main()
{
    int n;
    float s=1,x,t=1;        //最终结果在 float 范围内, 乘法计算初始值为 1
    scanf("%f%d",&x,&n);
    for (int i=1;i<=n;++i)
    {
        t*=x;               //  $x^i$  可转化为  $x^{i-1} * x$ 
        s+=t;                //累加  $x^i$ 
    }
    printf("%.2f\n",s);     //保留 2 位小数
    return 0;
}
```

4. 与 7 无关的数【1.5 编程基础之循环控制 39】

一个正整数, 如果它能被 7 整除, 或者它的十进制表示法中某一位上的数字为 7, 则称其为与 7 相关的数。现求所有小于等于 n ($n < 100$) 与 7 无关的正整数的平方和。

输入:

一行, 正整数 n ($n < 100$)。

输出:

一行,包含一个整数,即小于等于 n 的所有与 7 无关的正整数的平方和。

样例输入:

21

样例输出:

2336

【参考程序】

```
#include<iostream>
using namespace std;
int main()
{
    int n;
    long long s=0;           //初始值为 0
    cin>>n;
    for (int i=1;i<=n;++i)    //枚举所有小于 n 的数
        if (i%7!=0 && i/10!=7 && i%10!=7)
            //不能被 7 整除,或十位、个位没有 7
            s+=i*i;           //累加平方和
    cout<<s<<endl;          //所有与 7 无关的正整数的平方和
    return 0;
}
```

5. 数 1 的个数【1.5 编程基础之循环控制 40】

给定一个十进制正整数 $n(1 \leq n \leq 10,000)$, 写下从 1 到 n 的所有整数, 然后数一下其中出现的数字“1”的个数。

例如当 $n=2$ 时, 写下 1, 2。这样只出现了 1 个“1”; 当 $n=12$ 时, 写下 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12。这样出现了 5 个“1”。

输入:

正整数 $n(1 \leq n \leq 10,000)$ 。

输出:

一个正整数, 即“1”的个数。

样例输入:

12

样例输出:

5

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int i, n, a, s=1, j;
    scanf("%d", &n);
    for(i=2; i<=n; ++i)      //枚举从 2 到 n 的每个数, 进行分离
    {
        a=i;                 //不能改变循环中 i 的值, 所以用 a 来代替
        do{
            j=a%10;          //不断分离出 a 的个位
            a=a/10;           //整除 10, 删去个位
        } while(j!=0);
        s+=j;
    }
    printf("%d", s);
}
```

```

        if (j==1)s++; //若该位为 1,计数器+1
    }while (a!=0); //还未分离完便继续循环
}
printf("%d\n",s);
return 0;
}

```

6. 数字统计【1.5 编程基础之循环控制 41】Noip2010 普及组第 1 题

请统计某个给定范围 $[l, r]$ 的所有整数中,数字 2 出现的次数。

比如给定范围 $[2, 22]$,数字 2 在数 2 中出现了 1 次,在数 12 中出现 1 次,在数 20 中出现 1 次,在数 21 中出现 1 次,在数 22 中出现 2 次,所以数字 2 在该范围内一共出现了 6 次。

输入:

一行,为两个正整数 l 和 r ,之间用一个空格隔开。

输出:

一行,表示数字 2 出现的次数。

样例输入:

样例 #1:

2 22

样例 #2:

2 100

样例输出:

样例 #1:

6

样例 #2:

20

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    int i,a,s=0,j,l,r;
    scanf("%d%d",&l,&r);
    for(i=l;i<=r;i++) //枚举 l 到 r 中含 2 的个数
    {
        a=i; //不能改变循环中 i 的值,所以用 a 来代替
        do
        {
            j=a%10; //不断分离出 a 的个位
            a=a/10; //除以 10,删去个位
            if (j==2) s++; //若该位为 2,计数器+1
        }while (a!=0); //循环至无法再分离为止
    }
    printf("%d\n",s);
    return 0;
}

```

7. 画矩形【1.5 编程基础之循环控制 42】

根据参数,画出矩形。输入四个参数:前两个参数为整数,依次代表矩形的高和宽(高不少于 3 行不多于 10 行,宽不少于 5 列不多于 10 列);第三个参数是一个字符,表示用来画矩形的符号;第四个参数为 1 或 0,0 代表空心,1 代表实心。

输入:

一行,包括四个参数:前两个参数为整数,依次代表矩形的高和宽(高不少于 3 行不多于 10 行,宽不少于 5 列不多于 10 列);第三个参数是一个字符,表示用来画矩形的符号;第四个

参数为 1 或 0, 0 代表空心, 1 代表实心。

输出:

输出画出的图形。

样例输入:

7 7 @ 0

样例输出:

```

@@@@@@@
@       @
@       @
@       @
@       @
@       @
@@@@@@@

```

【参考程序】

```

#include<iostream>
using namespace std;
int x,y,flag,i,j;
char a;
int main()
{
    cin>>x>>y>>a>>flag;
    if(flag) //实心
        for(i=1;i<=x;i++)
        {
            for(j=1;j<=y;j++)
                cout<<a; //按格式输出 x 行 y 列图形,只有字符
            cout<<endl; //一行结束,换行
        }
    else //空心
        for(i=1;i<=x;i++)
        {
            for(j=1;j<=y;j++)
                if((i!=1)&&(i!=x)&&(j!=1)&&(j!=y))
                    cout<<' '; //不在第一行或最后一行或第一列或最后一列,就空格
                else cout<<a; //否则输出字符
            cout<<endl; //一行结束,换行
        }
    return 0;
}

```

8. 质因数分解【1.5 编程基础之循环控制 43】Noip2012 普及组第 1 题

已知正整数 n 是两个不同的质数的乘积, 试求出较大的那个质数。

输入:

只有一行, 包含一个正整数 n 。

对于 60% 的数据, $6 \leq n \leq 1000$ 。

对于 100% 的数据, $6 \leq n \leq 2 \times 10^9$ 。

输出:

一行,包含一个正整数 p ,即较大的那个质数。

样例输入:

21

样例输出:

7

【参考程序】

```
#include<cstdio>
#include<cmath>           //用 sqrt 须调用 cmath 库
using namespace std;
int main()
{
    int n,a,b;
    scanf("%d",&n);
    for(int i=2;i<=sqrt(n);i++) //两个不同的质数中,其中一个必定<=sqrt(n)
    {
        if(n%i==0)           //能整除,说明是其中一个质数
        { a=i;break; }       //保存其值
    }
    b=n/a;                   //求另一个质数
    if(a>b) printf("%d\n",a);
    else printf("%d\n",b);    //输出较大的质数
    return 0;
}
```

9. 第 n 小的质数【1.5 编程基础之循环控制 44】

输入一个正整数 n ,求第 n 小的质数。

输入:

一个不超过 10000 的正整数 n 。

输出:

第 n 小的质数。

样例输入:

10

样例输出:

29

【参考程序】

```
#include<cstdio>
#include<cmath>
using namespace std;
int main()
{
    int n,i,s=0;
    scanf("%d",&n);
    for( i=2; ;++i)
    {
        int x=2;
        while(x<=floor(sqrt(i))&&i%x!=0) x++; //判断 i 是否为素数
        if (x>floor(sqrt(i)))
```



```

    {
        s++; //若是素数累加 s,该数是第 s 小的
        if (s==n) //若 s=n,则已经找到第 n 小的质数,结束
        {
            printf("%d\n",i); break;
        }
    }
}
return 0;
}

```

10. 金币【1.5 编程基础之循环控制 45】

国王将金币作为工资,发放给忠诚的骑士。第 1 天,骑士收到一枚金币;之后两天(第 2 天和第 3 天)里,每天收到两枚金币;之后三天(第 4、第 5、第 6 天)里,每天收到三枚金币;之后四天(第 7、第 8、第 9、第 10 天)里,每天收到四枚金币……这种工资发放模式会一直这样延续下去:当连续 n 天每天收到 n 枚金币后,骑士会在之后的连续 $n+1$ 天里,每天收到 $n+1$ 枚金币(n 为任意正整数)。

你需要编写一个程序,确定从第一天开始的给定天数内,骑士一共获得了多少金币。

输入:

一个整数(范围 $1 \sim 10,000$),表示天数。

输出:

骑士获得的金币数。

样例输入:

6

样例输出:

14

【参考程序】

```

#include<iostream>
#include<cstdlib>
using namespace std;
int n,s=0,t=0;
int main()
{
    cin>>n;
    for(int i=1;;i++)
        for(int j=1;j<=i;j++) //连续 j 天,每天能得到 i 个金币
        {
            s+=i; t++; //累加总金币数,目前天数
            if (t==n) //直到第 n 天,输出总得金币,结束
            {
                cout<<s;exit(0);
            }
        }
    return 0;
}

```

11. 不定方程求解【小学奥数 7650】

给定正整数 a, b, c 。求不定方程 $ax+by=c$ 关于未知数 x 和 y 的所有非负整数解的

组数。

输入：

一行,包含三个正整数 a, b, c , 整数之间用单个空格隔开。每个数均不大于 1000。

输出：

一个整数,即不定方程的非负整数解的组数。

样例输入：

2 3 18

样例输出：

4

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
int main()
{
    int a,b,c,x,y,s=0;
    cin>>a>>b>>c;
    for (x=0;x<=c/a;++x)           //因为  $ax+by=c$ , 故  $x$  的循环范围为  $0\sim c/a$ 
        for (y=0;y<=c/b;++y)       //y 的循环范围  $0\sim c/b$ 
            if (a * x + b * y == c) s++; //符合条件,计数器+1
    cout<<s<<endl;                //输出符合条件的个数
    return 0;
}
```

第五章 数组

第一节 一维数组

1. 与指定数字相同的数的个数【1.6 编程基础之一维数组 01】

输出一个整数序列中与指定数字相同的数的个数。

输入：

输入包含三行：

第 1 行为 n ($n \leq 100$), 表示整数序列的长度；

第 2 行为 n 个整数, 整数之间以一个空格分开；

第 3 行包含一个整数, 为指定的数字 m 。

输出：

输出为 n 个数中与 m 相同的数的个数。

样例输入：

3

2 3 2

2

样例输出：

2

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int a[1001],n,m,k=0;
    scanf("%d",&n);           //共有 n 个数
    for (int i=1;i<=n;++i)
        scanf("%d",&a[i]);    //读入 n 个数
    scanf("%d",&m);           //指定数 m
    for (int i=1;i<=n;++i)
        if (a[i]==m) ++k;      //若这个数与 m 相同,计数器加 1
    printf("%d",k);            //输出相同的个数
    return 0;
}
```

2. 陶陶摘苹果【1.6 编程基础之一维数组 02】Noip2005 普及组第 1 题

陶陶家的院子里有一棵苹果树,每到秋天树上就会结出 10 个苹果。苹果成熟的时候,陶陶就会跑去摘苹果。陶陶有个 30 厘米高的板凳,当她不能直接用手摘到苹果的时候,就会踩到板凳上再试试。

现在已知 10 个苹果到地面的高度,以及陶陶把手伸直的时候能够达到的最大高度,请帮陶陶算一下她能够摘到的苹果的数目。假设她碰到苹果,苹果就会掉下来。

输入:

两行数据。第 1 行包含 10 个 100~200 之间(包括 100 和 200)的整数(以厘米为单位)分别表示 10 个苹果到地面的高度,两个相邻的整数之间用一个空格隔开。第 2 行只包括一个 100~120 之间(包含 100 和 120)的整数(以厘米为单位),表示陶陶把手伸直的时候能够达到的最大高度。

输出:

一行,这一行只包含一个整数,表示陶陶能够摘到的苹果的数目。

样例输入:

100 200 150 140 129 134 167 198 200 111
110

样例输出:

5

【参考程序】

```
#include<stdio>
using namespace std;
int a[100001],h,i,k=0;
int main()
{
    const int n=10;                //只有 10 个数,所以 n 为定值 10
    for(i=1;i<=n;++i)
        scanf("%d",&a[i]);        //读入 10 个苹果到地面的高度
    scanf("%d",&h);
    h+=30;                          //陶陶有个 30 厘米高的板凳
    for(i=1;i<=n;++i)
        if (h>=a[i]) k++;          //摘到的高度大于或等于苹果的高度,则累加个数
    printf("%d\n",k);              //输出能摘到的个数
    return 0;
}
```

3. 计算书费【1.6 编程基础之一维数组 03】

下面是一个图书的单价表:

图书	单价	图书	单价
《计算概论》	28.9 元/本	《数据结构与算法》	32.7 元/本
《数字逻辑》	45.6 元/本	《C++ 程序设计教程》	78 元/本
《人工智能》	35 元/本	《计算机体系结构》	86.2 元/本
《编译原理》	27.8 元/本	《操作系统》	43 元/本
《计算机网络》	56 元/本	《JAVA 程序设计》	65 元/本

给定每种图书购买的数量,编程计算应付的总费用。

输入:

一行,包含 10 个整数(大于等于 0,小于等于 100),分别表示购买的《计算概论》《数据结构与算法》《数字逻辑》《C++ 程序设计教程》《人工智能》《计算机体系结构》《编译原理》《操作系统》《计算机网络》《JAVA 程序设计》的数量(以本为单位)。每两个整数之间用一个空格分开。

输出:

一行, 包含一个浮点数 f , 表示应付的总费用。精确到小数点后一位。

样例输入:

1 5 8 10 5 1 1 2 3 4

样例输出:

2140.2

【参考程序】

```
#include <cstdio>
using namespace std;
double a[10] = {28.9, 32.7, 45.6, 78, 35, 86.2, 27.8, 43, 56, 65};
//10 种图书的单价, 为实数

double b[10];
int main()
{
    double s = 0;
    for (int i = 0; i <= 9; ++i)
        scanf("%lf", &b[i]); //因为数组从 a[0] 开始, 所以循环由 0~9
    for (int i = 0; i <= 9; ++i)
        s += a[i] * b[i]; //每种书的总价 = 单价 × 数量, 累加到应付的总费用
    printf("%.1lf", s); //精确到小数点后一位
    return 0;
}
```

4. 数组逆序重放【1.6 编程基础之一维数组 04】

将一个数组中的值按逆序重新存放。例如, 原来的顺序为 8, 6, 5, 4, 1。要求改为 1, 4, 5, 6, 8。

输入:

两行: 第 1 行为数组中元素的个数 n ($1 < n < 100$); 第 2 行是 n 个整数, 每两个整数之间用空格分隔。

输出:

一行: 逆序后数组的整数, 每两个整数之间用空格分隔。

样例输入:

5
8 6 5 4 1

样例输出:

1 4 5 6 8

【参考程序】

```
#include <cstdio>
using namespace std;
int a[1001];
int main()
{
    int n, t;
    scanf("%d", &n);
    for (int i = 1; i <= n; ++i)
        scanf("%d", &a[i]);
    for (int i = 1; i <= n/2; ++i) //交换要 n/2 次
    {
```

```

    t=a[i];a[i]=a[n+1-i];a[n+1-i]=t;//a[i]和 a[n+1-i]交换
}
for (int i=1; i<=n; ++i)
    printf("%d ",a[i]);          //输出逆序后的数组
return 0;
}

```

5. 年龄与疾病【1.6 编程基础之一维数组 05】

某医院想统计一下某项疾病的获得与否与年龄是否有关,需要对以前的诊断记录进行整理,按照 0~18,19~35,36~60,61 以上(含 61)四个年龄段统计的患病人数占总患病人数的比例。

输入:

共两行,第 1 行为以往病人的数目 $n(0 < n \leq 100)$,第 2 行为每个病人患病时的年龄。

输出:

按照 0~18,19~35,36~60,61 以上(含 61)四个年龄段输出该段患病人数占总患病人数的比例,以百分比的形式输出,精确到小数点后两位。每个年龄段占一行,共四行。

样例输入:

```

10
1 11 21 31 41 51 61 71 81 91

```

样例输出:

```

20.00%
20.00%
20.00%
40.00%

```

【参考程序】

```

#include<iostream>
#include<cstdio>
using namespace std;
int a[10000];
int main()
{
    int i,n;
    double b1,b2,b3,b4,a1=0,a2=0,a3=0,a4=0;
    cin>>n;          //共有 n 个病人
    for (i=1; i<=n; ++i)
    { //按照 0—18,19—35,36—60,61 以上(含 61)四个年龄段分别计数
        cin>>a[i];
        if (a[i]<=18) a1=a1+1; else
        if (a[i]<=35) a2=a2+1; else
        if (a[i]<=60) a3=a3+1; else
        if (a[i]>=61) a4=a4+1;
    }
    b1=a1/n*100; b2=a2/n*100;
    b3=a3/n*100; b4=a4/n*100; //百分率=个数/n*100
    printf("%.2lf%%\n",b1);
    printf("%.2lf%%\n",b2);
    printf("%.2lf%%\n",b3);
    printf("%.2lf%%\n",b4);    //输出四个年龄段患病人数所占百分率
    return 0;
}

```

6. 校门外的树【1.6 编程基础之一维数组 06】Noip2005 普及组第 2 题

某校大门外长度为 L 的马路上有一排树，每两棵相邻的树之间的间隔都是 1 米。我们可以把马路看成一个数轴，马路的一端在数轴 0 的位置，另一端在 L 的位置；数轴上的每个整数点，即 $0, 1, 2, \dots, L$ ，都种有一棵树。

由于马路上有一些区域要用来建地铁。这些区域用它们在数轴上的起始点和终止点表示。已知任一区域的起始点和终止点的坐标都是整数，区域之间可能有重合的部分。现在要把这些区域中的树(包括区域端点处的两棵树)移走。你的任务是计算将这些树都移走后，马路上还有多少棵树。

输入：

第 1 行有两个整数 L ($1 \leq L \leq 10,000$) 和 M ($1 \leq M \leq 100$)， L 代表马路的长度， M 代表区域的数目， L 和 M 之间用一个空格隔开。接下来的 M 行每行包含两个不同的整数，用一个空格隔开，表示一个区域的起始点和终止点的坐标。

对于 20% 的数据，区域之间没有重合的部分；对于其他的数据，区域之间有重合的情况。

输出：

一行，这一行只包含一个整数，表示马路上剩余的树的数目。

样例输入：

```
500 3
150 300
100 200
470 471
```

样例输出：

```
298
```

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int a[10001]; int l,m,x,y,s=0;
    scanf("%d%d",&l,&m);
    for (int i=0;i<=l;++i) a[i]=1;           //先初始化为所有树都没移走
    for (int i=1;i<=m;++i)
    {
        scanf("%d%d",&x,&y);               //输入需移走的范围
        for (int j=x;j<=y;++j) a[j]=0;     //对需要移走的树进行标记
    }
    for (int i=0;i<=l;++i)
        if (a[i]==1) ++s;                   //若没移走，计数器+1
    printf("%d",s);
    return 0;
}
```

7. 向量点积计算【1.6 编程基础之一维数组 09】

在线性代数、计算几何中，向量点积是一种十分重要的运算。给定两个 n 维向量 $\mathbf{a} = (a_1, a_2, \dots, a_n)$ 和 $\mathbf{b} = (b_1, b_2, \dots, b_n)$ ，求点积 $\mathbf{a} \cdot \mathbf{b} = a_1 \cdot b_1 + a_2 \cdot b_2 + \dots + a_n \cdot b_n$ 。

输入:

第 1 行是一个整数 $n(1 \leq n \leq 1000)$;

第 2 行包含 n 个整数 $a_1, a_2, \dots, a_n$;

第 3 行包含 n 个整数 $b_1, b_2, \dots, b_n$ 。

相邻整数之间用单个空格隔开。每个整数的绝对值都不超过 1000。

输出:

一个整数,即两个向量的点积结果。

样例输入:

```
3
1 4 6
2 1 5
```

样例输出:

```
36
```

【参考程序】

```
#include<cstdio>
using namespace std;
int main()
{
    int a[1001],b[1001],n,s=0;
    scanf("%d",&n);
    for (int i=1;i<=n;++i)
        scanf("%d",&a[i]);           //读入 a 数组
    for (int i=1;i<=n;++i)
        scanf("%d",&b[i]);           //读入 b 数组
    for (int i=1;i<=n;++i)
        s+=a[i]*b[i];                 //累加 a[i]*b[i]
    printf("%d",s);                   //输出向量的点积结果
    return 0;
}
```

8. 开关灯【1.5 编程基础之循环控制 31】

假设有 n 盏灯(n 为不大于 5000 的正整数),从 $1 \sim n$ 按顺序依次编号,初始时全部处于开启状态;有 m 个人(m 为不大于 n 的正整数)也从 $1 \sim m$ 依次编号。

第一个人(1 号)将灯全部关闭,第二个人(2 号)将编号为 2 的倍数的灯打开,第三个人(3 号)将编号为 3 的倍数的灯做相反处理(即将打开的灯关闭,将关闭的灯打开)。依照编号递增顺序,以后的人都和 3 号一样,将凡是自己编号倍数的灯做相反处理。

请问:当第 m 个人操作之后,哪几盏灯是关闭的,按从小到大输出其编号,其间用逗号间隔。

输入:

输入正整数 n 和 m ,以单个空格隔开。

输出:

顺次输出关闭的灯的编号,其间用逗号间隔。

样例输入:

```
10 10
```

样例输出:

```
1,4,9
```

【参考程序】

```
#include<iostream>
```



```

#include<cstdio>
#include<cstring>
using namespace std;
int a[5001];
int main()
{
    int n,m,z=1;
    cin>>n>>m;
    memset(a,0,sizeof(a));           //初始化全部关闭
    for (int i=1; i<=m; ++i)
        for (int j=1; j<=n; ++j)
            if (j%i==0) a[j]=! a[j]; //编号是 i 的倍数的灯做相反处理
    for (int i=1; i<=n; ++i)
        if (a[i])                     //是关闭的灯
        {
            if (z) z=0;
            else printf(",");          //第一次不用先输出逗号
            printf("%d ",i);           //顺次输出关闭的灯的编号
        }
    printf("\n");
    return 0;
}

```

9. 查找特定的值【1.9 编程基础之顺序查找 01】

在一个序列(下标从 1 开始)中查找一个给定的值,输出第一次出现的位置。

输入:

第 1 行包含一个正整数 n ($1 \leq n \leq 10000$),表示序列中元素个数。

第 2 行包含 n 个整数,依次给出序列的每个元素,相邻两个整数之间用单个空格隔开。
元素的绝对值不超过 10000。

第 3 行包含一个整数 x ,为需要查找的特定值。 x 的绝对值不超过 10000。

输出:

若序列中存在 x ,输出 x 第一次出现的下标;否则输出 -1。

样例输入:

```

5
2 3 6 7 3
3

```

样例输出:

```

2

```

【参考程序】

```

#include<cstdio>
using namespace std;
int a[10001];
int main()
{
    int n,x,i,k=-1;                //还没找到特定值,k 先赋值-1
    scanf("%d",&n);
    for (i=1; i<=n; ++i) scanf("%d",&a[i]);
    scanf("%d",&x);
}

```

```

i=0;
while (i<=n&& k<0)           //还没找到特定值,便继续循环
{
    i+=1;
    if (a[i]==x) k=i;         //找到,标记,然后可以直接退出
}
printf("%d\n",k);             //输出第一次出现的下标或-1
return 0;
}

```

10. 不高兴的津津【1.9 编程基础之顺序查找 03】Noip2004 普及组第 1 题

津津上初中了。妈妈认为津津应该更加用功学习,所以津津除了上学之外,还要参加妈妈为她报名的各科复习班。另外,妈妈每周还会送她去学习朗诵、舞蹈和钢琴。但是津津如果一天上课超过 8 个小时就会不高兴,而且上得越久就会越不高兴。假设津津不会因为其他事不高兴,并且她的不高兴不会持续到第二天。请你帮忙检查一下津津下周的日程安排,看看下周她会不会不高兴;如果会的话,哪天最不高兴。

输入:

七行数据,分别表示周一到周日的日程安排。每行包括两个小于 10 的非负整数,用空格隔开,分别表示津津在学校上课的时间和妈妈安排她上课的时间。

输出:

一行,这一行只包含一个数字。如果不会不高兴,则输出 0;如果会,则输出最不高兴的是周几(用 1,2,3,4,5,6,7 分别表示周一、周二、周三、周四、周五、周六、周日)。如果有两天或两天以上不高兴的程度相当,则输出时间最靠前的一天。

样例输入:

```

5 3
6 2
7 2
5 3
5 4
0 4
0 6

```

样例输出:

```

3

```

【参考程序】

```

#include<stdio>
#include<stdlib>
using namespace std;
int main()
{
    //先初始化为不会不高兴,如果有不高兴的星期数,再替换
    int x,y,c[1001],i,max=0;
    for (i=1;i<=7;++i)
    {
        scanf("%d%d",&x,&y);
        c[i]=x+y;           //每天的上课时间=上午时间+下午时间
        if (c[i]>max && c[i]>8)
            max=c[i];       //若这天会不高兴,更新最大值
    }
    for (i=1;i<=7;++i)      //找到最不高兴的最靠前的一天,结束

```

```

    if (c[i] == max) {printf("%d", i); exit(0);}
    return 0;
}

```

11. 最大值和最小值的差【1.9 编程基础之顺序查找 05】

输出一个整数序列中最大的数和最小的数的差。

输入：

第 1 行为 m, 表示整数个数, 整数个数不会大于 10,000;

第 2 行为 m 个整数, 分别以空格隔开, 每个整数的绝对值不会大于 10,000。

输出：

m 个数中最大值和最小值的差。

样例输入：

```

5
2 5 7 4 2

```

样例输出：

```

5

```

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    //最大值初始化为很小的数,保证能替换数组里的数,最小值同理
    int a[10001], m, max = -10001, min = 10001;
    scanf("%d", &m);
    for (int i = 1; i <= m; ++i)
    {
        scanf("%d", &a[i]);
        if (a[i] > max) max = a[i];           //如果比目前的最大值大则替换
        if (a[i] < min) min = a[i];         //如果比目前的最小值小则替换
    }
    printf("%d", max - min);               //输出最大值和最小值的差
    return 0;
}

```

12. 不与最大数相同的数字之和【1.9 编程基础之顺序查找 07】

输出一个整数数列中不与最大数相同的数字之和。

输入：

分为两行：

第 1 行为 n(n 为接下来数的个数, $n \leq 100$);

第 2 行 n 个整数, 分别以一个空格分开, 每个整数的大小在 -1000,000 与 1000,000 之间。

输出：

n 个数中除去最大数后其余数字之和。

样例输入：

```

3
1 2 3

```

样例输出：

```

3

```

【参考程序】

```

#include<cstdio>
using namespace std;
int main()
{
    //最大值初始化为很小的数,保证能被数组里的数替换
    int a[1001],n,s=0,i,max=-1000;
    scanf("%d",&n);
    for (i=1;i<=n;++i)
    {
        scanf("%d",&a[i]);
        if (a[i]>max) max=a[i];           //更新最大值
    }
    for (i=1;i<=n;++i)
        if (a[i]!=max) s+=a[i];         //若不与最大值相同,累加
    printf("%d",s);
    return 0;
}

```

13. 白细胞计数【1.9 编程基础之顺序查找 08】

医院采样了某临床病例治疗期间的白细胞数量样本 n 份,用于分析某种新抗生素对该病例的治疗效果。为了降低分析误差,要先从这 n 份样本中去除一个数值最大的样本和一个数值最小的样本,然后将剩余 $n-2$ 个有效样本的平均值作为分析指标。同时,为了观察该抗生素的疗效是否稳定,还要给出该平均值的误差,即所有有效样本(不包括已扣除的两个样本)与该平均值之差的绝对值的最大值。

现在请你编写程序,根据提供的 n 个样本值,计算出该病例的平均白细胞数量和对应的误差。

输入:

第 1 行是一个正整数 $n(2 \leq n \leq 300)$,表明共有 n 个样本。

以下共有 n 行,每行为一个浮点数,为对应的白细胞数量,其单位为 $10^9/L$ 。数与数之间以一个空格分开。

输出:

两个浮点数,中间以一个空格分开。分别为平均白细胞数量和对应的误差,单位也是 $10^9/L$ 。计算结果需保留到小数点后 2 位。

样例输入:

```

5
12.0
13.0
11.0
9.0
10.0

```

样例输出:

```

11.00 1.00

```

【参考程序】

```

#include<cstdio>
#include<iostream>
#include<cmath>           //用 abs 须调用 cmath 库
using namespace std;
double a[1001],b[1001];
int main()

```

```

{
    int k,v;
    double w,max=-10001,min=10001,s=0,n; //白细胞数量为实数
    scanf("%lf",&n);
    for (int i=1;i<=n;++i) cin>>a[i];
    for (int i=1;i<=n;++i)
    {
        s+=a[i];
        if (a[i]>max) {max=a[i];k=i;}
        if (a[i]<min) {min=a[i];v=i;} //找最大值、最小值
    }
    w=(s-a[v]-a[k])/(n-2.00); //求平均白细胞数量
    printf("%.2lf",w); //保留两位小数
    for (int i=1;i<=n;++i)
        if (i!=k&&i!=v) b[i]=abs(w-a[i]); //计算每个有效样本与平均数的差值
        else b[i]=-10001; //否则赋值最小,不可能为最大值
    max=-10001;
    for (int i=1;i<=n;++i)
        if (b[i]>max) max=b[i]; //找最大的差值
    printf("%.2lf\n",max);
    return 0;
}

```

14. 直方图【1.9 编程基础之顺序查找 09】

给定一个非负整数数组,统计里面每一个数的出现次数。我们只统计到数组里最大的数。

假设 $F_{\max}$ ($F_{\max} < 10,000$) 是数组里最大的数,那么我们只统计 $\{0, 1, 2, \dots, F_{\max}\}$ 里每个数出现的次数。

输入:

第 1 行 n ($1 \leq n \leq 10,000$) 是数组的大小。

紧接着一行是数组的 n 个元素。

输出:

按顺序输出每个数的出现次数,一行一个数。如果没有出现过,则输出 0。

对于例子中的数组,最大的数是 3,因此我们只统计 $\{0, 1, 2, 3\}$ 的出现频数。

样例输入:

```

5
1 1 2 3 1

```

样例输出:

```

0
3
1
1

```

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int a[10001],i,n,x,maxn=0;

```

```
int main()
{
    memset(a,0,sizeof(0));           //频数全部初始化为 0
    cin>>n;
    for(i=1;i<=n;i++)
    {
        scanf("%d",&x);
        a[x]++;                       //统计频数,a[x]表示 x 出现的频数
        if(maxn<x)maxn=x;             //找最大值
    }
    for(i=0;i<=maxn;i++)
        printf("%d\n",a[i]);         //输出所有频数
    return 0;
}
```

15. 最长平台【1.9 编程基础之顺序查找 12】

已知一个已经从小到大排序的数组,这个数组的一个平台(Plateau)就是连续的一串值相同的元素,并且这一串元素不能再延伸。例如,在 1,2,2,3,3,3,4,5,5,6 中 1,2—2,3—3—3,4,5—5,6 都是平台。试编写一个程序,接收一个数组,把这个数组最长的平台找出来。在上面的例子中 3—3—3 就是最长的平台。

输入:

第 1 行有一个整数 n,为数组元素的个数;第 2 行有 n 个整数,整个整数之间以一个空格分开。

输出:

最长平台的长度。

样例输入:

```
10
1 2 2 3 3 3 4 5 5 6
```

样例输出:

```
3
```

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int a[10001],n,i,k=1,maxn=0;
int main()
{
    cin>>n;
    a[0]=-34567;                       //a[0]不能和 a[1]相等
    for(i=1;i<=n;i++)
    {
        scanf("%d",&a[i]);
        if(a[i]==a[i-1]) k++;           //和前一个相等,则平台长度加一
        else k=1;                       //不相等,则开始计算下一个平台
        if(k>maxn)maxn=k;               //找最长平台
    }
    cout<<maxn;
    return 0;
}
```

16. 整数去重【1.9 编程基础之顺序查找 13】

给定含有 n 个整数的序列,要求对这个序列进行去重操作。所谓去重,就是对这个序列中每个重复出现的数,只保留该数第一次出现的位置,删除其余位置。

输入:

包含两行:

第 1 行包含一个正整数 $n(1 \leq n \leq 20,000)$,表示第 2 行序列中数字的个数;

第 2 行包含 n 个整数,每个整数之间以一个空格分开。每个整数大于等于 10、小于等于 100。

输出:

一行,按照输入的顺序输出其中不重复的数字,每个整数之间用一个空格分开。

样例输入:

```
5
10 12 93 12 75
```

样例输出:

```
10 12 93 75
```

【参考程序】

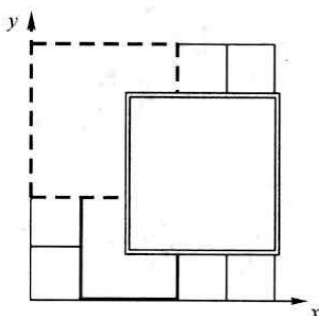
```
#include<iostream>
#include<cstring>
using namespace std;
int a[20001];
bool s[20001];
int main()
{
    int n;
    cin>>n;
    memset(s,true,sizeof(s));           //初始化为没有重复
    for (int i=1; i<=n; ++i)
        cin>>a[i];
    for (int i=1; i<=n; ++i)
    {
        if(s[i])                         //判断 a[i] 是否已被标记为不能输出
            for (int j=i+1; j<=n; ++j)   //只要找后面是否有与 a[i] 重复的数
                if (a[i]==a[j]) s[j]=false; //有出现重复的数,标记为不能输出
    }
    for (int i=1; i<=n; ++i)             //输出没重复的数
        if (s[i]==true) cout<<a[i]<<" ";
    return 0;
}
```

17. 铺地毯【1.9 编程基础之顺序查找 14】Noip2011 提高组第 1 题

为了准备一个独特的颁奖典礼,组织者在会场的一片矩形区域(可看作是平面直角坐标系的第一象限)铺上一些矩形地毯。一共有 n 张地毯,编号从 1~ n 。现在将这些地毯按照编号从小到大的顺序平行于坐标轴先后铺设,后铺的地毯覆盖在前面已经铺好的地毯之上。地毯铺设完成后,组织者想知道覆盖地面某个点的最上面的那张地毯的编号。注意:在矩形地毯边界和四个顶点上的点也算被地毯覆盖。

输入输出样例 1 说明:如下图,1 号地毯用实线表示,2 号地毯用虚线表示,3 号用双实线表示,覆盖点(2,2)的最上面一张地毯是 3 号地毯。

输入输出样例 2 说明:如下图,1 号地毯用实线表示,2 号地毯用虚线表示,3 号用双实线表示,覆盖点(4,5)的最上面一张地毯是 3 号地毯。



输入:

第 1 行, 一个整数 n , 表示总共有 n 张地毯。

接下来的 n 行中, 第 $i+1$ 行表示编号 i 的地毯的信息, 包含四个正整数 a, b, g, k , 每两个整数之间用一个空格隔开, 分别表示铺设地毯的左下角的坐标 (a, b) 以及地毯在 x 轴和 y 轴方向的长度。

第 $n+2$ 行包含两个正整数 x 和 y , 表示所求的地面的点的坐标 (x, y) 。

输出:

一行, 包含一个整数, 表示所求的地毯的编号; 若此处没有被地毯覆盖则输出 -1 。

样例输入:

样例 #1:

```
3
1 0 2 3
0 2 3 3
2 1 3 3
2 2
```

样例 #2:

```
3
1 0 2 3
0 2 3 3
2 1 3 3
4 5
```

样例输出:

样例 #1:

3

样例 #2:

-1

【参考程序】

```
#include<stdio>
#include<stdlib>
using namespace std;
int main()
{
    int n,x,y,a[10001],b[10001],g[10001],k[10001];
    scanf("%d",&n);
    for (int i=1; i<=n; ++i)
        scanf("%d%d%d%d",&a[i],&b[i],&g[i],&k[i]);
    //输入铺设地毯的左下角的坐标(a,b)以及地毯在 x 轴和 y 轴方向的长度
    scanf("%d%d",&x,&y);
    for (int i=n; i>=1; --i) //找最上面覆盖的地毯,所以从上往下找
        if (x>=a[i] && x<=a[i]+g[i] && y>=b[i] && y<=b[i]+k[i])
        {
            printf("%d\n",i); exit(0); //找到覆盖的,输出,结束
        }
    printf("-1\n"); //没找到,输出-1
    return 0;
}
```


第二节 二维数组

1. 矩阵交换行【1.8 编程基础之多维数组 01】

给定一个 5×5 的矩阵(数学上,一个 $r \times c$ 的矩阵是一个由 r 行 c 列元素排列成的矩形阵列),将第 n 行和第 m 行交换,输出交换后的结果。

输入:

共六行,前五行为矩阵的每一行元素,元素与元素之间以一个空格分开。

第 6 行包含两个整数 $m, n (1 \leq m, n \leq 5)$,以一个空格分开。

输出:

交换之后的矩阵,矩阵的每一行元素占一行,元素之间以一个空格分开。

样例输入:

```
1 2 2 1 2
5 6 7 8 3
9 3 0 5 3
7 2 1 4 6
3 0 8 2 4
1 5
```

样例输出:

```
3 0 8 2 4
5 6 7 8 3
9 3 0 5 3
7 2 1 4 6
1 2 2 1 2
```

【参考程序】

```
#include <stdio.h>
using namespace std;
int main()
{
    int n, m, a[6][6], t, i, j;
    for (i = 1; i <= 5; ++i)
        for (j = 1; j <= 5; ++j)
            scanf("%d", &a[i][j]); //输入 5×5 矩阵
    scanf("%d%d", &m, &n); //输入要交换的两行
    for (i = 1; i <= 5; ++i)
    {
        //交换这两行的每个对应元素
        t = a[m][i]; a[m][i] = a[n][i]; a[n][i] = t;
    }
    for (i = 1; i <= 5; ++i)
    {
        for (j = 1; j <= 5; ++j) printf("%d ", a[i][j]);
        printf("\n"); //输出交换后的矩阵
    }
    return 0;
}
```

2. 同行列对角线的格【1.8 编程基础之多维数组 02】

输入三个自然数 $n, i, j (1 \leq i \leq n, 1 \leq j \leq n)$, 输出在一个 $n \times n$ 格的棋盘上(行列均从 1 开始编号), 与格子 (i, j) 同行、同列、同一对角线的所有格子的位置。

如: $n=4, i=2, j=3$ 表示了棋盘中的第二行第三列的格子, 如下图:

第一列	第二列	第三列	第四列	
				第一行

		(2,3)		第二行
				第三行
				第四行

当 $n=4, i=2, j=3$ 时, 输出的结果是:

(2,1) (2,2) (2,3) (2,4)

同一行上格子的位置

(1,3) (2,3) (3,3) (4,3)

同一列上格子的位置

(1,2) (2,3) (3,4)

左上到右下对角线上的格子的位置

(4,1) (3,2) (2,3) (1,4)

左下到右上对角线上的格子的位置

输入:

一行, 三个自然数 $n(1 \leq n \leq 10), i, j$, 相邻两个数之间用单个空格隔开。

输出:

第 1 行: 从左到右输出同一行格子位置;

第 2 行: 从上到下输出同一列格子位置;

第 3 行: 从左上到右下输出同一对角线格子位置;

第 4 行: 从左下到右上输出同一对角线格子位置。

其中每个格子位置用如下格式输出: (x, y) , x 为行号, y 为列号, 采用英文标点, 中间无空格。相邻两个格子位置之间用单个空格隔开。

样例输入:

4 2 3

样例输出:

(2,1) (2,2) (2,3) (2,4)

(1,3) (2,3) (3,3) (4,3)

(1,2) (2,3) (3,4)

(4,1) (3,2) (2,3) (1,4)

【参考程序】

```
#include<cstdio>
using namespace std;
int a[1001][1001];
int main()
{
    int i,n,t,s,j,x,y;
    scanf("%d%d%d",&n,&x,&y);
    for(i=1;i<=n;++i)
        printf("(%d,%d) ",x,i);    //同一行格子位置输出
    printf("\n");
    for(i=1;i<=n;++i)
        printf("(%d,%d) ",i,y);    //同一列格子位置输出
    printf("\n");
    for(i=1;i<=n;++i)
        for(j=1;j<=n;++j)    //从左上到右下输出同一对角线格子位置
            if (x-y==i-j) printf("(%d,%d) ",i,j);
    printf("\n");
    for(i=n;i>=1;--i)
        for(j=n;j>=1;--j)    //从左下到右上输出同一对角线格子位置
            if (x+y==i+j) printf("(%d,%d) ",i,j);
    printf("\n");
    return 0;
}
```

3. 计算矩阵边缘元素之和【1.8 编程基础之多维数组 03】

输入一个整数矩阵,计算位于矩阵边缘的元素之和。所谓矩阵边缘的元素,就是第一行和最后一行的元素以及第一列和最后一列的元素。

输入:

第 1 行分别为矩阵的行数 m 和列数 n ($m < 100, n < 100$),两者之间以一个空格分开。

接下来输入的 m 行数据中,每行包含 n 个整数,整数之间以一个空格分开。

输出:

对应矩阵的边缘元素和。

样例输入:

```
3 3
3 4 1
3 7 1
2 0 1
```

样例输出:

```
15
```

【参考程序】

```
#include <cstdio>
using namespace std;
int a[1001][1001];
int main()
{
    int m,n,i,j,s=0;
    scanf("%d%d",&m,&n);
    for (i=1;i<=m;++i)
        for (j=1;j<=n;++j)
            scanf("%d",&a[i][j]);    //读入这个矩阵
    for (i=1;i<=m;++i)
        for (j=1;j<=n;++j)
            if (i==1 || j==1 || i==m || j==n) s+=a[i][j];
    //判断是否第一行,最后一行的元素或第一列,最后一列的元素,是则累加元素值
    printf("%d",s);    //对应矩阵的边缘元素总和
    return 0;
}
```

4. 计算鞍点【1.8 编程基础之多维数组 05】

给定一个 5×5 的矩阵,每行只有一个最大值,每列只有一个最小值,寻找这个矩阵的鞍点。鞍点指的是矩阵中的一个元素,它是所在行的最大值,并且是所在列的最小值。

例如:在下面的例子中(第 4 行第 1 列的元素就是鞍点,值为 8)。

```
11 3 5 6 9
12 4 7 8 10
10 5 6 9 11
8 6 4 7 2
15 10 11 20 25
```

输入:

包含一个 5 行 5 列的矩阵。

输出:

如果存在鞍点,输出鞍点所在的行、列及其值,如果不存在,输出“not found”。

样例输入：

```
11 3 5 6 9
12 4 7 8 10
10 5 6 9 11
8 6 4 7 2
15 10 11 20 25
```

样例输出：

```
4 1 8
```

【参考程序】

```
#include<cstdio>
#include<cstdlib>
using namespace std;
int main()
{
    int a[6][6],i,j,k,l; bool w,f;
    for (i=1;i<=5;++i)
    for (j=1;j<=5;++j)
        scanf("%d",&a[i][j]);          //读入 5 行 5 列的矩阵
    for (i=1;i<=5;++i)
    for (j=1;j<=5;++j)                  //枚举整个数组
    {
        for (k=1;k<=5;++k)
            if (a[i][j]<a[i][k] && k!=j) {f=false; break;}
            else f=true;                //判断这个数是否所在行的最大值
        for (l=1;l<=5;++l)
            if (a[i][j]>a[l][j] && l!=i) {w=false; break;}
            else w=true;                //判断这个数是否所在列的最小值
        if (f==true && w==true)
        { //这个数是所在行的最大值,所在列的最小值,它是鞍点,输出,结束
            printf("%d %d %d\n",i,j,a[i][j]); exit(0);
        }
    }
    printf(" not found\n");              //不存在鞍点
    return 0;
}
```

5. 图像相似度【1.8 编程基础之多维数组 06】

给出两幅相同大小的黑白图像(用 0—1 矩阵)表示,求它们的相似度。说明:若两幅图像在相同位置上的像素点颜色相同,则称它们在该位置具有相同的像素点。两幅图像的相似度定义为相同像素点数占总像素点数的百分比。

输入：

第 1 行包含两个整数 m 和 n ($1 \leq m \leq 100, 1 \leq n \leq 100$),表示图像的行数和列数,中间用单个空格隔开。

之后 m 行,每行 n 个整数 0 或 1,表示第一幅黑白图像上各像素点的颜色。相邻两个数之间用单个空格隔开。

之后 m 行,每行 n 个整数 0 或 1,表示第二幅黑白图像上各像素点的颜色。相邻两个数之间用单个空格隔开。

输出:

一个实数,表示相似度(以百分比的形式给出),精确到小数点后两位。

样例输入:

```
3 3
1 0 1
0 0 1
1 1 0
1 1 0
0 0 1
0 0 1
```

样例输出:

```
44.44
```

【参考程序】

```
#include <cstdio>
using namespace std;
int a[101][101], b[101][101];
int main()
{
    int m, n;
    double s = 0;
    scanf("%d %d", &m, &n);
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            scanf("%d", &a[i][j]);
    for (int i = 1; i <= m; ++i)
        for (int j = 1; j <= n; ++j)
            scanf("%d", &b[i][j]); // 读入两个矩阵
    for (int i = 1; i <= m; ++i) // 在相同位置上的像素点颜色相同, 计数器+1
        for (int j = 1; j <= n; ++j)
            if (a[i][j] == b[i][j]) s++;
    printf("%.2lf\n", s / (n * m) * 100); // 输出相同个数所占百分比
    return 0;
}
```

6. 矩阵加法【1.8 编程基础之多维数组 08】

输入两个 n 行 m 列的矩阵 A 和 B , 输出它们的和 $A+B$ 。

输入:

第 1 行包含两个整数 n 和 m ($1 \leq n \leq 100, 1 \leq m \leq 100$), 表示矩阵的行数和列数。

接下来 n 行, 每行 m 个整数, 表示矩阵 A 的元素。

接下来 n 行, 每行 m 个整数, 表示矩阵 B 的元素。

相邻两个整数之间用单个空格隔开, 每个元素均在 $1 \sim 1000$ 之间。

输出:

n 行, 每行 m 个整数, 表示矩阵加法的结果。相邻两个整数之间用单个空格隔开。

样例输入:

```
3 3
1 2 3
1 2 3
```

1 2 3
1 2 3
4 5 6
7 8 9

样例输出:

2 4 6
5 7 9
8 10 12

【参考程序】

```
#include<cstdio>
using namespace std;
int a[100][100],b[100][100];
int main()
{
    int n,m;
    scanf("%d %d",&n,&m);
    for (int i=1;i<=n;++i)           //读入两个矩阵
        for (int j=1;j<=m;++j) scanf("%d",&a[i][j]);
    for (int i=1;i<=n;++i)
        for (int j=1;j<=m;++j) scanf("%d",&b[i][j]);
    for (int i=1;i<=n;++i)
    {
        for (int j=1;j<=m;++j)       //输出两个矩阵中相同位置两个元素的和
            printf("%d",a[i][j]+b[i][j]); //中间以空格分开
        printf("\n");                //输出一行后换行
    }
    return 0;
}
```

7. 矩阵乘法【1.8 编程基础之多维数组 09】

计算两个矩阵的乘法。 $n \times m$ 阶的矩阵 A 乘以 $m \times k$ 阶的矩阵 B 得到的矩阵 C 是 $n \times k$ 阶的,且 $C[i][j]=A[i][0] * B[0][j]+A[i][1] * B[1][j]+\dots+A[i][m-1] * B[m-1][j]$ ($C[i][j]$ 表示 C 矩阵中第 i 行第 j 列元素)。

输入:

第 1 行为 n, m, k , 表示 A 矩阵是 n 行 m 列, B 矩阵是 m 行 k 列, n, m, k 均小于 100。

然后先后输入 A 和 B 两个矩阵, A 矩阵 n 行 m 列, B 矩阵 m 行 k 列, 矩阵中每个元素的绝对值不会大于 1000。

输出:

输出矩阵 C , 一共 n 行, 每行 k 个整数, 整数之间以一个空格分开。

样例输入:

3 2 3
1 1
1 1
1 1
1 1 1
1 1 1

样例输出:

2 2 2

第三节 字符类型和字符数组

1. 统计数字字符个数【1.7 编程基础之字符串 01】

输入一行字符,统计出其中数字字符的个数。

输入:

一行字符串,总长度不超过 255。

输出:

一行,输出字符串里面数字字符的个数。

样例输入:

Peking University is set up at 1898.

样例输出:

4

【参考程序 1】

```
#include <iostream>
#include <cstdio>
#include <cstring>                                //使用 strlen()须调用 cstring 库
using namespace std;
int main()
{
    char ch[256];int num=0,l;
    gets(ch);                                    //用 gets()读入字符数组
    l=strlen(ch);                                //计算字符串的长度
    for(int i=0;i<l;i++)
        if(ch[i]>='0'&&ch[i]<='9')              //判断是否为数字字符
            num++;                                //累加数字字符的个数
    cout<<num;                                  //输出数字字符的个数
    return 0;
}
```

【参考程序 2】

```
#include<string>                                //使用 getline()须调用 string 库
#include<iostream>
using namespace std;
int main()
{
    string s; int i,t=0;
    getline(cin,s);                              //用 getline()读入整行字符串
    for(i=0;i<s.size();i++)
    {
        if((s[i]>='0')&&(s[i]<='9'))              //判断是否为数字字符
            t++;                                //累加数字字符的个数
    }
    cout<<t<<endl;                              //输出数字字符的个数
    return 0;
}
```

2. 找第一个只出现一次的字符【1.7 编程基础之字符串 02】

给定一个只包含小写字母的字符串,请你找到第一个仅出现一次的字符。如果没有,输出 no。

输入:

一个字符串,长度小于 100,000。

输出:

第一个仅出现一次的字符,若没有则输出 no。

样例输入:

abcbabd

样例输出:

c

【参考程序 1】

```
#include<cstring>
#include<cstdio>
#include<iostream>
using namespace std;
char s[100000];
int c[26];
int main()
{
    gets(s); int l=strlen(s);
    for (int i=0; i<l; ++i)
        c[s[i]-'a']++;
    for (int i=0; i<l; ++i)
        if (c[s[i]-'a']==1)
        {
            printf("%c\n",s[i]);
            return 0;
        }
    printf("no\n");
    return 0;
}
```

//记录是否只出现一次

//找到第一个仅出现一次的字符

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
int main()
{
    int i,j,l,k; char a[100000];
    scanf("%s",a);
    l=strlen(a);
    for (i=0;i<l;++i)
    {
        k=0;
        for (j=0;j<l;++j)
```

//使用 exit 须调用 cstdlib 库

//使用 scanf 读入字符数组

// k 存储第 i 个字符的出现次数,先置 0

//从头找与第 i 个字符相同的字符


```

        if(a[i]==a[j])k++; //找到相同的字符就累计其出现次数
        if(k==1)
        {
            printf("%c ",a[i]);
            exit(0);
        } //如果第 i 个字符只出现一次就输出并退出程序,不用再找
    }
    cout<<" no "<<endl; //如果没有找到只出现一次的字符,就输出 no
    return 0;
}

```

3. 基因相关性【1.7 编程基础之字符串 03】

为了获知基因序列在功能和结构上的相似性,经常需要将几条不同序列的 DNA 进行比对,以判断该比对的 DNA 是否具有相关性。

现比对两条长度相同的 DNA 序列。定义两条 DNA 序列相同位置的碱基为一个碱基对,如果一个碱基对中的两个碱基相同的话,则称为相同碱基对。接着计算相同碱基对占总碱基对数量的比例,如果该比例大于等于给定阈值时则判定该两条 DNA 序列是相关的,否则不相关。

输入:

有三行,第 1 行是用来判定出两条 DNA 序列是否相关的阈值,随后 2 行是两条 DNA 序列(长度不大于 500)。

输出:

若两条 DNA 序列相关,则输出“yes”,否则输出“no”。

样例输入:

```

0.85
ATCGCCGTAAGTAACGGTTTTAAATAGGCC
ATCGCCGGAAGTAACGGTCTTAAATAGGCC

```

样例输出:

yes

【参考程序 1】

```

#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

int main()
{
    int len, tot=0;
    double x;
    char s1[501], s2[501];
    cin >> x >> s1 >> s2;
    len=strlen(s1); //计算字符串长度
    for (int i=0; i<len; ++i) //从头枚举 a 字符串与 b 字符串的碱基是否相同
        if (s1[i]==s2[i])
            tot++; //统计相同碱基对的个数
    if ((double)tot / len >= x) //tot/len 为相同碱基对占总碱基对数量的比例
        cout << " yes " << endl;
    else

```

2 2 2

2 2 2

【参考程序】

```

#include<cstdio>
using namespace std;
int a[100][100],b[100][100],c[100][100];
int main()
{
    int n,m,k;
    scanf("%d %d %d",&n,&m,&k);
    for (int i=1;i<=n;++i)           //读入两个矩阵
        for (int j=1;j<=m;++j) scanf("%d",&a[i][j]);
    for (int i=1;i<=m;++i)
        for (int j=1;j<=k;++j) scanf("%d",&b[i][j]);
    for (int i=1;i<=n;++i)
        for (int j=1;j<=k;++j)
            for (int t=1;t<=m;++t)
                c[i][j]+=a[i][t]*b[t][j];    //累加 c[i][j] 的值
    for (int i=1;i<=n;++i)
    {
        for (int j=1;j<=k;++j)
            printf("%d ",c[i][j]);          //输出 n * k 阶的 c 数组
        printf("\n");
    }
    return 0;
}

```

8. 矩阵转置【1.8 编程基础之多维数组 10】

输入一个 n 行 m 列的矩阵 A , 输出它的转置 A^T 。

输入:

第 1 行包含两个整数 n 和 m ($1 \leq n \leq 100, 1 \leq m \leq 100$), 表示矩阵 A 的行数和列数。接下来 n 行, 每行 m 个整数, 表示矩阵 A 的元素。相邻两个整数之间用单个空格隔开, 每个元素均在 $1 \sim 1000$ 之间。

输出:

m 行, 每行 n 个整数, 为矩阵 A 的转置。相邻两个整数之间用单个空格隔开。

样例输入:

```

3 3
1 2 3
4 5 6
7 8 9

```

样例输出:

```

1 4 7
2 5 8
3 6 9

```

【参考程序】

```

#include<cstdio>
using namespace std;
int a[1001][1001];
int main()
{
    int m,n,i,j;
    scanf("%d%d",&n,&m);

```

```

for (i=1;i<=n;++i)
    for (j=1;j<=m;++j)
        scanf("%d",&a[i][j]);           //读入矩阵
for (i=1;i<=m;++i)
{
    for (j=1;j<=n;++j)                   //输出矩阵 A 的转置,即交换 a[i][j]和 a[j][i]
        printf("%d",a[j][i]);
    printf("\n");                         //输出一行,即换行
}
return 0;
}

```

9. 图像旋转【1.8 编程基础之多维数组 11】

输入一个 n 行 m 列的黑白图像,将它顺时针旋转 90° 后输出。

输入:

第 1 行包含两个整数 n 和 m ($1 \leq n \leq 100, 1 \leq m \leq 100$),表示图像包含像素点的行数和列数。

接下来 n 行,每行 m 个整数,表示图像的每个像素点灰度。相邻两个整数之间用单个空格隔开,每个元素均在 $0 \sim 255$ 之间。

输出:

m 行,每行 n 个整数,为顺时针旋转 90° 后的图像。相邻两个整数之间用单个空格隔开。

样例输入:

```

3 3
1 2 3
4 5 6
7 8 9

```

样例输出:

```

7 4 1
8 5 2
9 6 3

```

【参考程序】

```

#include<stdio>
using namespace std;
int a[1001][1001];
int main()
{
    int m,n,i,j;
    scanf("%d%d",&n,&m);
    for (i=1;i<=n;++i)           //读入矩阵
        for (j=1;j<=m;++j) scanf("%d",&a[i][j]);
    for (i=1;i<=m;++i)
    { //按从第一列到最后一列,从最后一行到第一行的规律输出
        for (j=n;j>=1;--j) printf("%d",a[j][i]);
        printf("\n");           //输出一行,换行
    }
    return 0;
}

```

10. 图像模糊处理【1.8 编程基础之多维数组 13】

给定 m 行 n 列的图像各像素点的灰度值,要求用如下方法对其进行模糊化处理:

(1) 四周最外侧的像素点灰度值不变;

(2) 中间各像素点新灰度值为该像素点及其上下左右相邻四个像素点原灰度值的平均值(舍入到最接近的整数)。

输入:

第 1 行包含两个整数 n 和 m ($1 \leq n \leq 100, 1 \leq m \leq 100$), 表示图像包含像素点的行数和列数。

接下来 n 行, 每行 m 个整数, 表示图像的每个像素点灰度。相邻两个整数之间用单个空格隔开, 每个元素均在 $0 \sim 255$ 之间。

输出:

m 行, 每行 n 个整数, 为模糊处理后的图像每个像素点的灰度。相邻两个整数之间用单个空格隔开。

样例输入:

```
4 5
100 0 100 0 50
50 100 200 0 0
50 50 100 100 200
100 100 50 50 100
```

样例输出:

```
100 0 100 0 50
50 80 100 60 0
50 80 100 90 200
100 100 50 50 100
```

【参考程序】

```
#include <cstdio>
using namespace std;
int a[1001][1001];
double b[1001][1001];
int main()
{
    int n, m, i, j;
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; ++i)
        for (j = 1; j <= m; ++j)
        {
            scanf("%d", &a[i][j]);
            b[i][j] = a[i][j]; //需要分别保存原图和模糊后的图像
        }
    for (i = 2; i <= n - 1; ++i)
        for (j = 2; j <= m - 1; ++j)
            b[i][j] = (a[i - 1][j] + a[i][j - 1] + a[i + 1][j] + a[i][j + 1] + a[i][j]) / 5.0;
    //各像素点新灰度值为该像素点及其上下左右相邻四个像素点原灰度值的平均值
    //结果实数, 所以 / 5.0
    for (i = 1; i <= n; ++i)
    { //保留 1 位小数, 输出模糊后的图像
        for (j = 1; j <= m; ++j) printf("%.0lf ", b[i][j]);
        printf("\n");
    }
    return 0;
}
```

```

    cout << " no " << endl;
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<string>
using namespace std;
int main()
{
    string a,b; double j,n;           //比例可能为小数,故用 double 类型
    int i,l;
    cin>>n;                           //读入给定阈值
    cin>>a>>b;
    l=a.size();
    for(i=0;i<l;i++)                 //从头枚举 a 字符串与 b 字符串的碱基是否相同
        if(a[i]==b[i])j++;           //统计相同碱基对的个数
    if(j/l>=n)cout<<" yes "<<endl;    //j/l 为相同碱基对占总碱基对数量的比例
                                        //若该比例大于等于给定阈值输出 yes
    else cout<<" no "<<endl;           //否则输出 no
    return 0;
}

```

4. 石头剪子布【1.7 编程基础之字符串 04】

石头剪子布,是一种猜拳游戏。起源于中国,然后传到日本、朝鲜等地,随着亚欧贸易的不断发展它传到了欧洲,到了近现代逐渐风靡世界。简单明了的规则,使得石头剪子布没有任何规则漏洞可钻,单次玩法比拼运气,多回合玩法比拼心理博弈,使得石头剪子布这个古老的游戏同时拥有“意外”与“技术”两种特性,深受世界人民喜爱。

游戏规则:石头打剪刀,布包石头,剪刀剪布。

现在,需要你写一个程序来判断石头剪子布游戏的结果。

输入:

第 1 行是一个整数 $n(1 \leq n \leq 100)$,表示一共进行了 n 次游戏。

接下来 n 行的每一行包括两个字符串,表示游戏参与者 Player1,Player2 的选择(石头、剪子或者布):

s1 s2

字符串之间以空格隔开,s1,s2 只可能取值在{"Rock", "Scissors", "Paper"} (大小写敏感)中。

输出:

输出包括 n 行,每一行对应一个胜利者(Player1 或者 Player2),或者游戏出现平局,则输出 Tie。

样例输入:

```

3
Rock Scissors
Paper Paper
Rock Paper

```

样例输出:

```

Player1
Tie
Player2

```

【参考程序 1】

```

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    char s1[10],s2[10];
    int n;
    cin>>n;
    for (int i=1;i<=n;++i)
    {
        scanf("%s%s",s1,s2);
        if(s1[0]=='S'&&.&s2[0]=='P' || s1[0]=='P'&&.&s2[0]=='R' || s1[0]=='R'
        '&.&s2[0]=='S') cout<<" Player1 "<<endl; //判定 Player1 赢 Player2 的三种情况
        else
            if(s2[0]=='S'&&.&s1[0]=='P' || s2[0]=='P'&&.&s1[0]=='R' || s2[0]=='R'
            '&.&s1[0]=='S') cout<<" Player2 "<<endl; //判定 Player2 赢 Player1 的三种情况
        else
            cout<<" Tie "<<endl; //否则就是平局
    }
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
using namespace std;
int main()
{
    int n,i,j,k;
    string a[101],b[101],s[101];
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i]>>b[i];
    for(i=1;i<=n;i++)
    {
        if(a[i]==b[i]) cout<<" Tie "<<endl; // 判定 Player1 打平 Player2 的情况
        else
            if((a[i]==" Rock " &&.&b[i]==" Scissors ") || (a[i]==" Scissors " &&.&b[i]=
            =" Paper ") || (a[i]==" Paper " &&.&b[i]==" Rock "))
                cout<<" Player1 "<<endl; //判定 Player1 赢 Player2 的三种情况
            else cout<<" Player2 "<<endl; //否则即为 Player2 赢
    }
    return 0;
}

```

5. 输出亲朋字符串【1.7 编程基础之字符串 05】

编写程序,求给定字符串 s 的亲朋字符串 s_1 。

亲朋字符串 s_1 定义如下:给定字符串 s 的第一个字符的 ASCII 值加第二个字符的

ASCII 值,得到第一个亲朋字符;给定字符串 s 的第二个字符的 ASCII 值加第三个字符的 ASCII 值,得到第二个亲朋字符;依此类推,直到给定字符串 s 的倒数第二个字符。亲朋字符串的最后一个字符由给定字符串 s 的最后一个字符 ASCII 值加 s 的第一个字符的 ASCII 值。

输入:

一行,一个长度大于等于 2,小于等于 100 的字符串。字符串中每个字符的 ASCII 值不大于 63。

输出:

一行,为变换后的亲朋字符串。输入保证变换后的字符串只有一行。

样例输入:

1234

样例输出:

cege

【参考程序】

```
#include<string>
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    char a[201]; int t; char y;
    int i,l;
    gets(a);
    l=strlen(a);
    for(i=0;i<=l-2;i++) //除最后一位,其余按顺序变换
    {
        t=a[i]+a[i+1]; y=t; //将字符变为亲朋字符
        cout<<y; //输出变换后的亲朋字符
    }
    t=a[0]+a[l-1]; y=t; //第一个和最后一个字符变换为最后一位的亲朋字符
    cout<<y<<endl; //输出最后一位亲朋字符
    return 0;
}
```

6. 合法 C 标识符查核【1.7 编程基础之字符串 06】

给定一个不包含空白符的字符串,请判断是否是 C 语言合法的标识符号(注:题目保证这些字符串一定不是 C 语言的保留字)。

C 语言标识符要求:

- (1)非保留字;
- (2)只包含字母、数字及下划线("_");
- (3)不以数字开头。

输入:

一行,包含一个字符串,字符串中不包含任何空白字符,且长度不大于 20。

输出:

一行,如果它是 C 语言的合法标识符,则输出 yes,否则输出 no。

样例输入:

RKPEGX9R;TWyYcp

样例输出：

no

【参考程序】

```
#include<iostream>
#include<string>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    int i,l,h;char a[256];
    bool f=true,b=true;
    gets(a);
    l=strlen(a)-1;
    if((a[0]>='0')&&(a[0]<='9'))
        b=false;           //如果以数字开头,不满足第3个条件,将f变为false
    for(i=0;i<=l;i++)
    {
        if((a[i]>='0'&&a[i]<='9')||(a[i]>='a'&&a[i]<='z')||(a[i]>='A'
            &&a[i]<='Z')||(a[i]=='_')); //如果只包含字母、数字及下划线
        else f=false;           //否则不满足第2个条件,将b变为false
    }
    if((f==true)&&(b==true))printf("yes\n");
    //如果f和b都为true,说明满足所有条件,就输出yes
    else printf("no\n");       //否则输出no
    return 0;
}
```

7. 配对碱基链【1.7 编程基础之字符串 07】

脱氧核糖核酸(DNA)由两条互补的碱基链以双螺旋的方式结合而成。而构成DNA的碱基共有4种,分别为腺嘌呤(A)、鸟嘌呤(G)、胸腺嘧啶(T)和胞嘧啶(C)。我们知道,在两条互补碱基链的对应位置上,腺嘌呤总是和胸腺嘧啶配对,鸟嘌呤总是和胞嘧啶配对。你的任务就是根据一条单链上的碱基序列,给出对应的互补链上的碱基序列。

输入：

一个字符串,表示一条碱基链。这个字符串只含有大写字母A、T、G、C,分别表示腺嘌呤、胸腺嘧啶、鸟嘌呤和胞嘧啶。字符串长度不超过255。

输出：

一个只含有大写字母A、T、G、C的字符串,为与输入的碱基链互补的碱基链。

样例输入：

ATATGGATGGTGTGGCTCTG

样例输出：

TATACCTACCACAAACCGAGAC

【参考程序 1】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
char s[255];
```



```

int main()
{
    int i,l;
    gets(s);
    l=strlen(s);
    for(i=0;i<l;++i)           //输出第 i 个碱基的配对碱基
    {
        if (s[i]=='A') printf(" T ");
        if (s[i]=='T') printf(" A ");
        if (s[i]=='G') printf(" C ");
        if (s[i]=='C') printf(" G ");
    }
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
string s;
int main()
{
    int i,l;
    getline(cin,s);
    l=s.size();
    for(i=0;i<l;++i)           //输出第 i 个碱基的配对碱基
    {
        if (s[i]=='A') printf(" T ");
        if (s[i]=='T') printf(" A ");
        if (s[i]=='G') printf(" C ");
        if (s[i]=='C') printf(" G ");
    }
    return 0;
}

```

8. 密码翻译【1.7 编程基础之字符串 09】

在情报传递过程中,为了防止情报被截获,往往需要对情报用一定的方式加密,简单的加密算法虽然不足以完全避免情报被破译,但仍然能防止情报被轻易地识别。我们给出一种最简的加密方法,对给定的一个字符串,把其中从 a—y,A—Y 的字母用其后继字母替代,把 z 和 Z 用 a 和 A 替代,其他非字母字符不变,则可得到一个简单的加密字符串。

输入:

一行,包含一个字符串,长度小于 80 个字符。

输出:

一行,字符串的加密字符串。

样例输入:

Hello! How are you!

样例输出:

Ifmmp! Ipx bsf zpv!

【参考程序 1】

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
char s[80];int l;
int main()
{
    gets(s);
    l=strlen(s);
    for (int i=0;i<l;++i)
    {
        if (s[i]=='z' || s[i]=='Z') s[i]=s[i]-25; //把 z 和 Z 用 a 和 A 替代
        else if (s[i]>='a' && s[i]<='z' || s[i]>='A' && s[i]<='Z')
            s[i]=s[i]+1; //其余用后继字母替代
        cout<<s[i]; //输出加密后的字母
    }
    return 0;
}
```

【参考程序 2】

```
#include<cstdio>
#include<iostream>
#include<string>
using namespace std;
string s;
int main()
{
    getline(cin,s);
    int len=s.size();
    for (int i=0; i < len; i++)
    {
        if (s[i]=='z' || s[i]=='Z')
            printf("%c",s[i]-25); //把 z 和 Z 用 a 和 A 替代并输出
        else if( ( s[i] >='a' && s[i] < 'z' ) || ( s[i] >='A' && s[i] < 'Z' ) )
            printf("%c",s[i]+1); //其余输出后继字母
        else printf("%c", s[i]); //直接输出
    }
    return 0;
}
```

9. 加密的病历单【1.7 编程基础之字符串 12】

小英是药学专业大三的学生，暑假期间获得了去医院药房实习的机会。

在药房实习期间，小英扎实的专业基础获得了医生的一致好评，得知小英在计算概论中取得过好成绩后，主任又额外交给她一项任务，解密抗战时期被加密过的一些伤员的名单。

经过研究，小英发现了如下加密规律(括号中是一个“原文→密文”的例子)

- (1)原文中所有的字符都在字母表中被循环左移了三个位置(dec→abz);
- (2)逆序存储(abcd→dcba);
- (3)大小写反转(abXY→ABxy)。

输入:

一个加密的字符串(长度小于 50 且只包含大小写字母)。

输出:

解密后的字符串。

样例输入:

GSOOWFASOq

样例输出:

Trvdizrrvj

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
char s[10001];
int main()
{
    int i,n,t;
    gets(s);
    n=strlen(s);
    for(i=0;i<n;i++)
    {
        if (s[i]>='a' && s[i]<='z') //小写字母与大写字母分别判断,此为小写的情况
        {
            t=s[i];t+=3;           //向右移动三个位置恢复原文
            if(t>122) t-=26;        //若为 x,y,z 就恢复为 a,b,c
            t-=32;s[i]=t;          //将小写转为大写
        }
        else if (s[i]>='A' && s[i]<='Z') //此为大写的情况
        {
            t=s[i];t+=3;           //向右移动三个位置恢复原文
            if (t>90) t-=26;        //同理,若为 X,Y,Z,就恢复为 A,B,C
            t+=32;s[i]=t;          //将大写转为小写
        }
    }
    for(i=n-1;i>=0;i--) cout<<s[i]; //逆序输出
    return 0;
}
```

【参考程序 2】

```
#include<string>
using namespace std;
string s;
int main()
{
    int i,n,t;
    getline(cin,s);
    n=s.size();
    for(i=0; i<n; i++)
```

```

{
    if (s[i]>='a' && s[i]<='z')//小写字母与大写字母分别判断,此为小写的情况
    {
        s[i]+=3;           //向右移动三个位置恢复原文
        if(s[i]>122) s[i]-=26; //若为 x,y,z 就恢复为 a,b,c
        s[i]-=32;          //将小写转为大写
    }
    else if (s[i]>='A' && s[i]<='Z')//此为大写的情况
    {
        s[i]+=3;           //向右移动三个位置恢复原文
        if (s[i]>90) s[i]-=26; //同理,若为 X,Y,Z,就恢复为 A,B,C
        s[i]+=32;          //将大写转为小写
    }
}
for(i=n-1;i>=0;i--)
    printf("%c",s[i]);    //逆序输出
return 0;
}

```

10. 将字符串中的小写字母转换成大写字母【1.7 编程基础之字符串 13】

给定一个字符串,将其中所有的小写字母转换成大写字母。

输入:

一行,包含一个字符串(长度不超过 100,可能包含空格)。

输出:

转换后的字符串。

样例输入:

helloworld123Ha

样例输出:

HELLOWORLD123HA

【参考程序 1】

```

#include<stdio>
#include<cstring>
#include<iostream>
using namespace std;
char s[100]; int i,len;
int main()
{
    gets(s);
    len=strlen(s);
    for(i=0;i<len;i++)
    {
        if(islower(s[i]))           //islower 函数判断是否为小写字母
            s[i]=toupper(s[i]);    //toupper 函数将小写字母转为大写
    }
    printf("%s",s);
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<string>
#include<cstdio>
using namespace std;
string s;
int l;
int main()
{
    getline(cin,s);
    l=s.size();
    for (int i=0; i<l;++i)
    {
        if ((s[i]>='a') && (s[i]<='z'))
            s[i]=s[i]-32;           //若为小写字母则转为大写字母
        printf("%c ",s[i]);
    }
    return 0;
}

```

11. 整理药名【1.7 编程基础之字符串 15】

医生在书写药品名的时候经常不注意大小写,格式比较混乱。现要求你写一个程序将医生书写混乱的药品名整理成统一规范的格式,即药品名的第一个字符如果是字母则大写,其他字母小写。如将 ASPIRIN、aspirin 整理成 Aspirin。

输入:

第 1 行一个数字 n,表示有 n 个药品名要整理,n 不超过 100。

接下来 n 行,每行一个单词,长度不超过 20,表示医生手书的药品名。药品名由字母、数字和“—”组成。

输出:

n 行,每行一个单词,对应输入的药品名的规范写法。

样例输入:

```

4
AspiRin
cisapride
2—PENICILLIN
Cefradine—6

```

样例输出:

```

Aspirin
Cisapride
2—penicillin
Cefradine—6

```

【参考程序 1】

```

#include<cstdio>
#include<cstring>
using namespace std;
char str[25];
int n;

```

```
int main()
{
    scanf("%d",&n);
    for (int i=0;i<n;i++)
    {
        scanf("%s",str);
        if(str[0]>='a'&&str[0]<='z')
            str[0]-=32;           //将第一个字母转大写
        int l=strlen(str);
        for (int i=1;i<l;i++)
            if(str[i]>='A'&&str[i]<='Z')
                str[i]+=32;       //若该药名除开头外为大写字母,则改为小写
        puts(str);
    }
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstring>
using namespace std;
string s[101],ss;
int main()
{
    int i,n,t,j,k,l;
    cin>>n;
    for(i=1;i<=n;++i) cin>>s[i];
    for(i=1;i<=n;++i)
    {
        ss=s[i];
        if (ss[0]>='a'&&ss[0]<='z')
            ss[0]-=32;           //若以小写字母开头就转为大写
        l=ss.size();
        for(j=1;j<l;++j)
            if (ss[j]>='A'&&ss[j]<='Z')
                ss[j]+=32;       //若该药名除开头外为大写字母,则改为小写
        s[i]=ss;
    }
    for(i=1;i<=n;++i) cout<<s[i]<<endl;
    return 0;
}
```

12. 验证子串【1.7 编程基础之字符串 18】

输入两个字符串,验证其中一个字符串是否为另一个字符串的子串。

输入:

两个字符串,每个字符串占一行,长度不超过 200 且不含空格。

输出:

若第 1 个字符串 s1 是第 2 个字符串 s2 的子串,则输出(s1) is substring of (s2);

否则,若第 2 个字符串 s2 是第 1 个字符串 s1 的子串,输出(s2) is substring of (s1);

否则,输出 No substring。

样例输入:

abc

dddn cabca

样例输出:

abc is substring of dddncabca

【参考程序 1】

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
char s1[200],s2[200];
int main()
{
    int i,n,t;
    gets(s1); gets(s2);
    if (strstr(s2,s1))                //strstr 函数判断子串
    {
        printf("%s is substring of %s\n",s1,s2);
        exit(0);
    }
    //如果是另一个串的子串则输出并退出,不需继续判断
    if (strstr(s1,s2))
    {
        printf("%s is substring of %s\n",s2,s1);
        exit(0);
    }
    //同理
    printf(" No substring\n");    //若不互为子串,则输出 No substring
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
string a,b;
int main()
{
    cin>>a>>b;
    if(a.find(b)!=a.npos)            //使用函数查找
        cout<<b<<" is substring of "<<a<<"\n ";
    else if(b.find(a)!=b.npos)
        cout<<a<<" is substring of "<<b<<"\n ";
    else cout<<" No substring\n ";
    return 0;
}
```

13. 删除单词后缀【1.7 编程基础之字符串 20】

给定一个单词,如果该单词以 er、ly 或者 ing 后缀结尾,则删除该后缀(题目保证删除后缀后的单词长度不为 0),否则不进行任何操作。

输入:

一行,包含一个单词(单词中间没有空格,每个单词最大长度为 32)。

输出:

按照题目要求处理后的单词。

样例输入:

referer

样例输出:

refer

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int i,l;
char s[32];
int main()
{
    scanf("%s",s);
    l=strlen(s);
    if (strcmp(&s[l-3],"ing")==0) s[l-3]='\0';
    //strcmp 函数判定字符串后三位是否为 ing,若是则删去
    if ((strcmp(&s[l-2],"er")==0) || (strcmp(&s[l-2],"ly")==0)) s[l-2]='\0';
    //判断字符串后两位是否为 er 或 ly,若是则删去
    puts(s);
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int i,l;
string s;
int main()
{
    getline(cin,s);
    l=s.size();
    if (strcmp(&s[l-3],"ing")==0) l-=3;
    //strcmp 函数判定字符串后三位是否为 ing,若是则删去
    else if ((strcmp(&s[l-2],"er")==0) || (strcmp(&s[l-2],"ly")==0)) l-=2;
    //判断字符串后两位是否为 er 或 ly,若是则删去
    for (int i=0;i<l;++i) cout<<s[i]; //puts 输出字符数组
    printf("\n");
    return 0;
}
```

14. 单词的长度【1.7 编程基础之字符串 24】

输入一行单词序列,相邻单词之间由 1 个或多个空格间隔,请对应地计算各个单词的长度。

注意:如果有标点符号(如连字符、逗号),标点符号算作与之相连的词的一部分。没有被空格间开的符号串,都算作单词。

输入:

一行单词序列,最少 1 个单词,最多 300 个单词,单词之间用至少 1 个空格间隔。单词序列总长度不超过 1000。

输出:

依次输出对应单词的长度,之间以逗号间隔。

样例输入:

She was born in 1990-01-02 and from Beijing city.

样例输出:

3,3,4,2,10,3,4,7,5

【参考程序 1】

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int t=0,l;
char s[1000];
int main()
{
    gets(s);
    l=strlen(s);
    for (int i=0;i<l;++i)
    if (s[i]!=' ') t++;          //若不是空格,则继续累加目前单词的长度
    else                        //若是空格则输出空格前已累加的单词长度
    {
        if (t>0) cout<<t<<" ";
        t=0;
    }
    cout<<t<<endl; //因最后一个单词后没有空格,故在此处输出最后一个单词的长度
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int i,n,t;
bool f=1;
string s;
int main()
{
    while (cin>>s)          //不断读入单词
    {
        if (f) f=0; //第一次 while 循环时 if 成立,不执行 else 后,输出逗号的命令
        else cout<<" "; //之后 f=0 使 if 不成立,便在单词前先输出逗号
    }
}
```

```

    cout<<s.size();          //输出单词的长度
}
return 0;
}

```

15. 最长最短单词【1.7 编程基础之字符串 25】

输入 1 行句子(不多于 200 个单词,每个单词长度不超过 100),只包含字母、空格和逗号。单词由至少一个连续的字母构成,空格和逗号都是单词间的间隔。

试输出第 1 个最长的单词和第 1 个最短单词。

输入:

一行句子。

输出:

第 1 行,第 1 个最长的单词。

第 2 行,第 1 个最短的单词。

样例输入:

I am studying Programming language C in Peking University.

样例输出:

Programming

I

提示:

如果所有单词长度相同,那么第一个单词既是最长单词也是最短单词。

【参考程序 1】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    char s[25000];
    int t=0,l,maxl=0,minl=101,maxi,mini;
    gets(s);
    l=strlen(s);
    s[l]=' ';          //在字符串后添加一个空格,方便判定最后一个单词
    for (int i=0; i<=l; ++i)
    {
        if ((s[i] != ' ') && (s[i] != ','))
            t++;          //若不是空格或逗号,则继续累加目前单词的长度
        else if (t>0)      //若是空格或逗号
        {
            if (t>maxl)    //判断是否需要更新目前的最长单词
            {
                maxl=t;    //maxl 存储最长的单词长度
                maxi=i-t;   //maxi 存储最长的单词在句子中的起始位置
            }
            if (t<minl)    //判断是否需要更新目前的最短单词
            {
                minl=t;    //minl 存储最短的单词长度
            }
        }
    }
}

```

```

        mini=i-t;           //mini 存储最短的单词在句子中的起始位置
    }
    t=0;                    //单词长度清零
}
}
for(int i=maxi; i<=maxi+maxl-1; i++)
    cout<<s[i];             //输出最长单词
    cout<<endl;
for(int i=mini; i<=mini+minl-1; i++)
    cout<<s[i];             //输出最短单词
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    string s;
    int t=0,l,maxl=0,minl=101,maxi,mini;
    getline(cin,s);
    l=s.size();
    s[l]="";                 //在字符串后添加一个空格,方便判定最后一个单词
    for (int i=0; i<=l; ++i)
    {
        if ((s[i]! ="") && (s[i]! =',')) t++;
        //若不是空格或逗号,则继续累加目前单词的长度
        else if (t>0)
        //若是空格或逗号
        {
            if(t>maxl)
            //判断是否需要更新目前的最长单词
            {
                maxl=t;           //maxl 存储最长的单词长度
                maxi=i-t;         //maxi 存储最长的单词在句子中的起始位置
            }
            if(t<minl)
            //判断是否需要更新目前的最短单词
            {
                minl=t;           //minl 存储最短的单词长度
                mini=i-t;         //mini 存储最短的单词在句子中的起始位置
            }
            t=0;                  //单词长度清零
        }
    }
    for(int i=maxi; i<=maxi+maxl-1; i++)
        cout<<s[i];             //输出最长单词
    cout<<endl;
}

```

```
for(int i=mini; i<=mini+minl-1; i++)
    cout<<s[i];          //输出最短单词
return 0;
```

16. 单词翻转【1.7 编程基础之字符串 27】

输入一个句子(一行),将句子中的每一个单词翻转后输出。

输入:

只有一行,为一个字符串,不超过 500 个字符。单词之间以空格隔开。

输出:

翻转每一个单词后的字符串,单词之间的空格需与原文一致。

样例输入:

hello world

样例输出:

olleh dlrow

【参考程序 1】

```
#include<cstdio>
#include<cstring>
using namespace std;
char s[10001];
int main()
{
    gets(s);
    int l,j;
    l=strlen(s);
    for(int i=0; i<l; i++)
    {
        if(s[i]==' ') printf(" ");          //保留所有空格
        else
        {
            int k=0;                        //单词长度清零
            for (j=i; s[j]!=' '; j++)
            {
                k++;                        //记录单词长度
                if (j>=l) break;           //避免越界
            }
            j--; k--;
            for (j=j; j>=i; j--)
                printf("%c",s[j]);          //将单词倒序输出
            i+=k;
        }
    }
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
```

```

#include<cstring>
using namespace std;
int main()
{
    int i,l,k=0;
    string s,t;           //l,k 分别储存 s,t 的长度
    getline(cin,s);       //读入句子
    l=s.size();
    for(i=0; i<l; i++)
        if (s[i] != ' ') t[k++] = s[i];
    else
    {
        if (k)             //倒序输出单词,即为反转单词
        {
            for (int j=k-1; j>=0; j--) printf("%c ",t[j]);
            k=0;
        }
        printf("%c ",s[i]);
    }
    if (k)
        for (int j=k-1; j>=0; j--) printf("%c ",t[j]);
    return 0;
}

```

17. 字符串 p 型编码【1.7 编程基础之字符串 31】

给定一个完全由数字字符(0,1,2,...,9)构成的字符串 str,请写出 str 的 p 型编码串。例如:字符串 122344111 可被描述为“1 个 1、2 个 2、1 个 3、2 个 4、3 个 1”,因此我们说 122344111 的 p 型编码串为 1122132431。类似的道理,编码串 101 可以用来描述 1111111111;00000000000 可描述为“11 个 0”,因此它的 p 型编码串即为 110;100200300 可描述为“1 个 1、2 个 0、1 个 2、2 个 0、1 个 3、2 个 0”,因此它的 p 型编码串为 112012201320。

输入:

仅一行,包含字符串 str。每一行字符串最多包含 1000 个数字字符。

输出:

该字符串对应的 p 型编码串。

样例输入:

122344111

样例输出:

1122132431

【参考程序 1】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    char s[1000];
    int t=1,l;
    gets(s);

```

```

l=strlen(s);
for (int i=0; i<l; ++i)
if (s[i]==s[i+1])
    t++;           //如果第 i 个数字与前一个数字相同,就累加它的连续出现次数
else
{
    cout<<t<<s[i]; //无法再往后连续时,输出第 i 个数字的连续出现次数与该数字
    t=1;           //实际上表示,开始累加第 i+1 个数字的连续出现次数
}
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    string s;
    int t=1,l;
    cin>>s;
    l=s.size();
    for (int i=0; i<l; ++i)
    if (s[i]==s[i+1])
        t++;           //如果第 i 个数字与前一个数字相同,就累加它的连续出现次数
    else
    {
        cout<<t<<s[i]; //无法再往后连续时,输出第 i 个数字的连续出现次数与该数字
        t=1;           //实际上表示,开始累加第 i+1 个数字的连续出现次数
    }
    return 0;
}

```

18. 判断字符串是否为回文【1.7 编程基础之字符串 33】

输入一个字符串,输出该字符串是否为回文。回文是指顺读和倒读都一样的字符串。

输入:

一行字符串(字符串中没有空白字符,字符串长度不超过 100)。

输出:

如果字符串是回文,输出 yes;否则输出 no。

样例输入:

abcdedcba

样例输出:

yes

【参考程序 1】

```

#include<cstring>
#include<cstdio>
#include<iostream>
using namespace std;

```

```

char s[100];
int main()
{
    gets(s);
    int i=strlen(s)-1;
    int j=0;           //从字符串的首尾同时往中间判定,i 为尾,j 为首
    while ((j<i)&&(s[j]==s[i]))
    {
        i--;
        j++;
    }
    //如果发现不是回文串则退出 while 循环,则 j 一定小于 i
    if (j>=i) cout<<" yes "<<endl; //j 大于等于 i 说明是回文串
    else cout<<" no "<<endl;       //反之则不是回文串
    return 0;
}

```

【参考程序 2】

```

#include<cstring>
#include<cstdio>
#include<iostream>
using namespace std;
string s;
int main()
{
    getline(cin,s);
    int i=s.size()-1;
    int j=0;           //从字符串的首尾同时往中间判定,i 为尾,j 为首
    while ((j<i)&&(s[j]==s[i]))
    {
        i--;
        j++;
    }
    //如果发现不是回文串则退出 while 循环,则 j 一定小于 i
    if (j>=i) cout<<" yes "<<endl; //j 大于等于 i 说明是回文串
    else cout<<" no "<<endl;       //反之则不是回文串
    return 0;
}

```

19. 最高分数的学生姓名【1.9 编程基础之顺序查找 02】

输入学生的人数,然后再输入每位学生的分数和姓名,求获得最高分数的学生的姓名。

输入:

第 1 行输入一个正整数 $n(n \leq 100)$,表示学生人数。接着输入 n 行,每行格式为:分数 姓名

- (1)分数是一个非负整数,且小于等于 100;
- (2)姓名为一个连续的字符串,中间没有空格,长度不超过 20。
- (3)数据保证最高分只有一位学生。

输出:

获得最高分数学生的姓名。

样例输入:

5

87 lilei
99 hanmeimei
97 lily
96 lucy
77 jim

样例输出:

hanmeimei

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
char s[1001],maxs[1001];
int main()
{
    int a,maxn=-1,i,n,t;
    scanf("%d",&n);
    for(i=1;i<=n;++i)
    {
        scanf("%d%s",&a,s);
        if (a>maxn)
        {
            maxn=a;                //更新最高分数
            for(int j=0;j<max(strlen(s),strlen(maxs));j++)
                maxs[j]=s[j];      //更新最高分数的姓名
        }                          //寻找最高分数的学生
    }
    cout<<maxs<<endl;
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int main()
{
    int a,maxn=-1,i,n,t;
    string s,maxs;
    scanf("%d",&n);
    for(i=1;i<=n;++i)
    {
        cin>>a>>s;
        if (a>maxn)
        {
            maxn=a;                //更新最高分数
            maxs=s;                //更新最高分数的姓名
        }
    }
}
```



```

    }
    //寻找最高分数的学生
}
cout<<maxs<<endl;
return 0;
}

```

20. 连续出现的字符【1.9 编程基础之顺序查找 11】

给定一个字符串,在字符串中找到第 1 个连续出现至少 k 次的字符。

输入:

第 1 行包含一个正整数 k($1 \leq k \leq 1000$),表示至少需要连续出现的次数。

第 2 行包含需要查找的字符串。字符串长度在 1~1000 之间,且不包含任何空白符。

输出:

若存在连续出现至少 k 次的字符,输出该字符;否则输出 No。

样例输入:

```

3
abcccaaab

```

样例输出:

```

c

```

【参考程序 1】

```

#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
int main()
{
    int i,j,k,t; char s[1000];
    scanf("%d",&k);
    scanf("%s",s);
    int l=strlen(s);
    for (i=0;i<=l;++i)
    {
        t=i;
        while (s[i]==s[t])++t; //计算字符连续出现的个数 t
        if (t-i==k) //t-i 即为字符连续出现的个数
        {
            printf("%c",s[i]);
            exit(0);
        } //如果找到连续出现 k 次的字符,就输出并退出程序,不需再往后找
    }
    printf("No\n"); //若没找到,程序未结束,就输出 No
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<string>
using namespace std;
string maxs,s;
int maxlen;

```

```
int main()
{
    while(cin>>s)                //不断输入单词
    {
        int len=s.size();
        if(s[len-1]=='.')
        {
            s[len-1]='\0';
            --len;
        }
        //若是带句点的最后一个单词,将句点删去并将单词长度减1
        if(len>maxlen)
        {
            maxlen=len;
            maxs.replace(0,1000,s); //将 s 赋给 maxs,maxs 存储最长单词
        }
        //更新最长单词
    }
    cout<<maxs;
    return 0;
}
```

21. 最长单词 2【1.13 编程基础之综合应用 16】

找出一个以“.”结尾的简单英文句子中的最长单词,输入时单词之间用空格分隔,没有缩写形式和其他特殊形式。

输入:

一个以“.”结尾的简单英文句子(长度不超过 500),单词之间用空格分隔,没有缩写形式和其他特殊形式。

输出:

该句子中最长的单词。如果多于一个,则输出第 1 个。

样例输入:

I am a student of Peking University.

样例输出:

University

【参考程序 1】

```
#include<iostream>
#include<cstring>
using namespace std;
char maxs[500],s[500];
int maxlen;
int main()
{
    while(cin>>s)                //不断输入单词
    {
        int len=strlen(s);
        if(s[len-1]=='.')
        {
            s[len-1]='\0';
            --len;
        }
        //若是带句点的最后一个单词,将句点删去并将单词长度减1
```

```

    if(len>maxlen)
    {
        maxlen=len;
        strcpy(maxs,s); //使用 strcpy 函数将 s 赋给 maxs,maxs 存储最长单词
    } //更新最长单词
}
cout<<maxs;
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<string>
using namespace std;
int i,l,len,maxlen;
string s,maxs;
int main()
{
    getline(cin,s);
    l=s.size();
    for(i=0;i<l;i++)
    {
        if (s[i]==' ') continue;
        if (s[i]=='.' ) break; //若为句点就退出
        len++; //累计长度
        if((s[i+1]=='.' ) || (s[i+1]==' '))
        { //判断是否是单词的最后一个字母
            if(len>maxlen)
            {
                maxlen=len;
                maxs=s.substr(i+1-len,len); //用 substr 函数把单词赋给 maxs
            }
            len=0;
        }
    }
    cout<<maxs;
    return 0;
}

```

第六章 函数

第一节 函数

【课堂练习】

1. 求正整数 2 和 100 之间的完全数。

完全数:因子之和等于它本身的自然数,如 $6=1+2+3$ 。

【参考程序】

```
#include<iostream>
using namespace std;
int k(int x)
{
    int j,s=0;
    for (j=1; j<=x-1; j++) //枚举 1 到 x 中,判断是否为 x 的因子
        if (x%j==0) s=s+j;    //若是 x 的因子,就由 s 累加
    return s;
}
int main()
{
    int i;
    for (i=2; i<=100; i++)
        if (k(i)==i)
            cout<<i<<endl;    //k(i)为 i 的因子之和,k(i)若与 i 相等,即满足条件输出
    return 0;
}
```

2. 编程求 2~n(n 为大于 2 的正整数)中有多少个素数。

【参考程序】

```
#include<cmath>
#include<iostream>
int sh(int i);
using namespace std;
int main()
{
    int i,n,s=0;
    cin>>n;
    for (i=2;i<=n;i++)
        if (sh(i)==true) s++;    //累加素数个数
    cout<<s<<endl;
    return 0;
}
int sh (int i)    //判断素数
```

```

{
    bool f=false; int x=2;
    while(x<=floor(sqrt(i))&&(i%x!=0))
        x++;
    if (x>floor(sqrt(i))) f=true; //若是素数返回 true
    return f;
}

```

3. 已知 $m = \frac{\max(a,b,c)}{\max(a+b,b,c) \times \max(a,b,b+c)}$, 输入 a, b, c, 求 m。把求三个数的最大数 $\max(x,y,z)$ 分别定义成函数和过程来做。

【参考程序】

```

#include<cstring>
#include<iostream>
using namespace std;
double k(int x,int y,int z)          //求三个数中的最大数
{
    if (x<y) x=y;
    if (x<z) x=z;
    return x;
}
int main()
{
    double a,b,c,m;
    cin>>a>>b>>c;
    m=k(a,b,c)/(k(a+b,b,c)*k(a,b,b+c))*1.0; //依题意算出 m
    printf("%.3lf ",m);
    return 0;
}

```

4. 如果一个自然数是素数,且它的数字位置经过对换后仍为素数,则称为绝对素数,例如 13。试求出所有二位绝对素数。

【参考程序】

```

#include<cmath>
#include<iostream>
int sh(int i);
using namespace std;
int main()
{
    int i;
    for (i=10;i<=99;i++)          //枚举所有两位整数
        if ((sh(i)==true)&&(sh((i%10)*10+(i/10))==true))
            cout<<i<<endl;
    //分别判断 i 是否为素数以及 i 变换后的数是否为素数,如果都是就输出
    return 0;
}
int sh (int i)                    //判断素数
{
    bool f=false; int x=2;

```

```
while(x<=floor(sqrt(i))&&(i%x!=0))
    x++;
if (x>floor(sqrt(i))) f=true; //若是素数返回 true
return f;
}
```

5. 自然数 a 的因子是指能将 a 整除的所有自然数,但不含 a 本身。例如 12 的因子为:1,2,3,4,6。若自然数 a 的因子之和为 b ,而且 b 的因子之和又等于 a ,则称 a, b 为一对“亲和数”。求最小的一对亲和数($a \neq b$)。

【参考程序】

```
#include<iostream>
using namespace std;
int num(int n)
{
    int x=0;
    for(int i=1;i<=n/2;i++)
        if(n%i==0) x+=i;
    return x;
} //num(n)为 n 的因子之和
int main()
{
    for(int i=1;i++) //从 1 开始枚举自然数 a
        if((num(num(i))==i)&&(i!=num(i)))
            //num(num(i))即为 i 的因子之和的因子之和
            {
                cout<<i<<' '<<num(i)<<endl; // i 与 num(i)若满足条件即为亲和数
                break; //找到最小亲和数就退出
            }
    return 0;
}
```

6. 如果一个数从左边读和从右边读都是同一个数,就称为回文数。例如 6886 就是一个回文数,求出所有的既是回文数又是素数的三位数。

【参考程序】

```
#include<iostream>
#include<cmath>
using namespace std;
int hw(int i);
int main ()
{
    int i;
    for (i=100; i<=999; i++) //枚举每一个三位数
    {
        if ((sh(i)==true)&&((i/100)==(i%10)))
            cout<<i<<endl; //如果该数是素数且百位与个位相同(即为回文数),输出该数
        }
    return 0;
}
int sh(int i) //判断素数
```

```

{
    bool f=false;
    int x=2;
    while (x<=floor(sqrt(i)) && (i%x!=0))
        x++;
    if (x>floor(sqrt(i))) f=true;
    return f;
}

```

7. 根据公式 $\arctan x(x) = x - x^3/3 + x^5/5 - x^7/7 + \dots$ 和 $\pi = 6 \arctan x(\frac{1}{\sqrt{3}})$, 定义函数 $\arctan x(x)$,

求当最后一项小于 10^{-6} 时 π 的值。

【参考程序】

```

#include<cstdio>
#include<cmath>
using namespace std;
double ax(double x)
{
    int i=1;
    double s=0.0,y,t=0.0;
    while (abs(t/i)>=1e-6) //1e-6 为 10-6, 当最后一项小于 10-6 时循环
    {
        s+=t/i; // t/i 可理解为公式一中的  $-x^3/3, x^5/5$  等单项式
        t=-1*x*x*t; // t 可理解为公式一中的  $-x^3, x^5, -x^7$  等分子
        i+=2; // i 可理解为公式一中的 3, 5, 7 等分母
    }
    return s;
}
int main()
{
    double a=ax(1/sqrt(3));
    double pi=6*a; //根据公式二计算 pi
    printf("%.10lf",pi);
    return 0;
}

```

8. 哥德巴赫猜想的命题之一是: 大于 6 的偶数等于两个素数之和。编程将 6~100 所有偶数表示成两个素数之和。

【参考程序】

```

#include<cstdio>
#include<cmath>
int n,x;
bool ss(int i) //判断素数
{
    int x=2;
    while (x<=floor(sqrt(i)) && i%x!=0) x++;
    if (x>floor(sqrt(i))) return true;
    else return false;
}

```

```
int main()
{
    for (int x=6;x<=100;x+=2)                //枚举 6~100 的所有偶数
        for (int i=2;i<=x/2;++i)            //将 x 分解为 i 和 x-i 两个数
            if (ss(i) && ss(x-i))
            {
                printf("%d=%d+%d\n",x,i,x-i); //若被分解的两个数皆为素数,即输出
                break;                        //并退出当前循环,继续枚举其他偶数
            }
}
```

【上机练习】

1. 简单算术表达式求值【1.12 编程基础之函数与过程抽象 01】

两位正整数的简单算术运算(只考虑整数运算),算术运算为:

+,加法运算;

-,减法运算;

*,乘法运算;

/,整除运算;

%,取余运算。

算术表达式的格式为(运算符前后可能有空格):

运算数 运算符 运算数

请输出相应的结果。

输入:

一行算术表达式。

输出:

整型算数运算的结果(结果值不一定为两位数,可能多于两位或少于两位)。

样例输入:

32+64

样例输出:

96

【参考程序】

```
#include<iostream>
using namespace std;
int js(int,char,int);
int main()
{
    int x,y; char z;                //x,y 为运算数,z 为运算符
    cin>>x>>z>>y;
    cout<<js(x,z,y)<<endl;
}
int js(int x,char z,int y)          //根据 z 进行运算
{
    if(z=='+') return x+y;
    if(z=='-') return x-y;
    if(z=='*') return x*y;
    if(z=='/') return x/y;
    if(z=='%') return x%y;
}
```


2. 短信计费【1.12 编程基础之函数与过程抽象 02】

用手机发短信,一条短信资费为 0.1 元,但限定一条短信的内容在 70 个字以内(包括 70 个字)。如果你一次所发送的短信超过了 70 个字,则会按照每 70 个字一条短信的限制把它分割成多条短信发送。假设已经知道你当月所发送的短信的次数与字数,试统计一下你当月短信的总资费。

输入:

第 1 行是整数 n ,表示当月发送短信的总次数,接着 n 行每行一个整数,表示每次短信的字数。

输出:

一行,当月短信总资费,单位为元,精确到小数点后 1 位。

样例输入:

10
39
49
42
61
44
147
42
72
35
46

样例输出:

1.3

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,i,hs(int n),a;
double s;
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>a;
        s=s+hs(a);           //累加每次短信的资费
    }
    printf("%.1lf",s/10.0);   //将以角为单位的数转为以元为单位的数
    return 0;
}

int hs(int n)                //计算每次短信的资费
{
    if(n%70==0)return n/70;
    else return n/70+1;
}
```

3. 甲流病人初筛【1.12 编程基础之函数与过程抽象 03】

目前正是甲流盛行时期,为了更好地进行分流治疗,医院在挂号时要求对病人的体温和

咳嗽情况进行检查,对于体温超过 37.5°C (含 37.5°C) 并且咳嗽的病人初步判定为甲流病人(初筛)。现需要统计某天前来挂号就诊的病人中有多少人被初筛为甲流病人。

输入:

第 1 行是某天前来挂号就诊的病人数 $n(n < 200)$ 。

其后有 n 行,每行是病人的信息,包括三个信息:姓名(字符串,不含空格,最多 8 个字符)、体温(float)、是否咳嗽(整数,1 表示咳嗽,0 表示不咳嗽)。每行三个信息之间以一个空格分开。

输出:

按输入顺序依次输出所有被筛选为甲流的病人的姓名,每个名字占一行。之后再输出一行,表示被筛选为甲流的病人数量。

样例输入:

```
5
Zhang 38.3 0
Li 37.5 1
Wang 37.1 1
Zhao 39.0 1
Liu 38.2 1
```

样例输出:

```
Li
Zhao
Liu
3
```

【参考程序】

```
#include<iostream>
using namespace std;
int i,n,k=0,b[201];
bool hs(float n,int a);
float a[201];
string name[201];
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>name[i]>>a[i]>>b[i];
    for(i=1;i<=n;i++)
        if(hs(a[i],b[i]))
        {
            k++;
            cout<<name[i]<<endl;
        }
    cout<<k;
    return 0;
}

bool hs(float n,int a)
{
    if((n>=37.5)&&(a==1))return true; //判断是否为甲流病人,满足条件就返回 true
    else return false;
}
```

4. 统计单词数【1.12 编程基础之函数与过程抽象 05】Noip2011 普及组第 2 题

一般的文本编辑器都有查找单词的功能,该功能可以快速定位特定单词在文章中的位置,有的还能统计出特定单词在文章中出现的次数。

现在,请你编程实现这一功能,具体要求是:给定一个单词,请你输出它在给定的文章中出现的次数和第一次出现的位置。

注意:匹配单词时,不区分大小写,但要求完全匹配,即给定单词必须与文章中的某一独立单词在不区分大小写的情况下完全相同(参见样例 1),如果给定单词仅是文章中某一单词的一部分则不算匹配(参见样例 2)。

输入:

第 1 行为一个字符串,其中只含字母,表示给定单词;

第 2 行为一个字符串,其中只可能包含字母和空格,表示给定的文章。

输出:

只有一行,如果在文章中找到给定单词则输出两个整数,两个整数之间用一个空格隔开,分别是单词在文章中出现的次数和第一次出现的位置(即在文章中第一次出现时,单词首字母在文章中的位置,位置从 0 开始);如果单词在文章中没有出现,则直接输出一个整数 -1。

样例输入:

样例 #1:

To
to be or not to be is a question

样例 #2:

to
Did the Ottoman Empire lose its power at that time

样例输出:

样例 #1:

2 0

样例 #2:

-1

【参考程序】

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<string>
using namespace std;
int main()
{
    int t=0,t2=0,l,l2;
    string s,ss;
    getline(cin,ss);
    getline(cin,s);                //使用 getline(cin,s)读入整行字符串
    l=s.size();
    l2=ss.size();
    for (int i=0;i<=l-l2;++i)//循环至 l-l2 足够,词首的下标不会大于 l-l2
    {
        int j;
        for (j=0;j<l-l2;++j)        //从要查找的单词词首和文章第 i 个字符开始逐个字符判定
```

```

{
    if (toupper(s[i+j]) != toupper(ss[j])) break;
    //若字符不匹配则退出当前循环,再从文章第 i+1 个字符开始判定
    if (i>0 && s[i-1] != ' ') break;
    //若第 i 个字符不是文章首且前一个字符不是空格,则其不是独立单词,同样退出循环
}
if (j==l2&& (s[i+j]==' ' || j+i==l))
{
    //若循环判定通过,且该单词词尾为文章尾或后有空格(独立单词),则满足条件
    ++t; //累加满足条件的单词数
    if (t==1) t2=i; //若第一次出现,用 t2 记录位置
}
}
if (t==0) printf("-1\n"); //判断是否出现,出现次数为 0 即为没有出现
else printf("%d %d\n", t, t2);
return 0;
}

```

5. 机器翻译【1.12 编程基础之函数与过程抽象 07】Noip2010 提高组第 1 题

小晨的电脑上安装了一个机器翻译软件,他经常用这个软件来翻译英语文章。

这个翻译软件的原理很简单,它只是从头到尾,依次将每个英文单词用对应的中文含义来替换。对于每个英文单词,软件会先在内存中查找这个单词的中文含义,如果内存中有,软件就会用它进行翻译;如果内存中没有,软件就会在外存中的词典内查找,查出单词的中文含义然后翻译,并将这个单词和译义放入内存,以备后续的查找和翻译。

假设内存中有 M 个单元,每个单元能存放一个单词和译义。每当软件将一个新单词存入内存前,如果当前内存中已存入的单词数不超过 $M-1$,软件会将新单词存入一个未使用的内存单元;若内存中已存入 M 个单词,软件会清空最早进入内存的那个单词,腾出单元来,存放新单词。

假设一篇英语文章的长度为 N 个单词。给定这篇待译文章,翻译软件需要去外存查找多少次词典?假设在翻译开始前,内存中没有任何单词。

输入:

输入文件共两行。每行中两个数之间用一个空格隔开。

第 1 行为两个正整数 M 和 N ,代表内存容量和文章的长度。

第 2 行为 N 个非负整数,按照文章的顺序,每个数(大小不超过 1000)代表一个英文单词。对于文章中的两个单词,当且仅当它们对应的非负整数相同时,才判定为是同一个单词。

输出:

只有 1 行,包含一个整数,为软件需要查词典的次数。

样例输入:

样例 #1:

3 7

1 2 1 5 4 4 1

样例 #2:

2 10

8 8 24 11 78 11 78 11 78 8 264

样例输出:

样例 #1:

5

样例 #2:

6

提示:

输入输出样例 1 说明:

整个查字典过程如下:每行表示一个单词的翻译,冒号前为本次翻译后的内存状况:
空:内存初始状态为空。

- (1)1:查找单词 1 并调入内存;
 - (2)1 2:查找单词 2 并调入内存;
 - (3)1 2:在内存中找到单词 1;
 - (4)1 2 5:查找单词 5 并调入内存;
 - (5)2 5 4:查找单词 4 并调入内存替代单词 1;
 - (6)2 5 4:在内存中找到单词 4;
 - (7)5 4 1:查找单词 1 并调入内存替代单词 2。
- 共计查了 5 次词典。

【参考程序】

```
#include<cstdio>
using namespace std;
int a[10001],b[10001],m,n,i,j,k=0,z=0;
int nc()
{
    for(int j=1;j<=m;j++)
        if (b[j]==a[i]) return 1;
    return 0;
}                                     //判断第 i 个单词是否在内存中
int main()
{
    scanf("%d%d",&m,&n);
    for (i=1; i<=m; ++i) b[i]=-1; //用 b 数组存放内存中的单词,初始内存中无单词
    for (i=1; i<=n; ++i) scanf("%d",&a[i]);
    for (i=1; i<=n; ++i)
    {
        if (! nc())                                     //若该单词需要查词典
        {
            ++z;                                         //累加查词典次数
            if (k>=m) k=1; //若内存已满,放入第 1 个单元(清空最早进入内存的单词)
            else ++k;                                     //若内存中未滿,那么放入第 k+1 个单元
            b[k]=a[i];                                   //放入查询后的单词
        }
    }
    printf("%d\n",z);
    return 0;
}
```

6. Vigenère 密码【1.12 编程基础之函数与过程抽象 08】Noip2012 提高组第 1 题

16 世纪法国外交家 Blaise de Vigenère 设计了一种多表密码加密算法——Vigenère 密码。Vigenère 密码的加密解密算法简单易用,且破译难度比较高,曾在美国南北战争中为南军所广泛使用。

在密码学中,我们称需要加密的信息为明文,用 M 表示;称加密后的信息为密文,用 C 表示;而密钥是一种参数,是将明文转换为密文或将密文转换为明文的算法中输入的数据,记为 k 。在 Vigenère 密码中,密钥 k 是一个字母串, $k=k_1k_2\cdots k_n$ 。当明文 $M=m_1m_2\cdots m_n$ 时,得到的密文 $C=c_1c_2\cdots c_n$,其中 $c_i=m_i\textcircled{R}k_i$,运算 $\textcircled{R}$ 的规则如下表所示:

Ⓡ	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Vigenère 加密在操作时需要注意：

- (1)Ⓡ运算忽略参与运算的字母的大小写,并保持字母在明文 M 中的大小写形式;
- (2)当明文 M 的长度大于密钥 k 的长度时,将密钥 k 重复使用。

例如,明文 M=Helloworld,密钥 k=abc 时,密文 C=Hfnlpyosnd。

明文 H e l l o w o r l d

密钥 a b c a b c a b c a

密文 H f n l p y o s n d

输入：

第 1 行为一个字符串,表示密钥 k,长度不超过 100,其中仅包含大小写字母。第 2 行为一个字符串,表示经加密后的密文,长度不超过 1000,其中仅包含大小写字母。

对于 100% 的数据,输入的密钥的长度不超过 100,输入的密文的长度不超过 1000,且都仅包含英文字母。

输出：

只有 1 行,一个字符串,表示输入密钥和密文所对应的明文。

样例输入：

CompleteVictory

Yvqgpxaimmklongnzfwpxmniytm

样例输出：

Wherethereisawillthereisaway

【参考程序】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
char s1[10001],s2[10001];
void turn(char &x)
```

```

{
    if(('z'>=x)&&('a'<=x))x-='a'-'A';
}
//将小写字母转为大写字母
int main()
{
    scanf("%s",s1);
    int len1=strlen(s1);
    for(int i=0;i<len1;++i)
        turn(s1[i]);
        //将密钥转为大写字母
    scanf("%s",s2);
    int len2=strlen(s2);
    int j=0;
        //密钥从头开始解密
    for(int i=0;i<len2;++i)
    {
        if(s2[i]<='Z')
        {
            s2[i]-=(s1[j]-'A');j++;
            if(s2[i]<'A')s2[i]='Z'-(('A'-s2[i])+1);
            //若明文小于“A”，就从字母表倒序转换
        }
        else
        {
            s2[i]-=(s1[j]-'A');j++;
            if(s2[i]<'a')s2[i]='z'-(('a'-s2[i])+1);
            //若明文小于“a”，就从字母表倒序转换
        }
        if(j>len1-1)j=0;
        // 若明文长度大于密钥长度，重复使用密钥
    }
    cout<<s2;
    return 0;
}

```

7. 素数对【1.12 编程基础之函数与过程抽象 10】

两个相差为 2 的素数称为素数对,如 5 和 7,17 和 19 等,本题目要求找出所有两个数均不大于 n 的素数对。

输入:

一个正整数 $n(1 \leq n \leq 10,000)$ 。

输出:

所有小于等于 n 的素数对。每对素数对输出一行,中间用单个空格隔开。若没有找到任何素数对,输出 empty。

样例输入:

100

样例输出:

3 5

5 7

11 13

17 19

29 31

41 43

59 61

71 73

【参考程序】

```
#include<iostream>
using namespace std;
int n,i;
bool flag=false,ss(int n);
int main()
{
    cin>>n;
    for(i=3;i<=n-2;i++)          //循环枚举较小数
    {
        if((ss(i))&&(ss(i+2)))    //判断是否为素数对
        {
            cout<<i<<' '<<i+2<<endl;
            flag=true;            //找到符合条件的素数对就将 flag 标记为 true
        }
    }
    if(! flag)cout<<" empty ";    //如果没找到就输出 empty
    return 0;
}
bool ss(int n)
{
    int i;
    for(i=2;i<n;i++)
        if(n%i==0)
        {
            return false;
        }
    return true;
}
//判断素数
```

8. 我家的门牌号【小学奥数 7649】

我家住在一条短胡同里,这条胡同的门牌号从 1 开始顺序编号。

若所有门牌号之和减去我家门牌号的两倍,恰好等于 n ,求我家的门牌号及总共有多少家。数据保证有唯一解。

输入:

一个正整数 $n(n < 100,000)$ 。

输出:

一行,包含两个正整数,分别是我家的门牌号及总共有多少家,中间用单个空格隔开。

样例输入:

100

样例输出:

10 15

【参考程序】

```
#include<cstdio>
#include<cstdlib>
```



```

using namespace std;
int n,i,j,k;
long long s=0;
void sum(int x)
{
    s=0;
    for (int i=1; i<=x; ++i) s+=i;
}
//计算所有门牌号之和
int main()
{
    scanf("%d",&n);
    for (i=2; i<=1000; ++i) //枚举总共的家数
    {
        sum(i);
        for (j=1; j<=i; ++j) //枚举我家的门牌号
            if (s-j*2==n) //判断是否满足条件
            {
                printf("%d %d\n",j,i);
                exit(0);
            }
    }
    return 0;
}

```

9. 质数的和与积【小学奥数 7827】

两个质数的和是 S, 它们的积最大是多少?

输入:

一个不大于 10000 的正整数 S, 为两个质数的和。

输出:

一个整数, 为两个质数的最大乘积。数据保证有解。

样例输入:

50

样例输出:

589

【参考程序】

```

#include<iostream>
#include<cmath> //使用 sqrt()调用 cmath 库
using namespace std;
int s,i;
bool ss(int n);
int main()
{
    cin>>s;
    for(i=s/2;i>=2;i--) //和为定值的两个数,越接近积越大,故从中间开始拆分
        if(ss(i)&&ss(s-i)) //判断两个拆分的数是否都为素数
        {
            cout<<i*(s-i);
            break;
        }
    return 0;
}

```

```

}
bool ss(int n)                //判断素数
{
    int i;
    if(n==2)return true;
    for(i=2;i<=sqrt(n);i++)
        if(n%i==0)return false;
    return true;
}

```

10. 单词替换【1.7 编程基础之字符串 21】

输入一个字符串,以回车结束(字符串长度 ≤ 100)。该字符串由若干个单词组成,单词之间用一个空格隔开,所有单词区分大小写。现需要将其中的某个单词替换成另一个单词,并输出替换之后的字符串。

输入:

第 1 行是包含多个单词的字符串 s ;
 第 2 行是待替换的单词 a (长度 ≤ 100);
 第 3 行是将 a 替换的单词 b (长度 ≤ 100)。
 s, a, b 最前面和最后面都没有空格。

输出:

输出只有 1 行,将 s 中所有单词 a 替换成 b 之后的字符串。

样例输入:

You want someone to help you
 You
 I

样例输出:

I want someone to help you

【参考程序】

```

#include<string>
#include<cstring>
#include<cstdio>
using namespace std;
int n;
char e,a[101],b[101],s[101][101];
void work(char * w)
{
    string ww=w,vv=a;
    if(ww==vv) printf("%s ",b);           //若该单词与待替换的单词相同,则替换
    else printf("%s ",w);                 //否则不替换
}
int main()
{
    while(scanf("%s%c",s[++n],&e))
        if(e!='\n') break;               //while 读入单词
    scanf("%s%s",a,b);
    for(int i=1;i<=n;++i) work(s[i]); //逐个判定单词是否需替换
    return 0;
}

```

11. 笨小猴【1.9 编程基础之顺序查找 06】Noip2008 提高组第 1 题

笨小猴的词汇量很小,所以每次做英语选择题的时候都很头疼。但是他找到了一种方法,经试验证明,用这种方法去选择选项的时候选对的概率非常大!

这种方法的具体描述如下:假设 $\max n$ 是单词中出现次数最多的字母的出现次数, $\min n$ 是单词中出现次数最少的字母的出现次数,如果 $\max n - \min n$ 是一个质数,那么笨小猴就认为这是个 Lucky Word,这样的单词很可能就是正确的答案。

输入:

只有一行,是一个单词,其中只可能出现小写字母,并且长度小于 100。

输出:

共两行,第 1 行是一个字符串,假设输入的单词是 Lucky Word,那么输出“Lucky Word”,否则输出“No Answer”;

第 2 行是一个整数,如果输入单词是 Lucky Word,输出 $\max n - \min n$ 的值,否则输出 0。

样例输入:

样例 #1:

error

样例 #2:

olympic

样例输出:

样例 #1:

Lucky Word

2

样例 #2:

No Answer

0

【参考程序】

```
#include<iostream>
#include<cmath>
#include<cstring>
using namespace std;
bool prime(int n)
{
    if(n<2) return false;
    else
    {
        for(int i=2;i<sqrt(n);i++)
            if(n%i==0) return false;
    }
    return true;
}
//判断质数
int main()
{
    int a[26],maxn=0,minn=1000;
    string s;
    int i;
    cin>>s;
    memset(a,0,sizeof(a));
    //初始化 a 数组置零
    for(i=0;i<s.length();i++)
        a[s[i]-'a']+=1;
    //统计字符串中各字母的出现次数
    for(i=0;i<26;i++)
    {
        if(maxn<a[i])
            maxn=a[i];
    }
}
```

```

        if(minn>a[i]&&.a[i]! =0)
            minn=a[i];
    }
    //查找各字母最多与最少出现次数
    if(prime(maxn-minn)==true) //满足条件即为 Lucky Word
        cout<<" Lucky Word "<<endl<<(maxn-minn)<<endl;
    else
        cout<<" No Answer "<<endl<<0<<endl;
    return 0;
}

```

12. 素数回文数的个数【1.13 编程基础之综合应用 05】

求 11~n 之间(包括 n),既是素数又是回文数的整数的个数。

输入:

一个大于 11 小于 1000 的整数 n。

输出:

11~n 之间的素数回文数个数。

样例输入:

23

样例输出:

1

提示:

回文数指左右对称的数,如:292,333。

【参考程序】

```

#include<stdio>
#include<cmath>
#include<cstring>
using namespace std;
int n,ans,min_=1,max_,anss[100001];

bool ss(int x)
{
    if(x==1)return 0;
    for(int i=2;i<=sqrt(x);++i)
        if(x%i==0)return 0;
    return 1;
}
//判断素数

bool hws(int x)
{
    char s[21];
    sprintf(s,"%d",x); //sprintf()函数将数字转为字符串
    int len=strlen(s);
    for(int i=0,j=len-1;i<=len/2;++i,--j)
        if(s[i]!=s[j])return 0;
    return 1;
}
//判断是否为回文数

int main()
{

```

```
scanf("%d",&n);
for(int i=11;i<=n;++i)
    if(ss(i)&&(hws(i)))ans++; //若满足条件,则累加个数
printf("%d",ans);
}
```

13. 判断素数个数【1.13 编程基础之综合应用 10】

输入两个整数 x 和 y , 输出两者之间的素数个数(包括 x 和 y)。

输入:

两个整数 x 和 y ($1 \leq x, y \leq 10^5$)。

输出:

输出一个整数, 表示 x, y 之间的素数个数(包括 x 和 y)。

样例输入:

1 100

样例输出:

25

【参考程序】

```
#include<iostream>
#include<cmath>
using namespace std;
int x,y,i,s=0;
bool ss(int n);
int main()
{
    cin>>x>>y;
    if(x>y)swap(x,y); //如果 x<y 则两数交换
    for(i=x;i<=y;i++) //枚举从 x~y 之间所有整数
        if(ss(i))s++; //是素数则计数器+1
    cout<<s;
    return 0;
}

bool ss(int n) //判断素数
{
    int i;
    if(n==1)return false;
    for(i=2;i<=sqrt(n);i++)
        if(n%i==0)
        {
            return false;
            break;
        }
    return true;
}
```

14. 最大质因子序列【1.13 编程基础之综合应用 21】

任意输入两个正整数 m, n ($1 < m < n \leq 5000$), 依次输出 $m \sim n$ 之间每个数的最大质因子(包括 m 和 n ; 如果某个数本身是质数, 则输出这个数自身)。

输入:

一行, 包含两个正整数 m 和 n , 其间以单个空格间隔。

输出:

一行,每个整数的最大质因子,以逗号间隔。

样例输入:

5 10

样例输出:

5,3,7,2,3,5

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cmath>
using namespace std;
int m,n,a,b;
bool zs(int x)                                //判断素数
{
    for(int i=2;i<=sqrt(x);++i)
        if(x%i==0)return 0;
    return 1;
}

int main()
{
    scanf("%d%d",&m,&n);
    for(int i=m;i<=n;++i)                    //枚举从 m~n 的所有整数
        for(int j=i;j>=2;--j)                //找每个数的最大质因子,从大到小找
            if((i%j==0)&&(zs(j)))              //找到最大质因子
            {
                printf("%d",j);
                if(i!=n)printf(",");          //输出,以逗号分隔
                break;
            }
    return 0;
}
```

15. 区间内的真素数【1.13 编程基础之综合应用 23】

找出正整数 m 和 n 之间(n 不小于 m)的所有真素数。

真素数的定义:如果一个正整数 P 为素数,且其反序也为素数,那么 P 就为真素数。

例如,11,13 均为真素数,因为 11 的反序还是为 11,13 的反序为 31 也为素数。

输入:

输入两个数 m 和 n ($1 \leq m \leq n \leq 100,000$),空格间隔。

输出:

按从小到大输出 m 和 n 之间(包括 m 和 n)的真素数,逗号间隔。如果之间没有真素数,则输出 No。

样例输入:

10 35

样例输出:

11,13,17,31

【参考程序】

```
#include<iostream>
```

```

#include<cstdio>
#include<cmath>
using namespace std;
int n,m,i,dx(int n),s=0,a[1001];           //a 数组存储所有真素数
bool flag=false,ss(int n);
int main()
{
    cin>>m>>n;
    for(i=m;i<=n;i++)                     //枚举从 m~n 的所有整数
    {
        if(ss(dx(i)))                    //这个数是素数,其逆序也是素数
        {
            s++;
            a[s]=i;
            flag=true;                   //存储到数组中
        }
    }
    if(flag)                             //含有真素数
    {
        for(i=1;i<=s;i++)
            printf("%d",a[i]);
        printf("%d ",a[s]);              //输出所有真素数
    }
    else printf(" No ");                  //不含真素数,输出“No”
    return 0;
}

bool ss(int n)                           //判断素数
{
    int i;
    if(n==1)return false;
    for(i=2;i<=sqrt(n);i++)
        if(n%i==0) return false;
    return true;
}

int dx(int n)                            //求一个数的逆序数
{
    int s=0;
    while(n>0)
    {
        s=s*10+n%10;
        n/=10;
    }
    return s;
}

```

16. 二进制分类【1.13 编程基础之综合应用 36】

若将一个正整数化为二进制数,在此二进制数中,我们将数字 1 的个数多于数字 0 的个数的这类二进制数称为 A 类数,否则就称其为 B 类数。

例如:

$(13)_{10} = (1101)_2$, 其中 1 的个数为 3, 0 的个数为 1, 则称此数为 A 类数;

$(10)_{10} = (1010)_2$, 其中 1 的个数为 2, 0 的个数也为 2, 称此数为 B 类数;

$(24)_{10} = (11000)_2$, 其中 1 的个数为 2, 0 的个数为 3, 则称此数为 B 类数;

程序要求: 求出 1~1000 之中(包括 1 与 1000), 全部 A、B 两类数的个数。

输入:

无。

输出:

一行, 包含两个整数, 分别是 A 类数和 B 类数的个数, 中间用单个空格隔开。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cmath>
using namespace std;
int A,B;
bool turn(int x)
{
    int a=0,b=0;
    while(x!=0)                //把一个数化为二进制数
    {
        if(x%2)b++;else a++;    //如果这位数为 1, 计数器 b+1, 否则计数器 a+1
        x/=2;
    }
    return b>a;                //判断数字 1 的个数是否多于数字 0 的个数
}
int main()
{
    for(int i=1;i<=1000;++i)
        if(turn(i))A++;else B++;    //1 的个数多的为 A 类数, 否则为 B 类数
    printf("%d %d ",A,B);
    return 0;
}
```

17. 确定进制【1.13 编程基础之综合应用 34】

$6 * 9 = 42$ 对于十进制来说是错误的, 但是对于 13 进制来说是正确的。即, $6_{(13)} * 9_{(13)} = 42_{(13)}$, 而 $42_{(13)} = 4 * 13^1 + 2 * 13^0 = 54_{(10)}$ 。

你的任务是写一段程序, 读入三个整数 p, q 和 r, 然后确定一个进制 B ($2 \leq B \leq 16$), 使得 $p * q = r$ 。如果 B 有很多选择, 输出最小的一个。

例如: $p=11, q=11, r=121$ 。则有 $11_{(3)} * 11_{(3)} = 121_{(3)}$ 因为 $11_{(3)} = 1 * 3^1 + 1 * 3^0 = 4_{(10)}$ 和 $121_{(3)} = 1 * 3^2 + 2 * 3^1 + 1 * 3^0 = 16_{(10)}$ 。对于进制 10, 同样有 $11_{(10)} * 11_{(10)} = 121_{(10)}$ 。这种情况下, 应该输出 3。如果没有合适的进制, 则输出 0。

输入:

一行, 包含三个整数 p, q, r。p, q, r 的所有位都是数字, 并且有 $1 \leq p, q, r \leq 1,000,000$ 。

输出:

一个整数, 即使得 $p * q = r$ 成立的最小的 B。如果没有合适的 B, 则输出 0。

样例输入:

6 9 42

样例输出:

13

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cmath>
using namespace std;
int turn(int B,int x)
{
    int value=1,x_B=0;
    while(x!=0)                //将 x 化为 B 进制数
    {
        if((x%10)>=B)return 2147483647; //在 B 进制下不得出现大于等于 B 的数位
        x_B+=(x%10)*value;
        value*=B;
        x/=10;
    }
    return x_B;
}
int main()
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    for(int i=2;i<=16;++i)      //枚举 2~16 进制
    if(turn(i,a)*turn(i,b)==turn(i,c))
    { printf("%d",i);return 0; } //找到使得 a*b=c 成立的最小的 B
    printf("0");
}
```

第二节 递归算法

【课堂练习】

1. 用递归的方法求 $1+2+3+\cdots+N$ 的值。

【参考程序】

```
#include <iostream>
using namespace std;
int nm(int n);
int x;
int main()
{
    cin>>x;
    cout<<nm(x);
    return 0;
}
int nm(int n)                                //计算  $1+2+\cdots+n$  的值
{
    if (n==0) return 0;                      //判断是否到达递归边界
    else return n+nm(n-1);                  //否则继续递归
}
```

2. 用递归函数输出斐波那契数列第 n 项。0,1,1,2,3,5,8,13...

【参考程序】

```
#include <iostream>
using namespace std;
int nm(int n);
int x;
int main()
{
    cin>>x;
    cout<<nm(x);
    return 0;
}
int nm(int n)                                //输出斐波那契数列第  $n$  项
{
    if (n==1) return 0;                     //判断是否到达递归边界: $n=1$ 
    else if (n==2) return 1;                //判断是否到达递归边界: $n=2$ 
    else return nm(n-1)+nm(n-2);            //否则继续递归
}
```

3. 输入一个非负整数,输出这个数的倒序数。例如输入 123,输出 321。

【参考程序】

```
#include<cstdio>
using namespace std;
void print(int x)                            //输出  $x$  的倒序数
{
    printf("%d ",x%10);                     //输出  $x$  的最后一位
    if(x>=10) print(x/10);                  //判断是否到达递归边界  $x<10$ ,否则继续递归
}
```

```

}
int main()
{
    int n;
    scanf("%d", &n);
    print(n);
    printf("\n");
    return 0;
}

```

4. 用递归算法将一个十进制数 X 转换成任意进制数 M ($M \leq 16$)。

【参考程序】

```

#include <cmath>
#include <iostream>
#include <cstdio>
using namespace std;
char d[16] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
int x, m;
int t(int n, int k);
int main()
{
    cin >> x >> m;
    t(x, m);
    cout << endl;
    return 0;
}
int t(int n, int k) //将十进制数 n 转换为 k 进制数
{
    int r;
    r = n % k; n = n / k; //进制转换
    if (n != 0) t(n, k); //判断是否到达递归边界: n=0, 否则继续递归
    cout << d[r];
}

```

5. 输入一串以“!”结束的字符,按逆序输出。

【参考程序】

```

#include <cstdio>
char ch[10000];
void gth(int x) //递归逆序输出字符串
{
    if (ch[x] != '!') gth(x + 1); //判断是否达到递归边界: ch[x] = "!", 否则继续递归
    else return; //若到达递归边界, 则结束递归
    printf("%c", ch[x]);
}
int main()
{
    gets(ch);
    gth(0);
    return 0;
}

```

【上机练习】

1. 阿克曼(Ackmann)函数 $A(m, n)$ 中, m, n 定义域是非负整数($m \leq 3, n \leq 10$), 函数值定义为:

$akm(m, n) = n + 1$ ($m = 0$ 时);
 $akm(m, n) = akm(m - 1, 1)$ ($m > 0, n = 0$ 时);
 $akm(m, n) = akm(m - 1, akm(m, n - 1))$ ($m, n > 0$ 时)。

【参考程序】

```
#include <iostream>
using namespace std;
int ack(int m, int n)
{
    if(m == 0)                //判断是否达到递归边界:m=0
        return(n+1);
    else if(n == 0)           //判断是否达到递归边界:n=0
        return(ack(m-1, 1));
    else
        return(ack(m-1, ack(m, n-1)));
    //否则继续递归
}
int main()
{
    int m, n;
    cin >> m >> n;
    int t = ack(m, n);        //t 储存 ack(m, n) 的值
    cout << t << endl;       //输出 t, 即 ack(m, n) 的值
    return 0;
}
```

2. 在程序中定义一函数 $digit(n, k)$, 它能分离出整数 n 从右边数第 k 个数字, 如 $digit(31859, 3) = 8, digit(2076, 5) = 0$ 。

【参考程序】

```
#include <iostream>
using namespace std;
long sh(int n, int k)
{
    k = k - 1;
    if (k == 0) return n % 10;    //判断是否达到递归边界:k=0
    else return sh(n/10, k);      //否则继续递归
}
int main()
{
    int n, k;
    cin >> n >> k;
    cout << sh(n, k);            //输出 sh(n, k) 的值
    return 0;
}
```

3. 用递归的方法求 Hermite 多项式的值

$$h_n(x) = \begin{cases} 1 & (n=0), \\ 2x & (n=1), \\ 2xh_{n-1}(x) - 2(n-1)h_{n-2}(x) & (n>1), \end{cases} \quad \text{对给定的 } x \text{ 和正整数 } n, \text{ 求多项式}$$

的值。

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
float h(int,int);
int main()
{
    int n,x;
    cin>>n>>x;
    printf("%.2f",h(n,x));    //输出 h(n,x)的值,精确到小数点后 2 位
    return 0;
}

float h(int n,int x)
{
    if (n==0)return 1;        //判断是否达到递归边界:n=0
    else if (n==1) return 2 * x; //判断是否达到递归边界:n=1
    else return (2 * x * h(n-1,x)-2 * (n-1) * h(n-2,x));
    //否则继续递归
}
```

4. 已知 $f(x,n)=\sqrt{n+\sqrt{(n-1)+\sqrt{(n-2)+\sqrt{\cdots+2+\sqrt{1+x}}}}$, 计算 $x=4.2, n=10$ 以及 $x=2.5, n=15$ 时 f 的值。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cmath>
using namespace std;
float f(float x,float n)
{
    if (n==1) return sqrt(1+x);    //判断是否达到递归边界:n=1
    else return sqrt(n+f(x,n-1));  //否则继续递归
}

int main()
{
    float a,b;
    cin>>a>>b;
    printf("%.2f",f(a,b));    //输出 f(a,b)的值,精确到小数点后 2 位
    return 0;
}
```

5. 已知

$$f(x, n) = \frac{x}{n + \frac{x}{(n-1) + \frac{x}{(n-2) + \dots + \frac{x}{1+x}}}}$$

用递归函数求解。

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
float f(float x, float n)
{
    if (n==1) return x/(1+x);           //判断是否达到递归边界:n=1
    else return x/(n+f(x, n-1));        //否则继续递归
}
int main()
{
    int a, b;
    cin>>a>>b;
    printf("%.2f ", f(a, b));           //输出 f(a, b) 的值, 精确到小数点后 2 位
    return 0;
}
```

第七章 指针及其应用

1. 利用指针,编写用于交换两个整型变量值的函数。

样例输入:5 6

样例输出:6 5

【参考程序】

```
#include <cstdio>
using namespace std;
int a, b, t;           //a,b 是输入的两个整数,t 为用于交换的变量
int *p, *q;
int main()
{
    scanf("%d%d", &a, &b);
    p=&a;               //指针 p 指向 a
    q=&b;               //指针 q 指向 b
    t=*p, *p=*q, *q=t; //交换两个指针所指向的元素
    printf("%d %d\n", a, b); //输出交换后的 a 和 b
    return 0;
}
```

2. 利用指针,编写主程序,将输入字符串反序输出。

样例输入:ABCDEFGHIJK

样例输出:KJIHGFEDCBA

【参考程序】

```
#include <cstdio>
#include <cstring>
using namespace std;
const int MaxLen=1e5+10;
int len;               //len 记录字符串的长度
char s[MaxLen], *p;    //s 表示输入的字符串,p 为指针
int main()
{
    scanf("%s", s);
    len=strlen(s);
    p=s+len-1;         //p 此时指向字符串的最后一个字符

    do
    {
        putchar(*p);   //输出 p 指向的字符
        --p;           //将 p 向前移动一位,指向前一个字符
    }while (p >= s);    //当 p 还未遍历完字符串时,继续循环

    return 0;
}
```

3. 编写一个用于在字符串中查找某字符的函数。

查找成功,函数返回该字符第一次出现的地址(指针);查找失败,返回 NULL。

编写主函数测试该函数。在主函数中输入原字符串和要查找的字符。如果找到,输出字符在原字符串中的序号;如果找不到,输出“no”。

输入:

包括两行,第 1 行为原字符串,第 2 行为要查找的字符。

输出:

一行,找到输出字符在原字符串中的序号(从 1 开始),找不到则输出“no”。

样例输入 1:

ABCDEFGHIJKLMN

D

样例输出 1:

4

样例输入 2:

ABCDEFGF

S

样例输出 2:

no

【参考程序】

```
#include <stdio>
#include <cstring>
using namespace std;
const int MaxLen=1e5+10;
char ch; //ch 表示需要查找的字符
char s[MaxLen]; //s 表示输入的字符串

char * FindChar(char * p, char ch) //这里函数的返回值是指针(地址)
{
    //这里指针 p 刚开始指向字符串的第一个字符
    while (*p != ch && *p != '\0')
        ++p; //出现需要找的字符或遍历完整整个字符串时停止遍历
    if (*p == ch)
        return p; //如果找到了这个字符,返回它的地址
    else
        return NULL; //否则就返回 NULL(空指针)
}

int main()
{
    scanf("%s\n", &s);
    ch=getchar();
    char * ans=FindChar(s, ch); //将函数返回的地址记录到 ans 里
    if (ans != NULL) //如果 ans 不为 NULL,意味着找到了这个字符
        printf("%d\n", ans-s+1); //计算出它在原字符串中的序号
    else
        puts("no"); //否则找不到就输出 no

    return 0;
}
```


4. 约瑟夫问题(使用链表)【3.2 数据结构之指针和链表 1748】

约瑟夫问题:有 n 只猴子,按顺时针方向围成一圈选大王(编号 $1 \sim n$),从第 1 号开始报数,一直数到 m ,数到 m 的猴子退出圈外,剩下的猴子再接着从 1 开始报数。就这样,直到圈内只剩下一只猴子时,这个猴子就是猴王,编程求输入 n, m 后,输出最后猴王的编号。

输入:

每行是用空格分开的两个整数,第 1 个是 n ,第 2 个是 m ($0 < m, n \leq 300$)。最后一行是:0 0。

输出格式:

对于每行输入数据(最后一行除外),输出数据也是一行,即最后猴王的编号。

样例输入:

```
6 2
12 4
8 3
0 0
```

样例输出:

```
5
1
7
```

【参考程序】

```
#include <cstdio>
using namespace std;

struct node
{
    int num; //num 记录这个结点对应猴子的编号
    node *nxt, *pre; //nxt 指向结点的后继,pre 指向结点的前驱
} *head, *tail, *p;

int n, m;

int main()
{
    while (scanf("%d%d", &n, &m), n || m) //判断 n 和 m 是否为 0 0
    {
        head = new node;
        head->num = 1;
        head->nxt = head->pre = NULL;
        tail = head; //先建立编号为 1 的结点,头指针和尾指针此时都指向它
        for (int i = 2; i <= n; ++i)
        {
            p = new node;
            p->num = i; //建立编号为 i 的结点
            tail->nxt = p;
            p->pre = tail; //将 p 指向的结点插入到 tail 指向的结点的后面
            p->nxt = NULL;
            tail = p; //将 tail 更新为 p,下次就插入在它的后面
        }
    }
}
```

```

head->pre=tail;
tail->nxt=head;           //因为是循环链表,所以头尾相连

p=head;
for (int i=1; i < n; ++i) //因为我们要输出剩下的最后一个的编号
{
    //所以只要循环(n-1)次,每次删除一个结点,即可确定答案
    for (int j=1; j < m; ++j)
    {
        p=p->nxt; //经过(m-1)次循环,p 指向要删除的那个结点

        p->nxt->pre=p->pre;
        p->pre->nxt=p->nxt; //删除结点 p

        p=p->nxt; //剩下的猴子就从刚刚删除的结点的后继开始报数
    }
    printf("%d\n", p->num); //最后剩余一个结点,则这个结点的编号即为答案
}
}

```

5. 删除数组中的元素(链表)【3.2 数据结构之指针和链表 6378】

给定 n 个整数,将这些整数中与 m 相等的删除。

假定给出的整数序列为:1,3,3,0,-3,5,6,8,3,10,22,-1,3,5,11,20,100,3,9,3

应该将其放在一个链表中,链表长度为 20。

要删除的数是 3,删除以后,链表中只剩 14 个元素:1,0,-3,5,6,8,10,22,-1,5,11,20,100,9

要求:必须使用链表,不允许使用数组,也不允许不删除元素直接输出。

程序中必须有链表的相关操作:建立链表,删除元素,输出删除后链表中元素,释放链表。

输入:

包含三行:

第 1 行是一个整数 n ($1 \leq n \leq 200,000$),代表数组中元素的个数。

第 2 行包含 n 个整数,代表数组中的 n 个元素。每个整数之间用空格分隔;每个整数的取值在 32 位有符号整数范围以内。

第 3 行是一个整数 k ,代表待删除元素的值(k 的取值也在 32 位有符号整数范围内)。

输出:

只有 1 行:

将数组内所有待删除元素删除以后,输出数组内的剩余元素的值,每个整数之间用空格分隔。

样例输入:

```

20
1 3 3 0 -3 5 6 8 3 10 22 -1 3 5 11 20 100 3 9 3
3

```

样例输出:

```

1 0 -3 5 6 8 10 22 -1 5 11 20 100 9

```

【参考程序】

```

#include <cstdio>
using namespace std;

```

```

struct node

```

```

{
    int num;                //num 记录这个结点对应元素的值
    node *nxt, *pre;        //nxt 指向结点的后继,pre 指向结点的前驱
} *head, *tail, *p;        //head 表示头指针,tail 表示尾指针

int n, k;

int main()
{
    scanf("%d", &n);
    head=new node;
    head->pre=head->nxt=NULL;
    tail=head;              //新建一个虚拟头结点(不储存元素的值)
                            //方便删除操作(不会越界),此时头尾指针都指向它

    for (int i=1; i <=n; ++i)
    {
        p=new node;
        scanf("%d", &p->num); //输入第 n 个元素的值,储存到新建的结点中
        p->pre=tail;
        tail->nxt=p;          //将 p 指向的结点插入到 tail 指向的结点的后面
        p->nxt=NULL;
        tail=p;              //将 tail 更新为 p,下次就插入在它的后面
    }

    p=new node;
    p->pre=tail;
    p->nxt=NULL;
    tail->nxt=p;
    tail=p;                  //和虚拟头结点类似,建立虚拟尾结点
                            //方便删除和查询、输出

    scanf("%d", &k);
    p=head->nxt;              //p 开始指向虚拟头结点的后继,即第 1 个元素
    while (p != tail)        //遍历到虚拟尾结点时停止
    {
        if (p->num==k)        //如果找到与 k 相同的元素,删除对应节点
        {
            p->nxt->pre=p->pre;
            p->pre->nxt=p->nxt;
        }
        p=p->nxt;
    }
    p=head->nxt;              //p 指向虚拟头结点的后继,即第 1 个元素
    while (p != tail)        //遍历到虚拟尾结点时停止
    {
        printf("%d ", p->num); //输出该结点的元素
    }
}

```

```

    p=p->nxt;
}
return 0;
}

```

6. 统计学生信息(使用动态链表完成)[3.2 数据结构之指针和链表 6379]

利用链表记录输入的学生信息(学号、姓名、性别、年龄、得分、地址)。其中,学号长度不超过 20,姓名长度不超过 40,性别长度为 1,地址长度不超过 40。

输入:

包括若干行,每一行都是一个学生的信息,如:

00630018 zhouyan m 20 10.0 28#460

输入的最后以“end”结束。

输出:

将输入的内容倒序输出。每行一条记录,按照下面的格式输出:

学号 姓名 性别 年龄 得分 地址

的格式输出

样例输入:

```

00630018 zhouyan m 20 10 28#4600
0063001 zhouyn f 21 100 28#460000
0063008 zhoyan f 20 1000 28#460000
0063018 zhouan m 21 10000 28#4600000
00613018 zhuyan m 20 100 28#4600
00160018 zouyan f 21 100 28#4600
01030018 houyan m 20 10 28#4600
0630018 zuyan m 21 100 28#4600
10630018 zouan m 20 10 28#46000
end

```

样例输出:

```

10630018 zouan m 20 10 28#46000
0630018 zuyan m 21 100 28#4600
01030018 houyan m 20 10 28#4600
00160018 zouyan f 21 100 28#4600
00613018 zhuyan m 20 100 28#4600
0063018 zhouan m 21 10000 28#4600000
0063008 zhoyan f 20 1000 28#460000
0063001 zhouyn f 21 100 28#460000
00630018 zhouyan m 20 10 28#4600

```

【参考程序】

```

#include <cstdio>
#include <cstring>
using namespace std;
struct data
{
    //建立结构体 data,方便储存信息,也方便输入、输出
    char num[25], name[45], sex, score[45], address[45];
    int age;

    bool scan()
    {

```

```

scanf("%s", num);
if (strcmp(num, "end") == 0)
    return true;           //如果输入 end,就停止输入,返回 true
scanf("%s %c %d %s %s\n", name, &sex, &age, &score, address);
return false;             //输入还没有停止,返回 false
}

void print()
{
    printf("%s %s %c %d %s %s\n", num, name, sex, age, score, address);
}
//输出信息
};

struct node
{
    data d;                //d 表示该节点对应元素的信息
    node *nxt, *pre;       //nxt 指向结点的后继,pre 指向结点的前驱
} * head, * tail, * p;    //head 表示头指针,tail 表示尾指针
int n, k;

int main()
{
    head = new node;
    head->pre = head->nxt = NULL;
    tail = head;           //新建一个虚拟头结点(不储存信息),方便操作

    while (1)
    {
        p = new node;
        if (p->d.scan())    //输入该学生的信息
            break;         //如果输入已经停止,就跳出循环

        p->pre = tail;
        tail->nxt = p;      //将 p 指向的结点插入到 tail 指向的结点的后面
        p->nxt = NULL;
        tail = p;          //将 tail 更新为 p,下次就插入在它的后面
    }

    p = tail;              //从尾结点向前遍历
    while (p != head)      //遍历到虚拟头结点时停止
    {
        p->d.print();       //输出该学生的信息
        p = p->pre;
    }
    return 0;
}

```

第八章 STL 模板库及其应用

1. 词典【3.9 数据结构之 C++ STL_1806】

你旅游到了一个国外的城市。那里的人们说的外国语言你不能理解。不过幸运的是，你有一本词典可以帮助你。

输入：

首先输入一个词典，词典中包含不超过 100,000 个词条，每个词条占据一行。每一个词条包括一个英文单词和一个外语单词，两个单词之间用一个空格隔开。而且在词典中不会有某个外语单词出现超过两次。词典之后是一个空行，然后给出一个由外语单词组成的文档，文档不超过 100,000 行，而且每行只包括一个外语单词。输入中出现单词只包括小写字母，而且长度不会超过 10。

输出：

在输出中，你需要把输入文档翻译成英文，每行输出一个英文单词。如果某个外语单词不在词典中，就把这个单词翻译成“eh”。

样例输入：

```
dog ogday
cat atcay
pig igpay
froot ootfray
loops oopslay
```

//是一个空行

```
atcay
ittenkay
oopslay
```

样例输出：

```
cat
eh
loops
```

提示：

输入比较大，推荐使用 C 语言的 I / O 函数。

【参考程序】

```
#include<iostream>
#include<cstring>
#include<cstdio>
#include<string>
#include<map>
using namespace std;
map<string,string>s;
int main()
{
    string a,b,c,d;
    while(getline(cin,a))
    {
        if(a==" ")break;
```

//每行按一个字符串读入，读到空行结束

```

c=a.substr(0,a.find(' '));           //复制空格前的子串,表示英文单词
d=a.substr(a.find(' ')+1,a.size());   //复制空格后的子串,表示外语单词
s[d]=c;                               //将词条加入词典,表示外语单词对应的英文单词
}
while(getline(cin,b))                 //读入需要翻译的外语单词
{
    if(s[b]==" ")cout<<" eh "<<endl;   //如果词典里没有这个外语单词的翻译,输出 eh
    else                               //否则输出这个字符串
    {
        for(int i=0;i<s[b].size();i++)
            printf("%c ",s[b][i]);
        cout<<endl;
    }
}
return 0;
}

```

2. List【3.9 数据结构之 C++ STL_3339】

写一个程序完成以下命令：

new id —— 新建一个指定编号为 id 的序列(id<10,000)

add id num —— 向编号为 id 的序列加入整数 num

merge id1 id2 —— 合并序列 id1 和 id2 中的数,并将 id2 清空

unique id —— 去掉序列 id 中重复的元素

out id —— 从小到大输出编号为 id 的序列中的元素,以空格隔开

输入：

第 1 行一个数 n(n≤200,000),表示有多少个命令。以后 n 行每行一个命令。

输出：

按题目要求输出。

样例输入：

```

16
new 1
new 2
add 1 1
add 1 2
add 1 3
add 2 1
add 2 2
add 2 3
add 2 4
out 1
out 2
merge 1 2
out 1
out 2
unique 1
out 1

```

样例输出

```
1 2 3
1 2 3 4
1 1 2 2 3 3 4
```

//空行

```
1 2 3 4
```

【参考程序】

```
#include<list>
#include<algorithm>
#include<iostream>
#include<cstdio>
using namespace std;
int n,Id,Num;
string s;
list<int> a[200001];
list<int>::iterator j; //链表迭代器
int main()
{
    cin>>n;
    while (n-->0)
    {
        cin>>s;
        if (s=="new ") cin>>Id;
        if (s=="add ") //加入整数 num
        {
            cin>>Id>>Num;
            a[Id].push_back(Num);
        }
        if (s=="merge ") //合并序列 id1 和 id2 中的数,并将 id2 清空
        {
            cin>>Id>>Num;
            a[Id].merge(a[Num]);
        }
        if (s=="unique ") //去掉序列 id 中重复的元素
        {
            cin>>Id;
            a[Id].sort(); //调用去重函数前应先按从小到大排序
            a[Id].unique(); //在一段连续且相同的数中,去重函数只保留一个
        }
        if (s=="out ") //从小到大输出编号为 id 的序列中的元素,以空格隔开
        {
            cin>>Id;
            if (! a[Id].empty()) //如果这个序列非空
            {
                a[Id].sort(); //从小到大排序
                for (j=a[Id].begin(); j!=a[Id].end(); ++j) //从头到尾输出
                    printf("%d ", *j);
            }
        }
    }
}
```



```

        printf("\n ");
    }
    else cout<<endl;    //否则只输出空行
}
}
}

```

3. Set【3.9 数据结构之 C++ STL_3341】

现有一整数集(允许有重复元素),初始为空。我们定义如下操作:

add x 把 x 加入集合

del x 把集合中所有与 x 相等的元素删除

ask x 对集合中元素 x 的情况询问

对每种操作,我们要求进行如下输出:

add 输出操作后集合中 x 的个数

del 输出操作前集合中 x 的个数

ask 先输出 0 或 1 表示 x 是否曾被加入集合(0 表示不曾加入),再输出当前集合中 x 的个数,中间用空格格开。

输入:

第 1 行是一个整数 $n(0 \leq n \leq 100,000)$,表示命令数。

后面 n 行命令,如 Description 中所述。

输出:

共 n 行,每行按要求输出。

样例输入:

```

7
add 1
add 1
ask 1
ask 2
del 2
del 1
ask 1

```

样例输出:

```

1
2
1 2
0 0
0
2
1 0

```

提示:

Please use STL's set and multiset to finish the task

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<map>
using namespace std;
map<int,int>a;

```

//类似于下标为整数,值为整数的数组

```
map<int,bool>b;           //类似于下标为整数,值为布尔型的数组
int main()
{
    int n,i,x;
    scanf("%d",&n);
    string s;
    for(i=1;i<=n;i++)
    {
        cin>>s>>x;
        if(s=="add")      //把 x 加入集合
        {
            a[x]++;        //a[x]表示当前值为 x 的个数
            b[x]=true;      //b[x]表示 x 是否曾经出现过
            printf("%d\n",a[x]);
        }
        else
        if(s=="del")
        {
            printf("%d\n",a[x]);
            a[x]=0;         //清除所有的 x,则 x 的个数变为 0
        }
        else
            printf("%d %d\n", b[x]/ * 0 * /,a[x]); //类型强转,有出现过输出 1,否则输出 0
    }
    return 0;
}
```

4. 字符串操作【3.9 数据结构之 C++ STL_3342】

给定 n 个字符串(从 1 开始编号),每个字符串中的字符位置从 0 开始编号,长度为 $1 \sim 500$,现有如下若干操作:

copy $N \ X \ L$:取出第 N 个字符串第 X 个字符开始的长度为 L 的字符串。

add $S1 \ S2$:判断 $S1, S2$ 是否为 $0 \sim 99,999$ 之间的整数,若是则将其转化为整数做加法,若不是,则作字符串加法,返回的值为一个字符串。

find $S \ N$:在第 N 个字符串中从左开始找寻找 S 字符串,返回其第一次出现的位置,若没有找到,返回字符串的长度。

rfind $S \ N$:在第 N 个字符串中从右开始找寻找 S 字符串,返回其第一次出现的位置,若没有找到,返回字符串的长度。

insert $S \ N \ X$:在第 N 个字符串的第 X 个字符位置中插入 S 字符串。

reset $S \ N$:将第 N 个字符串变为 S 。

print N :打印输出第 N 个字符串。

printall:打印输出所有字符串。

over:结束操作。

其中 N, X, L 可由 find 与 rfind 操作表达式构成, $S, S1, S2$ 可由 copy 与 add 操作表达式构成。

输入:

第 1 行为一个整数 n (n 在 $1 \sim 20$ 之间);

接下来 n 行为 n 个字符串,字符串不包含空格及操作命令等;

接下来若干行为一系列操作,直到 over 结束。

输出:

根据操作提示输出对应字符串。

样例输入:

```
3
329strjvc
Opadfk48
Ifjoqwoqejr
insert copy 1 find 2 1 2 2 2
print 2
reset add copy 1 find 3 1 3 copy 2 find 2 2 2 3
print 3
insert a 3 2
printall
over
```

样例输出:

```
Op29adfk48
358
329strjvc
Op29adfk48
35a8
```

提示:

推荐使用 string 类中的相关操作函数。

【参考程序 1】

```
#include<cstdio>
#include<iostream>
#include<string>
#include<cstdlib>
#define For(a,b,c) for(int a=b; a<=c; ++a)
#define For(a,b,c) for(int a=b; a<=c; ++a)
using namespace std;
string _a[201],as;
int _n;
string px(string xs);
string _add(string s1,string s2)
{
    int ls1=s1.size(),ls2=s2.size(),sum1=0,sum2=0,sum,c[11],t=0;
    string ans12=" ";
    For(j,0,ls1-1)
        if (s1[j]<'0' || s1[j]>'9') return s1+s2;
        else sum1=sum1*10+s1[j]-'0';    //把字符串转成数字
    For(j,0,ls2-1)
        if (s2[j]<'0' || s2[j]>'9') return s1+s2;
        else sum2=sum2*10+s2[j]-'0';
    if (sum1<0 || sum1>99999 || sum2<0 || sum2>99999) return s1+s2;
    sum=sum1+sum2;
    while (sum)
    {
```

```

        c[++t]=sum%10;
        sum/=10;
    }
    Fol(i,t,1)  ans12+=char(c[i]+'0'); //把数字转成字符串
    return ans12;
}

int py(string xs) //返回值为整数时调用 py,为字符串时调用 px,递归求解
{
    string s,a,b,c;
    int n,x,sum=0;
    if (xs=="find") //在第 b 个字符串的左边找字符串 a
    {
        cin>>a; s=px(a);
        cin>>b; n=py(b);
        return _a[n].find(s,0)!=string::npos? _a[n].find(s,0):_a[n].size();
    }
    if (xs=="rfind") //在第 b 个字符串的右边找字符串 a
    {
        cin>>a; s=px(a);
        cin>>b; n=py(b);
        return _a[n].rfind(s,_a[n].size()-1)!=string::npos? _a[n].rfind(s,_a[n].
size()-1):_a[n].size();
    }
    For(i,0,xs.size()-1) sum=sum*10+xs[i]-'0';
    return sum; //无包含任何函数返回原整数
}

string px(string xs)
{
    string a,b,c,s1,s2,s;
    int n,x,l;
    if (xs=="over") exit(0);
    if (xs=="copy")
    {
        cin>>a; n=py(a);
        cin>>b; x=py(b);
        cin>>c; l=py(c);
        return _a[n].substr(x,l);
    }
    if (xs=="add") //将两个字符串相加
    {
        cin>>a; s1=px(a);
        cin>>b; s2=px(b);
        return _add(s1,s2);
    }
    if (xs=="insert") //在指定字符串的指定位置插入一个字符串
    {

```

```

    cin>>a; s=px(a);
    cin>>b; n=py(b);
    cin>>c; x=py(c);
    _a[n].insert(x,s);
}
if (xs==" reset ")      //改变一个字符串
{
    cin>>a; s=px(a);
    cin>>b; n=py(b);
    _a[n]=s;
}
if (xs==" print ")      //输出一个字符串
{
    cin>>a; n=py(a);
    cout<<_a[n]<<endl;
}
if (xs==" printall ") For(i,1,_n) cout<<_a[i]<<endl;//输出全部字符串
return xs;              //无包含任何函数返回原字符串
}
int main()
{
    cin>>_n;
    For(i,1,_n) cin>>_a[i];
    while (cin>>as) px(as);
}

```

【参考程序 2】

```

#include<cstdio>
#include<string>
#include<sstream>
#include<iostream>
using namespace std;
int n;
string p,d;
string a[100001];
string get();
int ab(string s)          //字符转数字
{
    int x;
    stringstream ss(s);
    ss>>x;
    return x;
}
string ba(int x)          //数字转字符
{
    string s;
    stringstream ss;
    ss<<x;
}

```

```

    s=ss.str();
    return s;
}

string add()                                //求两数的和
{
    string s1=get();
    string s2=get();
    if (s1.size()>5 || s2.size()>5) return s1+s2;
    //判断是否越界

    int a1=ab(s1);
    int a2=ab(s2);
    string b1=ba(a1);
    string b2=ba(a2);                    //转成数字后再相加
    if (s1==b1 && s2==b2) return ba(a1+a2);
    return s1+s2;
}

string copy()                              //取出函数
{
    int c=ab(get());
    int x=ab(get());
    int y=ab(get());
    return a[c].substr(x,y);
}

string find()                              //从左往右查找
{
    string s=get();
    int x=ab(get());
    int y=a[x].find(s);
    if (y==string::npos) return ba(a[x].size());
    else return ba(y);
}

string rfind()                             //从右往左查找
{
    string s=get();
    int x=ab(get());
    int y=a[x].rfind(s);
    if (y==string::npos) return ba(a[x].size());
    else return ba(y);
}

void insert()                              //插入字符
{
    string s=get();
    int c=ab(get());
    int x=ab(get());
    a[c].insert(x,s);
}

```

```

void reset()                //替换字符
{
    string s=get();
    int x=ab(get());
    a[x]=s;
}

void print()                //输出单个字符串
{
    int x=ab(get());
    cout<<a[x]<<"\n";
}

void printall()             //输出所有字符串
{
    for (int i=1; i<=n; i++)
        cout<<a[i]<<"\n";
}

string get()                //读入字符串
{
    string s;
    cin>>s;                //判断是哪种操作
    if (s==" add ")         return add();      else
    if (s==" copy ")        return copy();     else
    if (s==" find ")        return find();     else
    if (s==" rfind ")       return rfind();    else
    if (s==" reset ")       reset();           else
    if (s==" print ")       print();           else
    if (s==" insert ")      insert();          else
    if (s==" printall ")    printall();        else
    return s;
    return " 666 ";
}

int main()
{
    scanf("%d",&n);
    for (int i=1; i<=n; i++)
        cin>>a[i];
    while (true)
    {
        p=get();            //读入字符
        if (p==" over ") break; //读到 over 结束
    }
    return 0;
}

```

5. 热血格斗场【3.9 数据结构之 C++ STL_3343】

为了迎接 2008 年的奥运会,让大家更加了解各种格斗运动,Facer 新开了一家热血格斗场。格斗场实行会员制,但是新来的会员不需要交入会费,而只要同一名老会员打一场表演赛,证明自己的实力。

我们假设格斗的实力可以用一个正整数表示,成为实力值。另外,每个人都有一个唯一的 id,也是一个正整数。为了使得比赛更好看,每一个新队员都会选择与他实力最为接近的人比赛,即比赛双方的实力值之差的绝对值越小越好,如果有两个人的实力值与他差别相同,则他会选择比他弱的那个(显然,虐人比被虐好)。

不幸的是,Facer 一不小心把比赛记录弄丢了,但是他还保留着会员的注册记录。现在请你帮 Facer 恢复比赛记录,按照时间顺序依次输出每场比赛双方的 id。

输入:

第 1 行一个数 $n(0 < n \leq 100,000)$,表示格斗场新来的会员数(不包括 Facer)。以后 n 行每 1 行两个数,按照入会的时间给出会员的 id 和实力值。一开始,Facer 就算是会员,id 为 1,实力值 1,000,000,000。输入保证两人的实力值不同。

输出:

n 行,每行两个数,为每场比赛双方的 id,新手的 id 写在前面。

样例输入:

```
3
2 1
3 3
4 2
```

样例输出:

```
2 1
3 2
4 2
```

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
using namespace std;
map<int,int>c;
map<int,int>::iterator k;          //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int n;
int main()
{
    scanf("%d",&n);
    c[1000000000]=1;              //Facer 的 id 为 1,实力值 1,000,000,000
    for(int i=1;i<=n;i++)
    {
        bool f=false;
        int a,b,id,v;
        scanf("%d%d",&id,&v);    //读入 id 和实力值
        k=c.upper_bound(v);      //找大于 v 的值中最靠前的位置
        if(k==c.end())k--,f=true; //如果没有比 v 大的值则和他的前一个人比赛
                                   //标记是最大
        a=k->first-v;              //目前实力的差值,为第一关键字
        b=k->second;              //目前对手的 id,为第二关键字
        if(k!=c.begin()&&f==false)//如果不是最小也不是最大
        {
```



```

    k--;
    if(a>(v-k->first)||f||(a==v-k->first&& k->first<v))b=k->second;
//如果前一个人和将要加入人的实力差值比目前实力的差值更小或差值相同,值更小则替换
}
printf("%d %d\n",id,b);
k=c.find(v);           //找到 v 的位置
if(k==c.end()||k->second>id)c[v]=id;
//如果将要加入的人的实力值不存在或 id 比之前的相同实力值的 id 小则加入
}
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<algorithm>
#include<cstdio>
#include<map>
using namespace std;
map<int,int>m;
map<int,int>::iterator k;           //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int n;
int main()
{
    scanf("%d",&n);
    m[1000000000]=1;                //原有成员
    while (n--)
    {
        int id,v,a,b;
        bool flag=true;
        scanf("%d%d",&id,&v);
        k=m.upper_bound(v);         //找第一次大于 v 的值的位
        if (k==m.end()) k--,flag=false;
        /* 若 k==m.end(),则说明新会员的实力值比所有老会员都高,那么记在 flag,
           直接找老会员中的实力值最大者 */
        a=k->first-v;
        //first 第一关键字,指向 k 位置的实
        b=k->second;                 //second 第二关键字,指向 k 位置的 id
        if(flag && k!=m.begin())
        {
            //若新会员的实力值不是最高的且不是最低的
            k--;                     //此时 k 指向小于 v 且最大的值的位置
            if(a>=v-k->first) b=k->second; //按题意比较选择
        }
        printf("%d %d\n",id,b);
        m[v]=id;                    //加入新成员
    }
    return 0;
}

```

【参考程序 3】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<map>
using namespace std;
map<int,int>a;
map<int,int>::iterator k;           //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int main()
{
    int n,id,v,x,y;
    scanf("%d",&n);
    a[1000000000]=1;                //关键词是实力值,数值是 id
    for (int i=1;i<=n;++i)
    {
        scanf("%d %d",&id,&v);
        k=a.upper_bound(v);         //k 表示 v 在 a 数组中的位置
        y=k->second;                 //y 表示 k 位置的数值,即 id
        if (k==a.end())              //若没有比当前会员实力值更大的会员,则当前会员定与实力
            值小于他的会员比赛
        {
            k--;
            y=k->second;
        }
        else if (k!=a.begin())
            //若当前会员的实力值既不是最大也不是最小
        {
            x=k->first-v;
            //x 表示实力值最小,并且不小于当前会员实力值,与当前会员实力值的差值
            k--;
            if (x>=v-k->first) y=k->second;
            //取实力值最大,且不大于当前会员实力值的会员,若两人的实力值与他差别相
            同,则选择实力值小的
        }
        printf("%d %d\n",id,y);
        a[v]=id;                    //加入新会员
    }
    return 0;
}
```

6. 冷血格斗场【3.9 数据结构之 C++ STL_3344】

为了迎接 2008 年的奥运会,让大家更加了解各种格斗运动,Facer 新开了一家冷血格斗场。格斗场实行会员制,但是新来的会员不需要交入会费,而只要同一名老会员打一场表演赛,证明自己的实力。

我们假设格斗的实力可以用一个正整数表示,成为实力值,两人的实力值可以相同。另外,每个人都有一个唯一的 id,也是一个正整数。为了使得比赛更好看,每一个新队员都会选择与他实力最为接近的人比赛,即比赛双方的实力值之差的绝对值越小越好,如果有多个人的实力值与他差别相同,则他会选择 id 最小的那个。

不幸的是, Facer 一不小心把比赛记录弄丢了, 但是他还保留着会员的注册记录。现在请你帮 Facer 恢复比赛记录, 按照时间顺序依次输出每场比赛双方的 id。

输入:

第 1 行一个数 $n(0 < n \leq 100,000)$, 表示格斗场新来的会员数(不包括 Facer)。以后 n 行每 1 行两个数, 按照入会的时间给出会员的 id 和实力值。一开始, Facer 就算是会员, id 为 1, 实力值 1,000,000,000。

输出:

n 行, 每行两个数, 为每场比赛双方的 id, 新手的 id 写在前面。

样例输入:

```
3
2 3
3 1
4 2
```

样例输出:

```
2 1
3 2
4 2
```

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<map>
using namespace std;
map<int,int>c;
map<int,int>::iterator k;           //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int n;
int main()
{
    scanf("%d",&n);
    c[1000000000]=1;                //Facer 的 id 为 1,实力值 1,000,000,000
    for(int i=1;i<=n;i++)
    {
        bool f=false;
        int a,b,id,v;
        scanf("%d%d",&id,&v);      //读入 id 和实力值
        k=c.upper_bound(v);         //找大于 v 的值中最靠前的位置
                                     //标记为最大
        if(k==c.end())k--,f=true;   //如果没有比 v 大的值则和他的前一个人比赛
        a=k->first-v;                //目前实力的差值,为第一关键字
        b=k->second;                 //目前对手的 id,为第二关键字
        if(k!=c.begin()&&f==false)//如果不是最小也不是最大
        {
            k--;
            if(a>(v-k->first)||f||(a==v-k->first&&b>(k->second)))b=k->second;
            //前一个人和将要加入人的实力差值比目前实力差值更小或相同,id 更小则替换
```

```

    }
    printf("%d %d\n",id,b);
    k=c.find(v);           //找到 v 的位置
    if(k==c.end()||k->second>id)c[v]=id;
    //如果将要加入的人的实力值不存在或 id 比之前的相同实力值的 id 小则加入
}
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<algorithm>
#include<cstdio>
#include<map>
using namespace std;
map<int,int>m;
map<int,int>::iterator k;           //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int n;
int main()
{
    scanf("%d",&n);
    m[1000000000]=1;               //原有成员
    while (n--)
    {
        int id,v,a,b;
        bool flag=true;
        scanf("%d%d",&id,&v);
        k=m.upper_bound(v);        //找第一次大于 v 的的位置
        if (k==m.end()) k--,flag=false;
        /* 若 k==m.end(),则说明新会员的实力值比所有老会员都高,那么记在 flag,直接找老会员中的实力值最大者 */
        a=k->first-v;
        //first 第一关键字,指向 k 位置的实力值,这里计算实力值的差值
        b=k->second;                //second 第二关键字,指向 k 位置的 id
        if(flag && k!=m.begin())
        {
            //若新会员的实力值不是最高的且不是最低的
            k--;                    //此时 k 指向小于 v 且最大的值的位置
            if(a>v-k->first || (a==v-k->first && b>k->second))
                b=k->second;        //按题意比较选择
        }
        printf("%d %d\n",id,b);
        k=m.find(v);
        if(k==m.end() || k->second>id) m[v]=id;
        /* 若没有老成员与新成员实力值相同,直接保存新成员 id;若有,保留较小的 id */
    }
    return 0;
}

```

【参考程序 3】

```

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<map>
using namespace std;
map<int,int>a;
map<int,int>::iterator k;          //声明 k 为迭代器,实质上表示数组 a 中的一个位置
int main()
{
    int n,id,v,x,y;
    scanf("%d",&n);
    a[1000000000]=1;              //关键词是实力值,数值是 id
    for (int i=1;i<=n;++i)
    {
        scanf("%d %d",&id,&v);
        k=a.upper_bound(v);      //k 表示 v 在 a 数组中的位置
        y=k->second;              //y 表示 k 位置的数值,即 id
        if (k==a.end())
        { //若没有比当前会员实力值更大的,则当前会员与实力值小于他的会员比赛
            k--;
            y=k->second;
        }
        else if (k!=a.begin())
        { //若当前会员的实力值既不是最大也不是最小
            x=k->first-v; //x 表示实力值最小,并且不小于当前会员实力值,与当前
            会员实力值的差值
            k--;
            if ((x>v-k->first) || (x==v-k->first && y>k->second)) y=k->
            second;
            //取实力值最大,且不大于当前会员实力值的会员,若两人的实力值与他差别
            相同,则选择实力值小的
        }
        printf("%d %d\n",id,y);
        k=a.find(v);
        if (k==a.end() || id<k->second) a[v]=id;
        //若没有会员与当前会员实力值相同,或当前会员的 id 比与他实力值相同的会员
        小,加入新会员
    }
    return 0;
}

```

7. priority queue 练习题【3.9 数据结构之 C++ STL_3345】

我们定义一个正整数 a 比正整数 b 优先的含义是:

- * a 的质因数数目(不包括自身)比 b 的质因数数目多;
- * 当两者质因数数目相等时,数值较大者优先级高。

现在给定一个容器,初始元素数目为 0,之后每次往里面添加 10 个元素,每次添加之后,要求输出优先级最高与最低的元素,并把该两元素从容器中删除。

输入:

第 1 行: num (添加元素次数, $\text{num} \leq 30$)。

下面 $10 * \text{num}$ 行, 每行一个正整数 $n (n < 10,000,000)$ 。

输出:

每次输入 10 个整数后, 输出容器中优先级最高与最低的元素, 两者用空格间隔。

样例输入:

```
1
10 7 66 4 5 30 91 100 8 9
```

样例输出:

```
66 5
```

【参考程序 1】

```
#include <map>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;
const int MaxSize=30 * 10+10;           //sort 解法
struct data
{
    int d;
    int tot;
}a[MaxSize];

int n;
int l, r;                                //l 表示目前数组的起始下标
                                        //r 表示目前数组的结束下标
bool cmp(const data &a, const data &b)  //按题目所给的方式排序
{
    if (a.tot != b.tot)
        return a.tot > b.tot;
    else
        return a.d > b.d;
}

int gettot(int num)                      //计算质因数个数
{
    int tmp=num;
    int ans=0;
    for (int i=2; num > 1 && i < tmp; ++i) //枚举每个可能的因数
        if (num % i == 0)
        {
            ans++;                        //能整除,计数器+1
            while (num % i == 0) num /= i; //类似分解质因数
        }
}
```

```

    return ans;
}

int main()
{
    l=1;
    scanf("%d", &n);
    for (int i=1; i <= n; ++i)
    {
        for (int j=1; j <= 10; ++j)
        {
            r++;
            scanf("%d", &a[r].d);           //读入一个数
            a[r].tot=gettot(a[r].d);         //求它的质因数并存入数组
        }
        sort(a+l, a+r+1, cmp);              //通过 sort 排序
        printf("%d %d\n", a[l].d, a[r].d);
        //得到的数组的第一个元素即是容器中优先级最高的,最后一个即是优先级最低的
        l++; r--;
    }
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<queue>           //使用优先队列需调用 queue
#include<cmath>
using namespace std;
int n;
int work(int x)           //计算质因数的个数
{
    int p=sqrt(x),s=0;
    bool flag=false;
    for(int i=2;i<=p;i++)
    if(x%i==0)
    {
        flag=true;
        break;
    }
    if(flag==false)        //没有因数直接输出 0
        return 0;
    for(int i=2;i<=x;i++)  //枚举每个可能的因数
    {
        if(x%i==0)s++;    //能整除,计数器+1
        while(x%i==0)x/=i; //分解质因数
    }
}

```

```

    return s;
}
struct big                                //优先级高的优先
{
    int z;
    friend bool operator < (big x, big y)
    {
        int a, b;
        a = work(x, z);
        b = work(y, z);
        if(a == b) return x.z > y.z; //如果 a 的质因数数目(不包括自身)比 b 多
                                     //则 a 比 b 优先
        else return a > b;           //否则,当两者质因数数目相等时,数值较大者优先级高
    }
};
struct small                              //优先级低的优先
{
    int z;
    friend bool operator < (small x, small y)
    {
        int a, b;
        a = work(x, z);
        b = work(y, z);
        if(a == b) return x.z < y.z; //如果 a 的质因数数目(不包括自身)比 b 少
                                     //则 b 比 a 优先
        else return a < b;           //否则,当两者质因数数目相等时,数值较小者优先级低
    }
};
priority_queue<big> head;                  //定义优先队列
priority_queue<small> tail;
int main()
{
    int i, j;
    scanf("%d", &n);
    for(i = 1; i <= n; i++)
    {
        for(j = 1; j <= 10; j++)
        {
            big c;
            small d;
            scanf("%d", &c.z);
            d.z = c.z;
            head.push(c);                //将新数放入两个优先队列
            tail.push(d);
        }
    }
    printf("%d %d\n", tail.top().z, head.top().z); //输出优先级最高和最低的
}

```



```

        head.pop();
        tail.pop();
    }
    return 0;
}

```

//将这两个数弹出队列

8. 排列【3.9 数据结构之 C++ STL_835】

大家知道,给出正整数 n ,则 1 到 n 这 n 个数可以构成 $n!$ 种排列,把这些排列按照从小到大的顺序(字典顺序)列出,如 $n=3$ 时,列出 1 2 3,1 3 2,2 1 3,2 3 1,3 1 2,3 2 1 六个排列。

给出某个排列,求出这个排列的下 k 个排列,如果遇到最后一个排列,则下 1 排列为第 1 个排列,即排列 1 2 3 ... n 。

比如: $n=3,k=2$ 给出排列 2 3 1,则它的下 1 个排列为 3 1 2,下 2 个排列为 3 2 1,因此答案为 3 2 1。

输入:

第 1 行是一个正整数 m ,表示测试数据的个数,下面是 m 组测试数据,每组测试数据第 1 行是 2 个正整数 n ($1 \leq n < 1024$) 和 k ($1 \leq k \leq 64$),第 2 行有 n 个正整数,是 1,2, ..., n 的一个排列。

输出:

对于每组输入数据,输出一行, n 个数,中间用空格隔开,表示输入排列的下 k 个排列。

样例输入:

```

3
3 1
2 3 1
3 1
3 2 1
10 2
1 2 3 4 5 6 7 8 9 10

```

样例输出:

```

3 1 2
1 2 3
1 2 3 4 5 6 7 9 8 10

```

【参考程序 1】

```

#include <map>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;

//使用 next_permutation 函数,需调用 algorithm

const int MaxN=1024+10;

int a[MaxN];

int main()

```

```
{
    int n, k, t;
    scanf("%d", &t);
    while (t--)
    {
        scanf("%d%d", &n, &k);
        for (int i=1; i <=n; ++i)
            scanf("%d", &a[i]);
        for (int i=1; i <=k; ++i)
            next_permutation(a+1, a+n+1);
        //用 next_permutation 函数求 k 次下一个排列

        for (int i=1; i <=n; ++i)
            printf("%d ", a[i]);    //输出的就是下 k 个排列
        putchar('\n');            //每次输出完要换行
    }
    return 0;
}
```

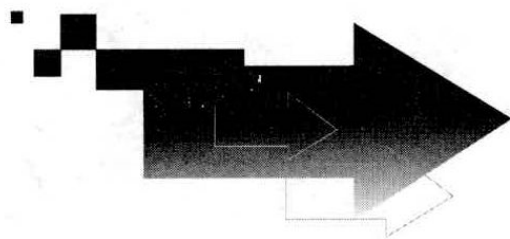
【参考程序 2】

```
#include<cstdio>
#include<iostream>
#include<algorithm>    //使用 next_permutation 函数,需调用 algorithm
#define For(a,b,c) for(int a=b; a<=c; ++a)
using namespace std;
int a[1025];
int m,n,k;
int main()
{
    scanf("%d", &m);
    while(m--)
    {
        scanf("%d%d", &n, &k);
        For(i,1,n) scanf("%d", &a[i]);
        For(i,1,k) next_permutation(a+1, a+n+1);
        //用 next_permutation 函数求下一个排列

        For(i,1,n) printf("%d ", a[i]);    //用 next_permutation 函数
        printf("\n");                        //每次输出完要换行
    }
}
```

第二部分

基础算法



第一章 高精度计算

1. 大整数加法【1.6 编程基础之一维数组 10】

求两个不超过 200 位的非负整数的和。

输入：

有两行，每行是一个不超过 200 位的非负整数，可能有多余的前导 0。

输出：

一行，即相加后的结果。结果里不能有多余的前导 0，即如果结果是 342，那么就不能输出为 0342。

样例输入：

```
22222222222222222222
33333333333333333333
```

样例输出：

```
55555555555555555555
```

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int a[2000],b[2000],c[2000];
char al[2000],bl[2000];
int main()
{
    int la,lb,lc,x=0;
    gets(al); gets(bl);           //读入字符串 al,bl
    la=strlen(al);lb=strlen(bl);   //la,lb 分别存字符串 al,bl 长度
    for (int i=0;i<=la-1;++i) a[la-i]=al[i]-48;
    for (int i=0;i<=lb-1;++i) b[lb-i]=bl[i]-48;
                                     //将两个字符分别转存到整形一维数组中
    lc=1;                           //从第一位加起
    while (lc<=la||lc<=lb)           //判断两个大整数是否都还没有算完
    {
        c[lc]=a[lc]+b[lc]+x;         //x 表示前一次的进位
        x=c[lc]/10;                 //判断是否进一
        c[lc]%10;                   //求相加后的数的个位
        lc++;                        //计算下一位数
    }
    c[lc]=x;
    while (c[lc]==0) lc--;           //去掉多余的 0
    for (int i=lc;i>=1;--i) cout<<c[i];
    cout<<endl;
    return 0;
}
```

2. 大整数减法【1.6 编程基础之综合应用 11】

求两个大的正整数相减的差。

输入:

共两行,第 1 行是被减数 a ,第 2 行是减数 $b(a > b)$ 。每个大整数不超过 200 位,不会有多余的前导 0。

输出:

一行,即所求的差。

样例输入：

99

99999999999999999999

样例输出：

999999999999999999999999999999000000000000000

【参考程序】

```
#include<stdio>
```

```
#include <cstring>
```

```
using namespace std;
```

```
char s1[1001],s2[1001];
```

```
int a[1001],b[1001];
```

```
int main()
```

{

scanf("%s",s1); scanf("%s",s2); //分别读入字符串 s1,s2

```
int len1=strlen(s1),len2=strlen(s2);    //字符长度
```

```
for(int i=len1-1;i>=0;--i) a[len1-(i+1)+1]=s1[i]+'0';
```

```
for(int i=len2-1;i>=0;--i) b[len2-(i+1)+1]=s2[i]-'0';
```

```
//将两串字符分别转存到整形一维数组
```

```
for(int i=1;i<=len1;++i)
```

{

$$a[i] - = b[i];$$

//将被减数的每一位分别减去减数的每一位

```
if(a[i]<0)a[i]+=10,a[i+1]--;    //如果不够减,就向前一位借1位
```

}

```
while(! a[len1]) len1--;
```

//去掉前面多余的0

```
for(int i=len1;i>=1;--i) printf("%d ",a[i]);
```

```
return 0;
```

}

3. 计算 2 的 N 次方【1.6 编程基础之一维数组 12】

任意给定一个正整数 $n(n \leq 100)$, 计算 2 的 n 次方的值。

输入:

一个正整数 n 。

输出:

2 的 n 次方的值。

样例输入：

5

样例输出：

32

【参考程序】

```
#include<iostream>
```

```

#include<stdio>
#include<cmath>
using namespace std;
int n,s,len=1,a[10001];
int main()
{
    scanf("%d",&n); a[1]=1;           //初始化为 1
    for(int k=1;k<=n;++k)             //进行 n 次连乘
    {
        for(int i=1;i<=len;++i) a[i]*=2; //每个位数都乘以 2
        for(int i=1;i<=len;++i)
        {
            s=0;
            if(a[i]>9)                 //判断当前的数是否需要进位
            {
                a[i+1]+=a[i]/10;       //对数进行进位
                a[i]%10;               //只留下个位
                s=max(s,i+1);
            }
        }
        len=max(len,s);               //判断当前的数的位数,避免多余地乘
    }
    for(int i=len;i>=1;--i) printf("%d",a[i]);
    return 0;
}

```

4. 大整数的因子【1.6 编程基础之一维数组 13】

已知正整数 k 满足 $2 \leq k \leq 9$, 现给出长度最大为 30 位的十进制非负整数 c , 求所有能整除 c 的 k 。

输入:

一个非负整数 c , c 的位数不大于 30。

输出:

若存在满足 $c \% k == 0$ 的 k , 从小到大输出所有这样的 k , 相邻两个数之间用单个空格隔开; 若没有这样的 k , 则输出“none”。

样例输入:

30

样例输出:

2 3 5 6

【参考程序 1】

```

#include<stdio>
#include<cstring>
using namespace std;
char c[31];
int i,k,n,flag=0,t;
int main()
{
    scanf("%s",c+1);           //c+1 方便计算,把数组 c 从 0,...,n-1 变成 1,...,n
    n=strlen(c+1);             //n 为该数的位数

```

```

for(k=2;k<=9;k++)
{
    t=0;
    for(i=1;i<=n;i++)
    {
        t=t*10+c[i]-48;    //将该位之前的余数与该位数构成新的被除数
        t%=k;              //求 c%i 的余数
    }
    if (t==0) flag=1,printf("%d ",k);
                                //如果余数等于 0,说明 c 是 i 的倍数
}
if (flag) printf("\n");
else printf(" none ");
return 0;
}

```

【参考程序 2】

```

#include <stdio>
#include <string>
const int N=31;
int l, h;
bool flag;
int c[N],int b[N];
char s[N];

bool div (int k)
{
    for (int i=0; i < l; i++)
        b[i]=c[i];
    for (int i=l-1; i >= 1; i--)
    {
        b[i-1]+=(b[i] % k) * 10;
        //将上一次计算得到的余数与原被除数的下一位构成新的被除数
        b[i] /= k;
    }
    if (b[0] % k == 0) return 1;    //判断是否满足
    return 0;
}

int main()
{
    gets (s);
    h=l=strlen (s);                //先用字符串读入
    for (int i=0; i < l; i++)      //将字符串转化成数字存于数组中
    {
        h--;
        c[i]=s[h]-'0';
    }
}

```



```
for (int k=2; k <=9; k++)
if (div (k))
{
    printf ("%d ", k);
    flag=1; //表示这个数在 2~9 范围内有因子
}
if (! flag) printf (" none\n");
return 0;
}
```

5. 求 10,000 以内 n 的阶乘【1.6 编程基础之一维数组 14】

求 10,000 以内 n 的阶乘。

输入:

只有一行输入,整数 $n(0 \leq n \leq 10,000)$ 。

输出:

一行,即 $n!$ 的值。

样例输入:

100

样例输出:

```
9332621544394415268169923885626670049071596826438162146859296389521759999
3229915608941463976156518286253697920827223758251185210916864000000000000
000000000000
```

【参考程序 1】

```
#include<iostream>
#include<cstring>
using namespace std;
int a[40010], n, x, len=1;
int main()
{
    cin>>n;
    a[1]=1; //初始化为 1
    for (int i=1; i <=n; ++i)
    {
        x=0;
        for (int j=1; j<=len; ++j) //len 表示当前整数的长度
        {
            a[j]=a[j] * i+x;
            x=a[j] / 10; //x 处理上一位的进位
            a[j]=a[j] % 10;
            if (x>0 && j>=len) len++; //判断进位后整数长度是否增加
        }
    }
    for(int i=len; i >=1; i--) cout<<a[i];
    return 0;
}
```

【参考程序 2】

```
#include <stdio> //高精乘低精
```

```

int n,t,l=1;
int a[1000001]={0,1};           //初始化为 1
int main()
{
    scanf ("%d", &n);
    for (int i=1; i <=n; i++)
    {
        for (int j=1; j <=l; j++)           //l 表示当前整数的长度
        {
            a[j] *= i;
            a[j] += a[j-1] / 10;           //处理上一个位数的进位
            a[j-1] %= 10;
            if (a[j]>9 && j>=1) l++;       //判断进位后整数长度是否增加
        }
    }
    for (int i=l; i >=1; i--) printf ("%d", a[i]);
    return 0;
}

```

6. 阶乘和【1.6 编程基础之一维数组 15】

用高精度计算出 $S=1!+2!+3!+\cdots+n!$ ($n\leq 50$), 其中“!”表示阶乘, 例如: $5!=5\times 4\times 3\times 2\times 1$ 。

输入正整数 n , 输出计算结果 S 。

输入:

一个正整数 n 。

输出:

计算结果 S 。

样例输入:

5

样例输出:

153

【参考程序】

```

#include<cstdio>
#include<iostream>
using namespace std;
int len=1,t[10001],ans[10001],anslen,n;
void jiecheng(int v)           //计算 n 的阶乘
{
    for(int i=1;i<=len;++i)t[i] *= v;   //每个位数乘 v
    int i=1;
    while((t[i]>9) || (i<len))           //判断是否需要进位
    {
        t[i+1] += t[i]/10;               //进位
        t[i] %= 10;                       //只保留个位数
        i++;                             //继续对下一位进行处理
    }
    len=i;
}

```

```

void jia()                                     //计算 1! + 2! + 3! + ... + n!
{
    for(int i=1;i<=len;++i)
    {
        ans[i] += t[i];                       //对每一位都相加
        if(ans[i]>9)                           //当满足进位条件
        {
            ans[i+1] += ans[i]/10;             //进位
            ans[i] %= 10;                      //只保留个位
            anslen = max(anslen, i+1);
        }
        anslen = max(anslen, i);               //判断当前数的位数,避免了多余的计算
    }
}

int main()
{
    scanf("%d",&n);
    t[len]=1;
    for(int i=1;i<=n;++i) jiecheng(i),jia();
    for(int i=anslen;i>=1;--i) printf("%d",ans[i]);
}

```

7. 大整数乘法【1.13 编程基础之综合应用 09】

求两个不超过 200 位的非负整数的积。

输入:

有两行,每行是一个不超过 200 位的非负整数,没有多余的前导 0。

输出:

一行,即相乘后的结果。结果里不能有多余的前导 0,即如果结果是 342,那么就不能输出为 0342。

样例输入:

```

12345678900
98765432100

```

样例输出:

```

1219326311126352690000

```

【参考程序】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
char a2[1001],b2[1001];
int a[1001],b[1001],s,ss,s3,x,c[10001];
int main()
{
    gets(a2);
    gets(b2);                               //整行读入字符串 a2,b2
    s=strlen(a2);
    ss=strlen(b2);                           //s,ss 分别为 a2,b2 的字符长度
    for (int i=0; i<=s-1; i++)

```

```

a[s-i]=a2[i]-'0';
for (int i=0; i<=ss-1; i++)
b[ss-i]=b2[i]-'0';           //存放到整形数组中方便处理
for (int i=1; i<=s; i++)
{
    x=0;
    for (int j=1; j<=ss; j++)   //将因数1的每一位分别与因数2的每一位相乘
    {
        c[i+j-1]=a[i]*b[j]+x+c[i+j-1];
                                   //将算出的结果加到积里
        x=c[i+j-1]/10;           //将多出个位的部分取出
        c[i+j-1]%=10;           //只保留个位
    }
    c[i+ss]=x;                   //进位
}
s3=s+ss;
while (c[s3]==0&& s3>1)         //删去多余的0
s3--;
for (int i=s3; i>=1; i--) cout<<c[i];
cout<<endl;
return 0;
}

```

8. 除以 13【1.13 编程基础之综合应用 27】

输入一个大于 0 的大整数 n, 长度不超过 100 位, 要求输出其除以 13 得到的商和余数。

输入:

一个大于 0 的大整数, 长度不超过 100 位。

输出:

两行, 分别为整数除法得到的商和余数。

样例输入:

2132104848488485

样例输出:

164008065268345

0

【参考程序】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int a[101], c[101];
int main()
{
    char al[101], cl[101];
    int lena, i, x=0, lenc, b;
    gets(al);
    b=13;           //被除数为 13
    lena=strlen(al);
    for(i=0; i<=lena-1; ++i)

```

```

a[i+1]=a1[i]-48;                                //转换为整形数组,方便处理
for(i=1;i<=lena;++i)
{
    c[i]=(x*10+a[i])/b;                            //将之前的余数加上新的一位再除以 13
    x=(x*10+a[i])%b;                                //求出余数
}
lenc=1;
while(c[lenc]==0&& lenc<lena)                    //删去多余的 0
lenc++;
for(i=lenc;i<=lena;++i) cout<<c[i];
cout<<endl;
cout<<x;
return 0;
}

```

第二章 数据排序

1. 谁考了第 k 名【1.10 编程基础之简单排序 01】

在一次考试中,每个学生的成绩都不相同,现知道了每个学生的学号和成绩,求考第 k 名学生的学号和成绩。

输入:

第 1 行有两个整数,分别是学生的人数 $n(1 \leq n \leq 100)$,和求第 k 名学生的 $k(1 \leq k \leq n)$ 。

其后有 n 行数据,每行包括一个学号(整数)和一个成绩(浮点数),中间用一个空格分隔。

输出:

第 k 名学生的学号和成绩,中间用空格分隔(注:请用 %g 输出成绩)。

样例输入:

```
5 3
90788001 67.8
90788002 90.3
90788003 61
90788004 68.4
90788005 73.9
```

样例输出:

```
90788004 68.4
```

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
string num[100001]; double s[100001];
int main()
{
    int i,n,a,b,c,k,j; string r; double t;
    cin>>n>>k;
    for(i=1;i<=n;++i) cin>>num[i]>>s[i];
    for(i=1;i<=n-1;++i) //从大到小,选择排序
        for(j=i+1;j<=n;++j)
            if (s[i]<s[j]) //判断前面的数是否比后面的小
            {
                swap(s[i],s[j]);
                swap(num[i],num[j]); //当条件成立,学号与成绩一起进行交换
            }
    cout<<num[k]<<' ';
    printf("%g\n",s[k]);
    return 0;
}
```

2. 奇数单增序列【1.10 编程基础之简单排序 02】

给定一个长度为 n (不大于 500) 的正整数序列, 请将其中的所有奇数取出, 并按升序输出。

输入:

第 1 行为 n ;

第 2 行为 n 个正整数, 其间用空格间隔。

输出:

增序输出的奇数序列, 数据之间以逗号间隔。数据保证至少有一个奇数。

样例输入:

10

1 3 2 6 5 4 9 8 7 10

样例输出:

1,3,5,7,9

【参考程序】

```
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<iostream>
using namespace std;
int main()
{
    int a[1001], b[1001], n, k=0, i, j, t;
    scanf("%d", &n);
    for (i=1; i<=n; ++i)
    {
        scanf("%d", &a[i]);
        if (a[i]%2 != 0)                //将奇数全部取出单独存放
        {
            ++k;
            b[k]=a[i];
        }
    }
    for (i=1; i<=k-1; ++i)              //从小到大, 选择排序
        for (j=i+1; j<=k; ++j)
            if (b[i]>b[j])
                swap(b[i], b[j]);      //当满足条件, 则进行交换
    for (i=1; i<=k-1; ++i)
        printf("%d,", b[i]);           //输出时处理逗号
    printf("%d\n", b[k]);
    return 0;
}
```

3. 成绩排序【1.10 编程基础之简单排序 03】

给出班里某门课程的成绩单, 请你按成绩从高到低对成绩单排序输出, 如果有相同分数, 则名字字典序小的在前。

输入:

第 1 行为 n ($0 < n < 20$), 表示班里的学生数目;

接下来的 n 行,每行为每个学生的名字和他的成绩,中间用单个空格隔开。名字只包含字母且长度不超过 20,成绩为一个不大于 100 的非负整数。

输出:

把成绩单按分数从高到低的顺序进行排序并输出,每行包含名字和分数两项,之间有一个空格。

样例输入:

```
4
Kitty 80
Hanmeimei 90
Joey 92
Tim 28
```

样例输出:

```
Joey 92
Hanmeimei 90
Kitty 80
Tim 28
```

【参考程序】

```
#include<iostream>
#include<cmath>
using namespace std;
int a[21],n,i,j;
string b[21];
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>b[i]>>a[i];
    for(i=1;i<n;i++)                //选择排序
        for(j=i;j<=n;j++)
            if((a[i]<a[j])||((a[i]==a[j])&&(b[i]>b[j])))
            { //前面同学的成绩是否小于后面的,相同分数则名字字典序小的在前
                swap(a[i],a[j]);
                swap(b[i],b[j]);    //满足条件,则同时交换名字和成绩
            }
    for(i=1;i<=n;i++)
        cout<<b[i]<<' '<<a[i]<<endl;
    return 0;
}
```

4. 奖学金【1.10 编程基础之简单排序 04】Noip2007 普及组第 1 题

某小学最近得到了一笔赞助,打算拿出其中一部分给学习成绩优秀的前 5 名学生发奖学金。期末,每个学生都有 3 门课的成绩:语文、数学、英语。先按总分从高到低排序,如果两个同学总分相同,再按语文成绩从高到低排序,如果两个同学总分和语文成绩都相同,那么规定学号小的同学排在前面,这样,每个学生的排序是唯一确定的。

任务:先根据输入的 3 门课的成绩计算总分,然后按上述规则排序,最后按排名顺序输出前五名学生的学号和总分。注意:在前 5 名同学中,每个人的奖学金都不相同,因此,你必须严格按上述规则排序。例如,在某个正确答案中,如果前两行的输出数据(每行输出两个数:学号、总分)是:

```
7 279
5 279
```

这两行数据的含义是:总分最高的两个同学的学号依次是 7 号、5 号。这两名同学的总分都是 279(总分等于输入的语文、数学、英语三科成绩之和),但学号为 7 的学生语文成绩更高一些。如果你的前两名的输出数据是:

5 279

7 279

则按输出错误处理,不能得分。

输入:

包含 $n+1$ 行:

第 1 行为一个正整数 n , 表示该校参加评选的学生人数。

第 2 到 $n+1$ 行, 每行有 3 个用空格隔开的数字, 每个数字都在 $0 \sim 100$ 之间。第 j 行的 3 个数字依次表示学号为 $j-1$ 的学生的语文、数学、英语的成绩。每个学生的学号按照输入顺序编号为 $1 \sim n$ (恰好是输入数据的行号减 1)。

输出:

共有 5 行, 每行是两个用空格隔开的正整数, 依次表示前 5 名学生的学号和总分。

样例输入:

样例 #1:

6

90 67 80

87 66 91

78 89 91

88 99 77

67 89 64

78 89 98

样例 #2:

8

80 89 89

88 98 78

90 67 80

87 66 91

78 89 91

88 99 77

67 89 64

78 89 98

样例输出:

样例 #1:

6 265

4 264

3 258

2 244

1 237

样例 #2:

8 265

2 264

6 264

1 258

5 258

【参考程序】

```
#include<iostream>
```

```
#include<cstdio>
```

```
using namespace std;
```

```
int a[301],b[301],c[301],y,s,i,j,n;
```

```
int main()
```

```
{
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>c[i]>>s>>y;
        a[i]=c[i]+s+y;           //求出当前同学总成绩
        b[i]=i;
    }
    for(i=1;i<n;i++)             //选择排序
    for(j=i;j<=n;j++)
    {
```

```

if(a[i]<a[j]) //判断前面同学的总分是否大于后面的
{
    swap(a[i],a[j]);
    swap(b[i],b[j]);
    swap(c[i],c[j]); //当满足交换条件,则进行交换
}
if((a[i]==a[j])&&(c[i]<c[j])) //判断,当总成绩相同时谁语文高
{
    swap(a[i],a[j]);
    swap(b[i],b[j]);
    swap(c[i],c[j]);
}
if((a[i]==a[j])&&(c[i]==c[j])&&(b[i]>b[j])) //判断,当总成绩相同且语文相同时,谁学号小
{
    swap(a[i],a[j]);
    swap(b[i],b[j]);
    swap(c[i],c[j]);
}
}
for(i=1;i<=5;i++)
cout<<b[i]<<"<<a[i]<<endl;
return 0;
}

```

5. 分数线划定【1.10 编程基础之简单排序 05】Noip2009 普及组第2题

世博会志愿者的选拔工作正在 A 市如火如荼地进行。为了选拔最合适的人才, A 市对所有报名的选手进行了笔试, 笔试分数达到面试分数线的选手方可进入面试。面试分数线根据计划录取人数的 150% 划定, 即如果计划录取 m 名志愿者, 则面试分数线为排名第 $m * 150\%$ (向下取整) 名的选手的分数, 而最终进入面试的选手为笔试成绩不低于面试分数线的所有选手。

现在就请你编写程序划定面试分数线, 并输出所有进入面试的选手的报名号和笔试成绩。

输入:

第 1 行, 两个整数 n, m ($5 \leq n \leq 5000, 3 \leq m \leq n$), 中间用一个空格隔开, 其中 n 表示报名参加笔试的选手总数, m 表示计划录取的志愿者人数。输入数据保证 $m * 150\%$ 向下取整后小于等于 n 。

第 2 行到第 $n+1$ 行, 每行包括两个整数, 中间用一个空格隔开, 分别是选手的报名号 k ($1000 \leq k \leq 9999$) 和该选手的笔试成绩 s ($1 \leq s \leq 100$)。数据保证选手的报名号各不相同。

输出:

第 1 行, 有两个整数, 用一个空格隔开, 第 1 个整数表示面试分数线; 第 2 个整数为进入面试的选手的实际人数。

从第 2 行开始, 每行包含两个整数, 中间用一个空格隔开, 分别表示进入面试的选手的报名号和笔试成绩, 按照笔试成绩从高到低输出, 如果成绩相同, 则按报名号由小到大的顺序输出。

样例输入:

```
6 3
1000 90
3239 88
2390 95
7231 84
1005 95
1001 88
```

样例输出:

```
88 5
1005 95
2390 95
1000 90
1001 88
3239 88
```

提示:

样例说明: $m * 150\% = 3 * 150\% = 4.5$, 向下取整后为 4。保证 4 个人进入面试的分数线为 88, 但因为 88 有重分, 所以所有成绩大于等于 88 的选手都可以进入面试, 故最终有 5 个人进入面试。

【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,m,i,j,a[5001],b[5001],x,k;
int main()
{
    cin>>n>>m;
    for(i=1;i<=n;i++)
        scanf("%d%d",&b[i],&a[i]);
    m=m*1.5; //求录取的人数
    for(i=1;i<=n;i++) //选择排序
        for(j=i;j<=n;j++)
            if((a[i]<a[j])||((b[i]>b[j])&&(a[i]==a[j])))
            { //判断前面的人成绩是否小于后面的,或成绩相同,判断谁的号码靠前
                swap(a[i],a[j]);
                swap(b[i],b[j]); //交换
            }
    x=a[m];k=0; //求录取的分数
    for(i=1;i<=n;i++)
        if(a[i]>=x) k++; //判断录取的人数
    cout<<x<<' '<<k<<endl;
    for(i=1;i<=k;i++)
        printf("%d %d\n",b[i],a[i]);
    return 0;
}
```

6. 整数奇偶排序【1.10 编程基础之简单排序 06】

给定 10 个整数的序列, 要求对其重新排序。排序要求:

- (1) 奇数在前, 偶数在后;
- (2) 奇数按从大到小排序;
- (3) 偶数按从小到大排序。

输入:

一行, 包含 10 个整数, 彼此以一个空格分开, 每个整数的范围是大于等于 0, 小于等于 100。

输出:

按照要求排序后输出一行, 包含排序后的 10 个整数, 数与数之间以一个空格分开。

样例输入:

4 7 3 13 11 12 0 47 34 98

样例输出:

47 13 11 7 3 0 4 12 34 98

【参考程序】

```
#include<iostream>
using namespace std;
int a[32768],b[32768];
int main()
{
    int n,i,j,k=0,m,t,f=0;
    n=10;
    for(i=1;i<=n;++i)
    {
        cin>>m;
        if(m%2==1)                //判断 m 奇偶
        {
            k++;                  //k 统计奇数个数
            b[k]=m;               //将奇数单独存放
        }
        else
        {
            f++;                  //f 统计偶数个数
            a[f]=m;               //将偶数单独存放
        }
    }
    for(i=1;i<=k;i++)             //对奇数选排,从大到小
    for(j=i+1;j<=k;j++)
    if(b[i]<b[j])
        swap(b[i],b[j]);
    for(i=1;i<=f;i++)             //对偶数选排,从小到大
    for(j=i+1;j<=f;j++)
    if(a[i]>a[j])
        swap(a[i],a[j]);
    for(i=1;i<=k;++i) cout<<b[i]<<" ";
    for(i=1;i<=f;++i) cout<<a[i]<<" "; //奇偶分别输出
    cout<<endl;
    return 0;
}
```

7. 合影效果【1.10 编程基础之简单排序 07】

小云和朋友们去爬香山,为美丽的景色所陶醉,想合影留念。如果他们站成一排,男生全部在左(从拍照者的角度),并按照从矮到高的顺序从左到右排,女生全部在右,并按照从高到矮的顺序从左到右排,请问他们合影的效果是什么样的(所有人的身高都不同)?

输入:

第 1 行是人数 $n(2 \leq n \leq 40)$,且至少有 1 个男生和 1 个女生)。

后面紧跟 n 行,每行输入一个人的性别(男 male 或女 female)和身高(浮点数,单位:米),两个数据之间以空格分隔。

输出:

n 个浮点数,模拟站好队后,拍照者眼中从左到右每个人的身高。每个浮点数需保留到小数点后 2 位,相邻两个数之间用单个空格隔开。

样例输入:

```
6
male 1.72
male 1.78
female 1.61
male 1.65
female 1.70
female 1.56
```

样例输出:

```
1.65 1.72 1.78 1.70 1.61 1.56
```

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<string>
using namespace std;
float a[32768],b[32768];
int main()
{
    int n,i,j,k=0,f=0; string s; float t,m;
    cin>>n;
    for(i=1;i<=n;++i)
    {
        cin>>s>>m;
        if(s==" male ") //判断男女
        {
            k++; b[k]=m; //男生的身高存到数组 b
        }
        else
        {
            f++; a[f]=m; //f 统计女生个数
        }
    }
    for(i=1;i<=k;i++) //对男生的身高进行选排,从小到大
    for(j=i+1;j<=k;j++)
    if(b[i]>b[j])
    {
        t=b[i]; b[i]=b[j]; b[j]=t;
    }
    for(i=1;i<=f;i++) //对女生的身高进行选排,从大到小
    for(j=i+1;j<=f;j++)
    if(a[i]<a[j])
    {
        t=a[i]; a[i]=a[j]; a[j]=t;
    }
}
```

```
for(i=1;i<=k;++i) printf("%.2f ",b[i]);
for(i=1;i<=f;++i) printf("%.2f ",a[i]);
cout<<endl;
return 0;
}
```

8. 病人排队【1.10 编程基础之简单排序 08】

病人登记看病。编写一个程序,将登记的病人按照以下原则排出看病的先后顺序:

- (1)老年人(年龄 ≥ 60 岁)比非老年人优先看病。
- (2)老年人按年龄从大到小的顺序看病,年龄相同的按登记的先后顺序排序。
- (3)非老年人按登记的先后顺序看病。

输入:

第 1 行,输入一个小于 100 的正整数,表示病人的个数;

后面按照病人登记的先后顺序,每行输入一个病人的信息,包括:一个长度小于 10 的字符串表示病人的 ID(每个病人的 ID 各不相同且只含数字和字母),一个整数表示病人的年龄,中间用单个空格隔开。

输出:

按排好的看病顺序输出病人的 ID,每行一个。

样例输入:

```
5
021075 40
004003 15
010158 67
021033 75
102012 30
```

样例输出:

```
021033
010158
021075
004003
102012
```

【参考程序】

```
#include<iostream>
#include<algorithm>
#include<cmath>
using namespace std;
int i,j,n,x=0,y=0,a[101],b[101],p[101];
string s,bs[101],cs[101];
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>s>>a[i];
        if(a[i]>=60) //将超过或刚好 60 岁的老人的岁数单独存放
        {
            x++;
            b[x]=a[i];
            bs[x]=s;
            p[x]=x;
        }
        else
        {
            y++;
```

```

        cs[y]=s;                                //将不到 60 岁的人单独存放
    }
}
for(i=1;i<x;i++)
for(j=i+1;j<=x;j++)
{
    if(b[i]<b[j])                                //判断哪个老人的岁数更大
    {
        swap(b[i],b[j]);
        swap(bs[i],bs[j]);
        swap(p[i],p[j]);                        //满足条件,则交换
    }
    if((b[i]==b[j])&&(p[i]>p[j]))                //岁数相同时则比到达的先后
    {
        swap(b[i],b[j]);
        swap(bs[i],bs[j]);
        swap(p[i],p[j]);
    }
}
for(i=1;i<=x;i++)                                //先输出老人(>=60 岁)的看病顺序
    cout<<bs[i]<<endl;
for(i=1;i<=y;i++)                                //再输出小于 60 岁的人的看病顺序
    cout<<cs[i]<<endl;
return 0;
}

```

9. 明明的随机数【1.10 编程基础之简单排序 09】Noip2006 普及组第 1 题

明明想在学校中请一些同学一起做一项问卷调查,为了实验的客观性,他先用计算机生成了 N 个 $1 \sim 1000$ 之间的随机整数($N \leq 100$),对于其中重复的数字,只保留一个,把其余相同的数去掉,不同的数对应着不同的学生的学号。然后再把这些数从小到大排序,按照排好的顺序去找同学做调查。请你协助明明完成“去重”与“排序”的工作。

输入:

有两行,第 1 行为 1 个正整数,表示所生成的随机数的个数 N ;
第 2 行有 N 个用空格隔开的正整数,为所产生的随机数。

输出:

也是两行,第 1 行为 1 个正整数 M ,表示不相同的随机数的个数。第 2 行为 M 个用空格隔开的正整数,为从小到大排好序的不相同的随机数。

样例输入:

10
20 40 32 67 40 20 89 300 400 15

样例输出:

8
15 20 32 40 67 89 300 400

【参考程序】

```

#include<iostream>
#include<cstring>
using namespace std;
int n,k,i,a[1001],x;
int main()
{
    cin>>n;

```

```

memset(a,0,sizeof(a));
for(i=1;i<=n;i++)
{
    cin>>x;
    if(a[x]==0) k++;           //统计数字的个数
    a[x]++;
}
cout<<k<<endl;
for(i=1;i<=1000;i++)
    if(a[i]>0) cout<<i<<' ';   //判断该数字是否出现过
return 0;
}

```

10. 单词排序【1. 10 编程基础之简单排序 10】

输入一行单词序列,相邻单词之间由 1 个或多个空格间隔,请按照字典序输出这些单词,要求重复的单词只输出一次。(区分大小写)

输入:

一行单词序列,最少 1 个单词,最多 100 个单词,每个单词长度不超过 50,单词之间用至少 1 个空格间隔。数据不含除字母、空格外的其他字符。

输出:

按字典序输出这些单词,重复的单词只输出一次。

样例输入:

She wants to go to Peking University to study Chinese

样例输出:

Chinese
Peking
She
University
go
study
to
wants

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
string s[100001];
int main()
{
    int i,n=0,t,j,k; string ss;
    while (cin>>ss)           //用于反复的读入
    {
        n++; s[n]=ss;
    }
    for(i=1;i<=n-1; ++i)       //对字符串的先后进行选择排序
        for(j=i+1;j<=n; ++j)
            if (s[i]>s[j]) swap(s[i],s[j]);   //判断哪个字符串更先
}

```



```

for(i=1;i<=n;++i)
    if (s[i-1] != s[i]) cout<<s[i]<<endl; //不重复地输出
return 0;
}

```

11. 出现次数超过一半的数【1.13 编程基础之综合应用 28】

给出一个含有 n ($0 < n \leq 1000$) 个整数的数组, 请找出其中出现次数超过一半的数。数组中的数大于 -50 且小于 50。

输入:

第 1 行包含一个整数 n , 表示数组大小;

第 2 行包含 n 个整数, 分别是数组中的每个元素, 相邻两个元素之间用单个空格隔开。

输出:

如果存在这样的数, 输出这个数; 否则输出 no。

样例输入:

3

1 2 2

样例输出:

2

【参考程序 1】

```

#include<cstdio>
using namespace std;
int n,b[200],x;
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
    {
        scanf("%d",&x);
        b[x+100]++;           //对 n 的数量加 1, 加 100 是因为 n 可能为负数
        if((b[x+100])>(n/2)) //判断当前的数的数量是否超过一半
        {
            printf("%d",x);
            return 0;        //输出并结束程序
        }
    }
    printf("no");           //如果程序没结束就表示没有出现次数超过一半的数
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<map>
#include<algorithm>
using namespace std;
int n,k,x;
map<int,int>b;
int main()
{
    scanf("%d",&n);

```

```

for (int i=1; i<=n; i++)
{
    scanf("%d",&x);
    b[x+100]++;           //x 的数量加 1
}
for (int i=50; i<=150; i++)
if (b[i]>n/2)             //判断出现次数是否超过一半
{
    cout<<i-100;
    exit(0);             //输出并结束程序
}
cout<<" no ";           //如果程序没结束就表示没有出现次数超过一半的数
return 0;
}

```

12. 统计字符数【1.13 编程基础之综合应用 29】

给定一个由 a~z26 个字符组成的字符串,统计其中哪个字符出现的次数最多。

输入:

一行,一个字符串,长度不超过 1000。

输出:

一行,包括出现次数最多的字符和该字符出现的次数,中间以一个空格分开。如果有多个字符出现的次数相同且最多,那么输出 ASCII 码最小的那一个字符。

样例输入:

abbccc

样例输出:

c 3

【参考程序】

```

#include<iostream>
#include<map>
#include<cstring>
using namespace std;
int n,k,x,maxx;
map<char,char>b;
char s[1001];
char ch;
int main()
{
    cin>>s;
    for(int i=0;i<=strlen(s)-1;i++) //从第一个字符搜到最后一个
    {
        b[s[i]]++;           //s[i]的数量加 1
    }
    for (int i='a'; i<='z'; i++)//模拟每个字符
    if (b[i]>maxx)             //求出现次数最多的字符
    {
        maxx=b[i];          //maxx 存储最多的次数
        ch=i;
    }
    cout<<ch<<" "<<maxx;
    return 0;
}

```

第三章 递推算法

1. 菲波那契数列(2)【2.3 基本算法之递归变递推 1760】

菲波那契数列是指这样的数列：数列的第 1 个和第 2 个数都为 1，接下来每个数都等于前面 2 个数之和。

给出一个正整数 a ，要求菲波那契数列中第 a 个数对 1000 取模的结果是多少。

输入：

第 1 行是测试数据的组数 n ，后面跟着 n 行输入。每组测试数据占 1 行，包括一个正整数 a ($1 \leq a \leq 1,000,000$)。

输出：

n 行，每行输出对应一个输入。输出应是一个正整数，为菲波那契数列中第 a 个数对 1000 取模得到的结果。

样例输入：

```
4
5
2
19
1
```

样例输出：

```
5
1
181
1
```

【参考程序】

```
#include<iostream>
#include<cmath>
using namespace std;
int a[1000001]={0,1,1},tot,n;
int main()
{
    for(int i=3;i<=1000000;i++)           //先用数组模拟递推
        a[i]=(a[i-1]+a[i-2])%1000;       //避免超出 1000 的范围
    cin>>tot;
    while(tot-->0)
    {
        cin>>n;
        cout<<a[n]<<endl;               //输出第 n 个菲波拉契数
    }
    return 0;
}
```

2. Pell 数列【2.3 基本算法之递归变递推 1788】

Pell 数列 $a_1, a_2, a_3, \dots$ 的定义是这样的, $a_1=1, a_2=2, \dots, a_n=2 * a_{n-1} + a_{n-2} (n > 2)$ 。

给出一个正整数 k ，要求 Pell 数列的第 k 项模上 32767 是多少。

输入:

第 1 行是测试数据的组数 n , 后面跟着 n 行输入。每组测试数据占 1 行, 包括一个正整数 k ($1 \leq k < 1,000,000$)。

输出:

n 行, 每行输出对应一个输入。输出应是一个非负整数。

样例输入:

2
1
8

样例输出:

1
408

【参考程序】

```
#include<iostream>
#include<cmath>
#include<algorithm>
using namespace std;
long long a[1000001],n,tot;
int main()
{
    cin>>tot;
    a[1]=1;
    a[2]=2;
    for(int i=3;i<1000000;i++)           //pell 递推
        a[i]=(2 * a[i-1]+a[i-2])%32767;    //除以 32767 是为了避免超出范围
    while(tot--)
    {
        cin>>n;
        cout<<a[n] <<endl;
    }
    return 0;
}
```

3. 上台阶【2.3 基本算法之递归变递推 3525】

楼梯有 n ($0 < n < 100$) 阶台阶, 上楼时可以一步上 1 阶, 也可以一步上 2 阶, 也可以一步上 3 阶, 编程计算共有多少种不同的走法。

输入:

输入的每一行包括一组测试数据, 即为台阶数 n 。最后一行为 0, 表示测试结束。

输出:

每一行输出对应一行输入的结果, 即为走法的数目。

样例输入:

1
2
3
4
0

样例输出：

1
2
4
7

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;
int a,f[105];
int main()
{
    f[1]=1; f[2]=2; f[3]=4;    //上第一、二、三级台阶分别有 1,2,4 种方法
    for (int i=4;i<=100;i++)    //计算出上各种级台阶的方法数
        f[i]=f[i-1]+f[i-2]+f[i-3];
    //在之前的基础上进行递推,上该台阶的方法数量就是前三种方法数量之和
    while(true)
    {
        cin>>a;
        if (a==0) break;
        else cout<<f[a]<<endl;
    }
    return 0;
}
```

4. 流感传染[2.3 基本算法之递归变递推 6262]

有一批易感人群住在网格状的宿舍区内,宿舍区为 $n \times n$ 的矩阵,每个格点为一个房间,房间里可能住人,也可能空着。在第一天,有些房间里的人得了流感,以后每天,得流感的人会使其邻居传染上流感(已经得病的不变),空房间不会传染。请输出第 m 天得流感的人数。

输入：

第一行一个数字 n , n 不超过 100,表示有 $n \times n$ 的宿舍房间。

接下来的 n 行,每行 n 个字符,“.”表示第一天该房间住着健康的人,“#”表示该房间空着,“@”表示第一天该房间住着得流感的人。

再接下来的一行是一个整数 m , m 不超过 100。

输出：

输出第 m 天,得流感的人数。

样例输入：

```
5
. . . . #
. # . @ .
. # @ . .
# . . . .
. . . . .
4
```

样例输出:

16

【参考程序】

```
#include<cstring>
#include<iostream>
using namespace std;
char a[101][101];
bool b[101][101];           //b 数组标记是否被传染
int main()
{
    int n,m,ans=0;
    memset(b,false,sizeof(b));    //初始化
    cin>>n;
    for (int i=1;i<=n;++i)
        for (int j=1;j<=n;++j) cin>>a[i][j];
    cin>>m;
    for (int k=2;k<=m;++k)    //模拟 m 天
    {
        for (int i=1;i<=n;++i)
            for (int j=1;j<=n;++j)
                if (a[i][j]=='@')    //判断当前房间是否有病人,每天传染邻居(上下左右)
                {
                    if (a[i-1][j]=='.') b[i-1][j]=true; //判断房间是否住人
                    if (a[i][j-1]=='.') b[i][j-1]=true;
                    if (a[i+1][j]=='.') b[i+1][j]=true;
                    if (a[i][j+1]=='.') b[i][j+1]=true;
                } //必须之后再更新,否则这一天刚被传染的人也会传染别人,违背了题意
        for (int i=1;i<=n;++i)
            for (int j=1;j<=n;++j)
                if (b[i][j]==true) a[i][j]='@'; //更新传染的新的病人
    }
    for (int i=1;i<=n;++i)
        for (int j=1;j<=n;++j)
            if (a[i][j]=='@') ans++;    //搜索累加得流感的人数
    cout<<ans<<endl;
    return 0;
}
```

5. 放苹果【2.3 基本算法之递归变递推 666】

把 M 个同样的苹果放在 N 个同样的盘子里,允许有的盘子空着不放,问共有多少种不同的分法?(用 K 表示)注意:5,1,1 和 1,5,1 是同一种分法。

输入:

第 1 行是测试数据的数目 t ($0 \leq t \leq 20$)。以下每行均包含两个整数 M 和 N ($1 \leq M, N \leq 10$),以空格分开。

输出:

对输入的每组数据 M 和 N ,用一行输出相应的 K 。

样例输入:

1

7 3

样例输出:

8

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int f[1001][1001];
int main()
{
    int n,m,a,b,t;
    cin>>t;
    for(n=0;n<=10;n++)
        for(m=0;m<=10;m++)    //求出所有方案
        {
            f[n][m]=1;
            if (n==0||m==1) f[n][m]=1;
            else if (n>=m) f[n][m]=f[n][m-1]+f[n-m][m];
            else f[n][m]=f[n][n];    //递推式
        }
    while (t--)
    {
        cin>>a>>b;
        cout<<f[a][b]<<endl;
    }
    return 0;
}
```

6. 吃糖果【2.6 基本算法之动态规划 1944】

名名的妈妈从外地出差回来,带了一盒好吃又精美的巧克力给名名(盒内共有 N 块巧克力, $0 < N < 20$)。妈妈告诉名名每天可以吃一块或者两块巧克力。假设名名每天都吃巧克力,问名名共有多少种不同的吃完巧克力的方案。例如:如果 $N=1$,则名名第 1 天就吃掉它,共有 1 种方案;如果 $N=2$,则名名可以第 1 天吃 1 块,第 2 天吃 1 块,也可以第 1 天吃 2 块,共有 2 种方案;如果 $N=3$,则名名第 1 天可以吃 1 块,剩 2 块,也可以第 1 天吃 2 块剩 1 块,所以名名共有 $2+1=3$ (种)方案;如果 $N=4$,则名名可以第 1 天吃 1 块,剩 3 块,也可以第 1 天吃 2 块,剩 2 块,共有 $3+2=5$ (种)方案。现在给定 N ,请你写程序求出名名吃巧克力的方案数目。

输入:

只有一行,即整数 N 。

输出:

只有一行,即名名吃巧克力的方案数。

样例输入:

4

样例输出:

5

【参考程序 1】

```
#include<cstdio>    //斐波那契数列
using namespace std;
```

```
int main()
{
    int f[1001],n;
    scanf("%d",&n);
    f[1]=1; f[2]=2;           //放糖果初始的 2 种方法
    for (int i=3; i<=n; ++i)
        f[i]=f[i-1]+f[i-2];    //菲波那契数列的递推
    printf("%d\n",f[n]);
    return 0;
}
```

7. 移动路线【2.6 基本算法之动态规划 2718】

桌子上有一个 m 行 n 列的方格矩阵,将每个方格用坐标表示,行坐标从下到上依次递增,列坐标从左到右依次递增,左下角方格的坐标为 $(1,1)$,则右上角方格的坐标为 (m,n) 。

小明是个调皮的孩子,一天他捉来一只蚂蚁,不小心把蚂蚁的右脚弄伤了,于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中,蚂蚁从左下角的方格中移动到右上角的方格中,每步移动一个方格。蚂蚁始终在方格矩阵内移动,请计算出不同的移动路线的数目。

对于 1 行 1 列的方格矩阵,蚂蚁原地移动,移动路线数为 1;对于 1 行 2 列(或 2 行 1 列)的方格矩阵,蚂蚁只需一次向右(或向上)移动,移动路线数也为 1,……对于一个 2 行 3 列的方格矩阵,如下图所示:

(2,1)	(2,2)	(2,3)
(1,1)	(1,2)	(1,3)

蚂蚁共有 3 种移动路线:

路线 1: $(1,1) \rightarrow (1,2) \rightarrow (1,3) \rightarrow (2,3)$

路线 2: $(1,1) \rightarrow (1,2) \rightarrow (2,2) \rightarrow (2,3)$

路线 3: $(1,1) \rightarrow (2,1) \rightarrow (2,2) \rightarrow (2,3)$

输入:

只有一行,包括两个整数 m 和 n ($0 < m+n \leq 20$),代表方格矩阵的行数和列数, m,n 之间用空格隔开。

输出:

只有一行,是不同的移动路线的数目。

样例输入:

2 3

样例输出:

3

【参考程序】

```
#include<iostream>           //递推
using namespace std;
int f[20][20];
int main()
{
    int n,m;
    cin>>n>>m;
    f[1][1]=1;                //如果 n=1 并且 m=1,蚂蚁只能在原地走动
    for (int i=1;i<=n;++i)
```



```

for (int j=1;j<=m;++j)
if (i!=1||j!=1) f[i][j]=f[i-1][j]+f[i][j-1];
//除了 i=1,j=1 的情况,其他所有点的路径都等于它左边与下面两个点的方案数和
cout<<f[n][m]<<endl;    //输出储存在 f[n][m] 的最大值
return 0;
}

```

8. 判断整除【2.6 基本算法之动态规划 3531】

一个给定的正整数序列,在每个数之前都插入+号或一号后计算它们的和。比如序列: 1,2,4 共有 8 种可能的序列:

```

(+1)+( +2)+( +4)=7
(+1)+( +2)+( -4)=-1
(+1)+( -2)+( +4)=3
(+1)+( -2)+( -4)=-5
(-1)+( +2)+( +4)=5
(-1)+( +2)+( -4)=-3
(-1)+( -2)+( +4)=1
(-1)+( -2)+( -4)=-7

```

所有结果中至少有一个可被整数 k 整除,我们则称此正整数序列可被 k 整除。例如上述序列可以被 3,5,7 整除,而不能被 2,4,6,8,... 整除。注意:0,-3,-6,-9,... 都可以认为是 3 的倍数。

输入:

第 1 行包含两个数: $N(2 \leq N \leq 10000)$ 和 $k(2 \leq k \leq 100)$,其中 N 代表一共有 N 个数, k 代表被除数;

第 2 行给出序列中的 N 个整数,这些整数的取值范围都在 $0 \sim 10,000$ 之间(可能重复)。

输出:

如果此正整数序列可被 k 整除,则输出 YES,否则输出 NO。(注意:都是大写字母)

样例输入:

```

3 2
1 2 4

```

样例输出:

```

NO

```

【参考程序】

```

#include<cstdio>
using namespace std;
int f[10001][1001],a[10001];
int main()
{
    int n,k;
    scanf("%d%d",&n,&k);
    for (int i=1; i<=n; i++) scanf("%d",&a[i]);
    f[1][(a[1]%k+k)%k]=1;
    for (int i=2; i<=n; i++)    //模拟各种情况
        for (int j=0; j<k; j++)
            f[i][j]=f[i-1][((j-a[i])%k+k)%k]||f[i-1][((j+a[i])%k+k)%k];
    if (f[n][0]) printf(" YES ");    //判断是否能行
    else printf(" NO ");
    return 0;
}

```

9. 踩方格【2.6 基本算法之动态规划 4982】

有一个方格矩阵,矩阵边界在无穷远处。我们做如下假设:

- (1)每走一步时,只能从当前方格移动一格,走到某个相邻的方格上;
- (2)走过的格子立即塌陷无法再走第二次;
- (3)只能向北、东、西三个方向走;

请问:如果允许在方格矩阵上走 n 步,共有多少种不同的方案?(两种走法只要有一步不一样,即被认为是不同的方案)

输入:

允许在方格上行走的步数 $n(n \leq 20)$ 。

输出:

计算出的方案数量

样例输入:

2

样例输出:

7

【参考程序】

```
#include<iostream>                //递推
using namespace std;
int f[21];
int main()
{
    int n;
    cin>>n;
    f[0]=1;                        //走 0 步时在原地,方案数为 1
    f[1]=3;                        //只能走三个方向,所以走一步有三种可能
    for (int i=2;i<=n;i++)
        f[i]=f[i-1]*2+f[i-2];
    //从第 2 步开始,往后每一步都是前一步的方案数的两倍加上前两步的方案数
    cout<<f[n]<<endl;
    return 0;
}
```

10. 山区建小学【2.6 基本算法之动态规划 7624】

政府在某山区修建了一条道路,恰好穿越总共 m 个村庄的每个村庄一次,没有回路或交叉,任意两个村庄只能通过这条路来往。已知任意两个相邻的村庄之间的距离为 d_i (为正整数),其中 $0 < i < m$ 。为了提高山区的文化素质,政府又决定从 m 个村中选择 n 个村建小学(设 $0 < n <= m < 500$)。请根据给定的 m, n 以及所有相邻村庄的距离,选择在哪些村庄建小学,才使得所有村到最近小学的距离总和最小,计算最小值。

输入:

第 1 行为 m 和 n ,其间用空格间隔;

第 2 行为 $(m-1)$ 个整数,依次表示从一端到另一端的相邻村庄的距离,整数之间以空格间隔。

例如:

10 3

2 4 6 5 2 4 3 1 3

表示在 10 个村庄建 3 所学校。第 1 个村庄与第 2 个村庄距离为 2,第 2 个村庄与第 3 个村庄距离为 4,第 3 个村庄与第 4 个村庄距离为 6,……,第 9 个村庄到第 10 个村庄的距离为 3。

输出:

各村庄到最近学校的距离之和的最小值。

样例输入:

10 2

3 1 3 1 1 1 1 3

样例输出:

18

【参考程序】

```
#include<iostream>                //递推
using namespace std;
int a[501][501]={0},c[501][501],f[501][501];
int main()
{
    int i,n,t,j,k,l,m,mid,s=0x7fffffff;
    cin>>m>>n;
    for(i=1;i<m;i++) cin>>a[i][i+1];
    for(i=1;i<=m;i++)
    for(j=i+1;j<=m;j++)
    {
        a[i][j]=a[j-1][j]+a[i][j-1];
        a[j][i]=a[i][j];           //a[i][j]表示 i,j 两地之间的距离
    }
    for(i=1;i<=m;i++)
    for(j=i+1;j<=m;j++)
    {
        mid=(i+j)/2; c[i][j]=0; //在两地中间建小学时,距离最短
        for(k=i;k<=j;k++)
            c[i][j]+=a[k][mid];    //c[i][j]表示两地到最近学校的距离
    }
    for(i=1;i<=m;i++)
        f[i][1]=c[1][i];          //赋初值
    for(i=1;i<=m;i++)
    for(j=2;j<=n;j++)
    {
        f[i][j]=s;                 //先将 f[i][j]值赋成最大值
        for (k=j-1;k<=i;k++)
            f[i][j]=min(f[i][j],f[k][j-1]+c[k+1][i]);
        //找出最小值
    }
    cout<<f[m][n]<<endl;
    return 0;
}
```

第四章 递归算法

1. 逆波兰表达式【2.2 基本算法之递归 1696】

逆波兰表达式是一种把运算符前置的算术表达式,例如普通的表达式 $2+3$ 的逆波兰表示法为 $+2\ 3$ 。逆波兰表达式的优点是运算符之间不必有优先级关系,也不必用括号改变运算次序,例如 $(2+3)*4$ 的逆波兰表示法为 $*+2\ 3\ 4$ 。本题求解逆波兰表达式的值,其中运算符包括“+”“-”“*”“/”四个。

输入:

一行,其中运算符和运算数之间都用空格分隔,运算数是浮点数。

输出:

一行,表达式的值。可直接用 `printf("%f\n", v)` 输出表达式的值 `v`。

样例输入:

`*+11.0 12.0+24.0 35.0`

样例输出:

`1357.000000`

【参考程序】

```
#include<stdio>
#include<algorithm>
#include<cstring>
using namespace std;
char str[101];
double hxs;
double exp()
{
    scanf("%s",str);
    switch (str[0])           //边递归读入边计算,
    {                          //case 语句分情况处理
        case '+':hxs=exp()+exp();break;
        case '-':hxs=exp()-exp();break;
        case '*':hxs=exp()*exp();break;
        case '/':hxs=exp()/exp();break;
        default : hxs=atof(str);
                                //使用 atof(str)把字符串 str 转换为一个 double 的浮点数
    }
    return hxs;
}
int main()
{
    printf("%f\n",exp());
}
```

2. 全排列【2.2 基本算法之递归 1750】

给定一个由不同的小写字母组成的字符串,输出这个字符串的所有全排列。

我们假设对于小写字母有 $a < b < \dots < y < z$,而且给定的字符串中的字母已经按照从

小到大的顺序排列。

输入:

只有一行,是一个由不同的小写字母组成的字符串,已知字符串的长度在 1~6 之间。

输出:

输出这个字符串的所有排列方式,每行一个排列。要求字母序比较小的排列在前面。字母序如下定义:

已知 $S=s_1s_2\cdots s_k$, $T=t_1t_2\cdots t_k$, 则 $S<T$ 等价于: 存在 $p(1\leq p\leq k)$, 使得 $s_1=t_1$, $s_2=t_2, \cdots, s_{(p-1)}=t_{(p-1)}$, $s_p<t_p$ 成立。

样例输入:

abc

样例输出:

abc

acb

bac

bca

cab

cba

【参考程序 1】

```
#include<cstdio>
#include<cstring>
using namespace std;

bool b[1001];           //b 表示字母是否被选过
char s[1001],ans[1001];
int len;
void dfs(int i)         //递归回溯
{
    for(int j=0;j<len;++j)
        if(! b[s[j]]) //判断是否重复
        {
            b[s[j]]=1;
            ans[i]=s[j];
            if(i==len-1)
                printf("%s\n",ans); //达到长度后输出方案
            else dfs(i+1);           //搜索下一长度的方案
            b[s[j]]=0;               //回溯法,退回上一长度重新搜索
        }
}

int main()
{
    scanf("%s",s);
    len=strlen(s); //len 存储字符串长度
    dfs(0);         //从长度 0 开始搜索方案
}
```

【参考程序 2】

```
#include<iostream>
```

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
char a[1001];int b[1001],c[1001]={0},tot;
void hxs(int n)
{
    for(int i=0;i<tot;i++)
    {
        if(c[i]==0)                //判断是否选择过
        {
            b[n]=i;
            c[i]=1;                //选择过存为真
            hxs(n+1);              //搜索下一长度的方案
            c[i]=0;                //回溯法,退回上一长度重新搜索
        }
    }
    if(n==tot-1)                   //当达到字符串长度,输出
    {
        for(int i=0;i<tot;i++)    //输出全排列
            cout<<a[b[i]];
        cout<<endl;
    }
}
int main()
{
    cin>>a;
    tot=strlen(a);                //tot 存储字符串长度
    hxs(0);                       //从长度 0 开始搜索方案
}
```

3. 分解因数【2.2 基本算法之递归 1751】

给出一个正整数 a , 要求分解成若干个正整数的乘积, 即 $a=a_1 * a_2 * a_3 * \dots * a_n$, 并且 $1 < a_1 \leq a_2 \leq a_3 \leq \dots \leq a_n$, 问: 这样的分解的种数有多少? 注意: $a=a$ 也是一种分解。

输入:

第 1 行是测试数据的组数 n , 后面跟着 n 行输入。每组测试数据占 1 行, 包括一个正整数 a ($1 < a < 32768$)。

输出:

n 行, 每行输出对应一个输入。输出应是一个正整数, 指明满足要求的分解的种数。

样例输入:

2
2
20

样例输出:

1
4

【参考程序 1】

```
#include<iostream>
```

```

using namespace std;
int j,k,n;
void ac(int s,int m)
{
    if(s==1)
    {
        k++; return;
    }
    //拆分数为 1 时,方案数+1,结束递归
    else for(int i=m;i<=s;++i)
    if(s%i==0)
    {
        ac(s/i,i);
        //s 除以正整数 i 后继续拆分
        //正整数从 i 开始继续枚举
    }
}
int main()
{
    cin>>n;
    for(j=1;j<=n;++j)
    {
        k=0; int t;
        cin>>t;
        ac(t,2);
        //把将要拆分的数 t 传递给 s
        //因子为 1 时会无限递归,所以从 2 开始
        cout<<k<<endl;
        //输出方案数
    }
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
using namespace std;
int n,j;
int f(int a,int b)
{
    if (a==1) return 1;
    if (b==1) return 0;
    if (a%b==0)
        return f(a/b,b)+f(a,b-1);
    //拆分数为 1 时,方案数+1,结束递归
    //因子为 1 时,避免无限循环,跳出
    //判断 b 是否为 a 的因子
    // (1) 拆分 a,因子从 b 开始枚举取一种方案
    // (2) 以 b-1 继续判断取一种方案
    // 否则以 b-1 继续判断
    return f(a,b-1);
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
    {
        scanf("%d",&j);
    }
}

```

```

        cout<<f(j,j)<<endl;
    }
}

```

4. 菲波那契数列【2.2 基本算法之递归 1755】

菲波那契数列是指这样的数列：数列的第 1 个和第 2 个数都为 1，接下来每个数都等于前面两个数之和。

给出一个正整数 a ，要求菲波那契数列中第 a 个数是多少。

输入：

第 1 行是测试数据的组数 n ，后面跟着 n 行输入。每组测试数据占 1 行，包括一个正整数 a ($1 \leq a \leq 20$)。

输出：

输出有 n 行，每行输出对应一个输入。输出应是一个正整数，为菲波那契数列中第 a 个数的大小。

样例输入：

```

4
5
2
19
1

```

样例输出：

```

5
1
4181
1

```

【参考程序】

```

#include<iostream>
#include<cstdio>
using namespace std;
int fib(int n)
{
    if(n==1||n==2) return 1; //n=1 或 2 时已递归到最前两位，所以均返回 1，并结束
    else return fib(n-1)+fib(n-2); //菲波那契数列：一个数等于前两个数相加之和
}
int main()
{
    int i,n,m,tot;
    cin>>n;
    for(i=0;i<n;i++) //有 n 组数据
    {
        cin>>m;
        tot=fib(m); //调用菲波那契数列递归函数，求出大小
        cout<<tot<<endl;;
    }
}

```

5. Pell 数列【2.2 基本算法之递归 1788】

Pell 数列 $a_1, a_2, a_3, \dots$ 的定义是这样的， $a_1=1, a_2=2, \dots, a_n=2 * a_{n-1} + a_{n-2} (n \geq 2)$ 。

给出一个正整数 k , 要求 Pell 数列的第 k 项模上 32767 是多少。

输入:

第 1 行是测试数据的组数 n , 后面跟着 n 行输入。每组测试数据占 1 行, 包括一个正整数 k ($1 \leq k < 1000000$)。

输出:

n 行, 每行输出对应一个输入。输出应是一个非负整数。

样例输入:

2
1
8

样例输出:

1
408

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
int ans[1000001];           //ans 存储答案
int work(int x)
{
    if(ans[x] != 0) return ans[x];           //已求过的答案可直接使用,不必再继续递归求解
    if(x==1) return ans[1]=1;
    if(x==2) return ans[2]=2;               //特判 x=1 或 2 的情况
    return ans[x]=(2 * work(x-1)+work(x-2))%32767;
                                           //递归求解:  $a_n = 2 * a_{n-1} + a_{n-2}$ 
                                           //每处理一次取模,避免数字过大
}
int main()
{
    for(int i=1;i<=1000000;++i) work(i);
                                           //求出所有的解,便于输入后直接输出

    int n;
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
    {
        int a;
        scanf("%d",&a);
        printf("%d\n",ans[a]);
    }
}
```

6. 扩号匹配问题【2.2 基本算法之递归 2705】

在某个字符串(长度不超过 100)中有左括号、右括号和大小写字母;规定(与常见的算数式子一样)任何一个左括号都从内到外与在它右边且距离最近的右括号匹配。写一个程序,找到无法匹配的左括号和右括号,输出原来字符串,并在下一行标出不能匹配的括号。不能匹配的左括号用“\$”标注,不能匹配的右括号用“?”标注。

输入:

包括多组数据,每组数据一行,包含一个字符串,只包含左右括号和大小写字母,字符串

长度不超过 100。

注意:cin.getline(str,100)最多只能输入 99 个字符!

输出:

对每组输出数据,输出两行,第 1 行包含原始输入字符,第 2 行由“\$”,“?”和空格组成,“\$”和“?”表示与之对应的左括号和右括号不能匹配。

样例输入:

```
((ABCD(x)
)(rttyy())sss)(
```

样例输出:

```
((ABCD(x)
$$
)(rttyy())sss)(
?          ? $
```

【参考程序 1】 //一般做法

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
using namespace std;
char a[101],aa[101];
int up(int d)
{
    if (d== -1) return -1;           //没有匹配的括号时,结束递归
    else if (aa[d]=='$') return d;    //找到匹配的括号时,标记返回
    else return up(d-1);             //向 d 之前的长度搜索
}
int main()
{
    while (scanf("%s",a) != EOF)
    {
        printf("%s\n",a);
        int n=strlen(a);
        memset(aa,' ',sizeof(aa));   //字符串每个元素默认为空格
        for (int i=0;i<n;i++)
        {
            if (a[i]=='(') aa[i]='$'; //找到左括号时,先默认为“$”
            else if (a[i]==')')
            {
                int m=up(i-1);        //寻找上一个左括号
                if (m== -1) aa[i]='? '; //没找到时默认为“?”
                else aa[m]=')';        //否则覆盖此右括号
            }
        }
        printf("%s\n",aa);
    }
    return 0;
}
```

【参考程序 2】 //栈做法

```

#include<iostream>
#include<cstring>
#include<cstdio>
#include<string>
using namespace std;
int l,i,top,a[10001],b[10001];
string s;
int main()
{
    while(getline(cin,s))
    {
        top=0;
        l=s.size();           //l 存储字符串长度
        memset(a,0,sizeof(a));
        memset(b,0,sizeof(b)); //每次处理前清空数组 a,b,防止重复
        for(i=0;i<l;i++)
        {
            if(s[i]=='(')
            {
                top++;
                a[top]=i;           //遇左括号时入栈
                b[i]=1;           //默认为存在左括号(1)
            }
            if(s[i]==')')
            {
                b[i]=2;           //先默认为存在右括号(2),若出栈则无实际作用
                if(top>0)
                {
                    b[a[top]]=0;   //覆盖上一个左括号(0)
                    b[i]=0;       //覆盖此右括号(0)
                    top--;         //覆盖后出栈
                }
            }
        }
        for(i=0;i<l;i++)
            printf("%c ",s[i]);
        printf("\n ");
        for(i=0;i<l;i++)
        {
            if(b[i]==0)printf(" ");
            if(b[i]==1)printf("$ ");
            if(b[i]==2)printf("? ");
        }
        printf("\n "); //判断输出
    }
    return 0;
}

```

}

7. 爬楼梯【2.2 基本算法之递归 3089】

树老师爬楼梯,他可以每次走 1 级或者 2 级,输入楼梯的级数,求不同的走法数。

例如:楼梯一共有 3 级,他可以每次都走一级,或者第一次走一级,第二次走两级,也可以第一次走两级,第二次走一级,一共 3 种方法。

输入:

包含若干行,每行包含一个正整数 $n(1 \leq n \leq 30)$,代表楼梯级数。

输出:

不同的走法数,每一行输入对应一行输出。

样例输入:

5
8
10

样例输出:

8
34
89

【参考程序】

```
#include<iostream>
using namespace std;
int dg(int);
int main()
{
    int n;
    while (cin>>n)                //循环读入
        cout<<dg(n)<<endl;        //递归后直接返回输出
    return 0;
}
int dg(int n)
{
    if (n==1) return 1;
    else if (n==2) return 2;        //n=1 或 2 时特判并直接返回
    else return dg(n-1)+dg(n-2);    //爬楼梯规律,为上一次和上上次的走法数的和
}
```

8. 汉诺塔问题【2.2 基本算法之递归 6261】

约 19 世纪末,在欧洲的商店中出售一种智力玩具,在一块铜板上有三根杆,最左边的杆上自上而下、由小到大顺序串着由 64 个圆盘构成的塔。目的是将最左边杆上的盘全部移到中间的杆上,条件是一次只能移动一个盘,且不允许大盘放在小盘的上面。

这是一个著名的问题,几乎所有的教材上都有这个问题。由于条件是一次只能移动一个盘,且不允许大盘放在小盘上面,所以 64 个盘的移动次数是:18,446,744,073,709,551,615

这是一个天文数字,若每一微秒可能计算(并不输出)一次移动,那么也需要几乎一百万年。我们仅能找出问题的解决方法并解决较小 n 值时的汉诺塔,但很难用计算机解决 64 层的汉诺塔。

假定圆盘从小到大编号为 1,2,...

输入:

一个整数后面跟三个单字符字符串。

整数为盘子的数目,后三个字符表示三个杆子的编号。

输出:

输出每一步移动盘子的记录。一次移动一行。

每次移动的记录为例如 $a \rightarrow 3 \rightarrow b$ 的形式,即把编号为 3 的盘子从 a 杆移至 b 杆。

样例输入:

2 a b c

样例输出:

$a \rightarrow 1 \rightarrow c$

$a \rightarrow 2 \rightarrow b$

$c \rightarrow 1 \rightarrow b$

【参考程序】

```
#include<iostream>
using namespace std;
int n;
void mov(int n,char a,char c,char b)
{
    //用 b 杆作为协助过渡,将 a 杆上的 n 个盘子移动到 c 杆上
    if(n==0) return;
    mov(n-1,a,b,c); //用 c 杆作为协助过渡,将 a 杆上的(n-1)个盘子移动到 b 杆上
    cout<<a<<"->"<<n<<"->"<<c<<endl;
    //输出方案
    mov(n-1,b,c,a); //用 a 杆作为协助过渡,将 b 杆上的(n-1)个盘子移动到 c 杆上
}
int main()
{
    char x,y,z;
    cin>>n;
    cin>>x>>y>>z;
    mov(n,x,y,z); //调用模拟函数
    return 0;
}
```

9. 放苹果【2.2 基本算法之递归 666】

把 m 个同样的苹果放在 n 个同样的盘子里,允许有的盘子空着不放,问:共有多少种不同的分法(用 k 表示)? 注意:5,1,1 和 1,5,1 是同一种分法。

输入:

第 1 行是测试数据的数目 $t(0 \leq t \leq 20)$ 。以下每行均包含两个整数 m 和 $n(1 \leq m, n \leq 10)$,以空格分开。

输出:

对输入的每组数据 m 和 n ,用一行输出相应的 k 。

样例输入:

1

7 3

样例输出:

8

【参考程序】

```
#include<cstdio>
using namespace std;
int t,m,n;
int up(int m,int n)
{
    if (m==0||n==1) return 1;
        //当盘子 1 个或苹果 0 个时,方法只有一种
    else if (m<n) return up(m,m);
        //当盘子大于苹果时,相当于把 m 个苹果放在 m 个盘子里
    else return up(m-n,n)+up(m,n-1);
        //每个盘子至少放一个苹果的分法数加上至少一个盘子为空的分法数的和
}
int main()
{
    scanf("%d",&t);
    while (t-->0)
    {
        scanf("%d%d",&m,&n);
        printf("%d\n",up(m,n));
    }
    return 0;
}
```

10. 求最大公约数问题【2.2 基本算法之递归 7592】

给定两个正整数,求它们的最大公约数。

输入:

一行,包含两个正整数(小于 1,000,000,000)。

输出:

一个正整数,即这两个正整数的最大公约数。

样例输入:

6 9

样例输出:

3

【参考程序】

```
#include<iostream>
using namespace std;
int gcd(int,int);
int main()
{
    int m,n;
    cin>>m>>n;
    cout<<gcd(m,n)<<endl;
    return 0;
}
int gcd(int m,int n)
{
    if(m%n==0) return n;
    else return gcd(n,m%n);
}
```

//直接调用函数输出
//直到 n 为 m 因数时返回最大公约数 n
//辗转相除法

11.2 的幂次方表示【2.2 基本算法之递归 8758】

任何一个正整数都可以用 2 的幂次方表示。例如：

$$137 = 2^7 + 2^3 + 2^0$$

同时约定方次用括号来表示，即 a^b 可表示为 $a(b)$ 。由此可知，137 可表示为：

$$2(7) + 2(3) + 2(0)$$

进一步： $7 = 2^2 + 2 + 2^0$ (2^1 用 2 表示)

$$3 = 2 + 2^0$$

所以最后 137 可表示为：

$$2(2(2) + 2 + 2(0)) + 2(2 + 2(0)) + 2(0)$$

又如：

$$1315 = 2^{10} + 2^8 + 2^5 + 2 + 1$$

所以 1315 最后可表示为：

$$2(2(2 + 2(0)) + 2) + 2(2(2 + 2(0))) + 2(2(2) + 2(0)) + 2 + 2(0)$$

输入：

一个正整数 $n(n \leq 20,000)$ 。

输出：

一行，符合约定的 n 的 0,2 表示(在表示中不能有空格)。

样例输入：

137

样例输出：

$$2(2(2) + 2 + 2(0)) + 2(2 + 2(0)) + 2(0)$$

【参考程序】

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
using namespace std;
int n;
int a[17];
int work(int x)
{
    if (x==0)
    {
        printf("0 ");
        return 0;
    }
    if (x==2)
    {
        printf("2 ");
        return 0;
    }
    //x=0 或 2 时不可以再拆分,直接返回原值
    while (1)
    {
        int i;
        for (i=16;a[i]>x;i--);
        //寻找小于 x 的最大 2 次幂
        if (a[i]!=2)
        {
            printf("2(");
```

```

        work(i);           //传递 i 继续拆分
        printf(")");       //补加末尾括号
    }
    else
    {
        printf(" 2 ");      //特判 2, 避免把 2 变成 2(2(0))
    }
    if (x-a[i]) printf("+"), x=x-a[i];
                                //当拆分完大的部分后仍有剩余时, 继续拆分小的部分
    else return 0;           //已拆分完则退出
}
}

int main()
{
    a[0]=1;
    for (int i=1; i<=16; i++) a[i]=a[i-1]*2;
                                //预处理出 2 的 1~16 次幂
    scanf("%d", &n);
    work(n);
}

```

12. 分数求和【1.13 编程基础之综合应用 12】

输入 n 个分数并对它们求和, 并用最简形式表示。所谓最简形式是指: 分子分母的最大公约数为 1; 若最终结果的分母为 1, 则直接用整数表示。

如: $5/6, 10/3$ 均是最简形式, 而 $3/6$ 需要化简为 $1/2$, $3/1$ 需要化简为 3。

分子和分母均不为 0, 也不为负数。

输入:

第 1 行是一个整数 $n(1 \leq n \leq 10)$, 表示分数个数。

接下来 n 行, 每行一个分数, 用“ p/q ”的形式表示, 不含空格, p, q 均不超过 10。

输出:

只有一行, 即最终结果的最简形式。若为分数, 用“ p/q ”的形式表示。

样例输入:

```

2
1/2
1/3

```

样例输出:

```

5/6

```

【参考程序】 //递归+最大公约数+最小公倍数

```

#include<cstdio>
#include<iostream>
using namespace std;
int n, p[10001], q[10001], gbs=1, totfz;
int gcd(int a, int b)           //求最大公约数
{
    return b? gcd(b, a%b):a;    //辗转相除法
}
int lcm(int a, int b)

```



```

{
    return a * b / gcd(a, b);           //最小公倍数:两数相乘除以最大公约数
}

int main()
{
    scanf("%d", &n);
    for(int i=1; i<=n; ++i)
    {
        scanf("%d/%d", &p[i], &q[i]);
        gbs = lcm(gbs, q[i]);
    }
    for(int i=1; i<=n; ++i)
        totfz += p[i] * (gbs / q[i]);
    int ansgcd = gcd(max(totfz, gbs), min(totfz, gbs));
    totfz /= ansgcd; gbs /= ansgcd;      //分子和分母分别约分至最简
    printf("%d", totfz);
    if(gbs != 1) printf("/%d", gbs);
}

```

13. 因子分解【1.13 编程基础之综合应用 22】

一个数,输出其素因子分解表达式。

输入:

一个整数 n ($2 \leq n < 100$)。

输出:

输出该整数的因子分解表达式。

表达式中各个素数从小到大排列。

如果该整数可以分解出因子 a 的 b 次方,当 b 大于 1 时,写作 a^b ; 当 b 等于 1 时,则直接写成 a 。

样例输入:

60

样例输出:

$2^2 * 3 * 5$

【参考程序】

```

#include <cstdio>
using namespace std;
int n, b[10001], a[10001];
void search(int x, int y)
{
    if((x == 0) || (y > x)) return;      //递归边界
    while(x % y == 0) x /= y, a[y]++;    //当 y 为 x 因子时, x/y, y 的次幂+1
    search(x, y+1);                      //否则从 y+1 开始继续判断
}

int main()
{
    scanf("%d", &n);
    search(n, 2);
    bool B=0;

```

```

for(int i=2;i<=n;++i)
{
    if(B&&.a[i]) printf(" * ");           //第一次不输出乘号,之后依次输出
    if(a[i]) B=1;
    if(a[i]==1) printf("%d",i);           //次幂为 1 时,直接输出数
    else if(a[i]>1) printf("%d^%d",i,a[i]); //否则输出数^次幂
}
return 0;
}

```

14. 判断元素是否存在【1.13 编程基础之综合应用 41】

有一个集合 M 是这样生成的:

- (1) 已知 k 是集合 M 的元素;
- (2) 如果 y 是 M 的元素,那么, $2y+1$ 和 $3y+1$ 都是 M 的元素;
- (3) 除了上述两种情况外,没有别的数能够成为 M 的一个元素。

问题:任意给定 k 和 x ,请判断 x 是否是 M 的元素。这里的 k 是无符号整数, x 不大于 100,000,如果是,则输出 YES,否则,输出 NO。

输入:

整数 k 和 x , 逗号间隔。

输出:

如果是,则输出 YES,否则,输出 NO。

样例输入:

0,22

样例输出:

YES

【参考程序】

```

#include<cstdio>
#include<iostream>
using namespace std;
int x,k;
bool search(int y)           //判定元素是否在集合中
{
    if(y>x) return 0;        //如果 y>x,则 y 必定不在集合中
    if(y==x) return 1;       //如果相等,则 y 必定在集合中
    if(y<x) return search(2 * y+1) || search(3 * y+1);
                                //如果 y 是 M 的元素,那么,2y+1 和 3y+1 都是 M 的元素
}
int main()
{
    scanf("%d,%d",&k,&x);
    if(search(k)) printf(" YES ");
    else printf(" NO ");
    return 0;
}

```

第五章 搜索与回溯算法

1. LETTERS【2.5 基本算法之搜索 156】

给出一个 $roe \times col$ 的大写字母矩阵,一开始的位置为左上角,你可以向上下左右四个方向移动,并且不能移向曾经经过的字母。问:最多可以经过几个字母?

输入:

第 1 行,输入字母矩阵行数 R 和列数 $S, 1 \leq R, S \leq 20$ 。

接着输入 R 行 S 列字母矩阵。

输出:

最多能走过的不同字母的个数。

样例输入:

样例输出:

3 6

6

HFDFFB

AJHGDH

DGAGEH

【参考程序】

```
#include<iostream>
#include<cstring>
using namespace std;
int fx[4]={0,1,0,-1};
int fy[4]={-1,0,1,0};           //四个方向
int ans=0,d[200][200],n,m;
char a[200][200];bool p[50000];
int dfs(int x,int y,int z)       //z 表示已经走过的步数
{
    ans=max(ans,z);             //ans 记录最多步数
    for(int i=0;i<4;i++)
    {
        int xx=x+fx[i];
        int yy=y+fy[i];
        if(xx<=n&&xx>0&&yy<=m&&yy>0&&! p[a[xx][yy]])
        {
            p[a[xx][yy]]=true;   //判断是否越界及是否能走
            dfs(xx,yy,z+1);       //标记走过的字母
            p[a[xx][yy]]=false;   //以(xx,yy)继续搜索,走过字母数加 1
            //回溯法,退回上一步重新搜索方案
        }
    }
}

int main()
{
    memset(p,false,sizeof(p));   //初始化,默认每个点都能走
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
```

```

    cin>>a[i][j];
    p[a[1][1]]=true;           //(1,1)记为已经走过,避免回到起点
    dfs(1,1,1);               //从(1,1)开始搜索,初始步数为 1
    cout<<ans<<endl;
    return 0;
}

```

2. 八皇后问题【2.5 基本算法之搜索 1700】

在国际象棋棋盘上放置八个皇后,要求每两个皇后之间不能直接吃掉对方。

输入:

无输入。

输出:

按给定顺序和格式输出所有八皇后问题的解(见 Sample Output)。

样例输入:

无输入。

样例输出:

```

No. 1
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
No. 2
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
... 以下省略

```

【参考程序】

```

#include<iostream>
using namespace std;
int a[10001],b[10001],w[10001],m[10001],tot=0;
int print()
{
    tot++;
    cout<<" No. " <<tot<<endl;
    for(int j=1;j<=8;++j)           //判断输出八皇后摆放矩阵
    {
        for(int i=1;i<=8;++i)
            if(j==a[i])
                cout<<1<<<" ";
    }
}

```

```

else
    cout<<0<<" ";
    cout<<endl;
}
}
int search(int j)
{
    for(int i=1;i<=8;++i)                //搜索可以放的 8 个位置
    {
        if(b[i]==0&&w[i-j+7]==0&&m[i+j]==0) //判断条件是否成立
        {
            a[j]=i;                        //记录列
            b[i]=1;                        //占领列,将占领的列标记为 1
            w[i-j+7]=1;
            m[i+j]=1;                      //占领两个对角线
            if(j==8)print();               //如果放置好输出方案
            else search(j+1);              //摆放下一个皇后
            b[i]=0;
            w[i-j+7]=0;
            m[i+j]=0;                      //回溯法,收回上一个放出的皇后
        }
    }
}
int main()
{
    search(1);                            //从第一个皇后开始放置
    return 0;
}

```

3. 八皇后【2.5 基本算法之搜索 1756】

会下国际象棋的人都很清楚:皇后可以在横、竖、斜线上不限步数地吃掉其他棋子。如何将 8 个皇后放在棋盘上(有 8×8 个方格),使它们谁也不能被吃掉!这就是著名的八皇后问题。

对于某个满足要求的 8 皇后的摆放方法,定义一个皇后串 a 与之对应,即 $a=b_1b_2\cdots b_8$,其中 b_i 为相应摆法中第 i 行皇后所处的列数。已经知道 8 皇后问题一共有 92 组解(即 92 个不同的皇后串)。

给出一个数 b ,要求输出第 b 个串。串的比较是这样的:皇后串 x 置于皇后串 y 之前,当且仅当将 x 视为整数时比 y 小。

输入:

第 1 行是测试数据的组数 n ,后面跟着 n 行输入。每组测试数据占 1 行,包括一个正整数 $b(1 \leq b \leq 92)$ 。

输出:

有 n 行,每行输出对应一个输入。输出应是一个正整数,是对应于 b 的皇后串。

样例输入:

```

2
1
92

```

样例输出:

15863724

84136275

【参考程序】

```
#include <cstring>
#include <iostream>
using namespace std;
int js(int), sc();
int a[9];
bool b[9], c[17], d[17];
int ans, n;
int main()
{
    memset(b, false, sizeof(b));
    memset(c, false, sizeof(c));
    memset(d, false, sizeof(d)); //全默认标记为没被占领
    int k;
    cin >> k;
    for (int i = 1; i <= k; ++i)
    {
        ans = 0;
        cin >> n;
        js(1); //放置第一个皇后
    }
    return 0;
}

int js(int i)
{
    for (int j = 1; j <= 8; ++j) //搜索可以放的 8 个位置
        if (b[j] == false && c[i+j] == false && d[i-j+7] == false)
        {
            a[i] = j; //记录列
            b[j] = true; //占领过的列记录为真
            c[i+j] = true;
            d[i-j+7] = true; //占领两个对角线
            if (i == 8) sc(); //满足条件, 输出
            else js(i+1); //自调用函数
            b[j] = false; //回溯法, 收回上一个放出的皇后
            c[i+j] = false;
            d[i-j+7] = false;
        }
    return 0;
}

int sc()
{
    ans++;
    if (ans == n)
```

```

{
    for (int i=1;i<=8;++i)
        cout<<a[i];                //输出皇后串
    cout<<endl;
}
return 0;
}

```

4. 迷宫【2.5 基本算法之搜索 1792】

一天 Extense 在森林里探险的时候不小心走入了一个迷宫,迷宫可以看成是由 $n * n$ 的格点组成,每个格点只有 2 种状态:“.”和“#”,前者表示可以通行,后者表示不能通行。同时当 Extense 处在某个格点时,他只能移动到东南西北(或者说上下左右)四个方向之一的相邻格点上,Extense 想要从点 A 走到点 B,问:在不走出迷宫的情况下能不能办到? 如果起点或者终点有一个不能通行(为“#”),则看成无法办到。

输入:

第 1 行是测试数据的组数 k ,后面跟着 k 组输入。每组测试数据的第 1 行是一个正整数 n ($1 \leq n \leq 100$),表示迷宫的规模是 $n * n$ 的。接下来是一个 $n * n$ 的矩阵,矩阵中的元素为“.”或者“#”。再接下来一行是 4 个整数 ha, la, hb, lb ,描述 A 处在第 ha 行,第 la 列,B 处在第 hb 行,第 lb 列。注意到 ha, la, hb, lb 全部是从 0 开始计数的。

输出:

k 行,每行输出对应一个输入。能办到则输出“YES”,否则输出“NO”。

样例输入:

```

2
3
. # #
. . #
# . .
0 0 2 2
5
. . . .
# # # . #
. . # .
# # # .
. . . #
0 0 4 0

```

样例输出:

```

YES
NO

```

【深搜程序】

```

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int dx[4]={1,-1,0,0},                //四个方向
    dy[4]={0,0,1,-1};
int t,n,x,y,xl,yl,nx,ny,m;
char str;

```

```

bool pd, flag[1001][1001];
int dfs(int x, int y)
{
    for(int i=0; i<4; i++)
    {
        nx=x+dx[i]; ny=y+dy[i];           //搜索四个方向
        if(nx>=0&&nx<n&&ny>=0&&ny<n&&! flag[nx][ny])
        {                                   //判断是否越界及是否走过
            flag[nx][ny]=true;
            if(nx==x1&&ny==y1)             //判断,到达后输出 YES
            {
                cout<<" YES "<<endl;
                pd=true;
                break;
            }
            else dfs(nx, ny);               //否则从现在的点开始继续搜索
        }
    }
}

int main()
{
    cin>>m;
    for(int t=1; t<=m; t++)
    {
        memset(flag, false, sizeof(flag));
        pd=false;
        cin>>n;
        for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
        {
            cin>>str;
            if(str=="#") flag[i][j]=true;    //“#”记为不可走
        }
        cin>>x>>y>>x1>>y1;
        if(flag[x][y]||flag[x1][y1])         //如果起点或终点是障碍直接输出 NO
        {
            cout<<" NO "<<endl; continue;
        }
        else dfs(x, y);                      //从起点开始搜索
        if(! pd) cout<<" NO "<<endl;
    }
}

```

【广搜程序】

```

#include<iostream>
#include<cstring>
using namespace std;
int dx[4]={-1,0,1,0},
    dy[4]={0,-1,0,1};

```



```

bool a[1001][1001]; char c[1001][1001]={0};
int dl[10005][4],n,m,tot,i,j,x,y,h1,h2,k;
int xx,yy,t=1,w=2,f=0;
int main()
{
    cin>>k;
    for (int l=1;l<=k;l++)
    {
        memset(a,0,sizeof(a));
        memset(dl,0,sizeof(dl));
        t=1,w=2,f=0;
        cin>>n;
        for (i=0;i<n;++i) cin>>c[i];
        cin>>x>>y>>h1>>h2;
        if (c[x][y]=='#' || c[h1][h2]=='#') cout<<" NO "<<endl;
        //起点或终点都为障碍时直接输出 NO
    }
    else
    {
        a[x][y]=true; dl[1][1]=x; dl[1][2]=y;
        while(t<w)
        {
            for (i=0; i<4; ++i) //往四个方向搜索
            {
                xx=dl[t][1]+dx[i];
                yy=dl[t][2]+dy[i]; //查找位置
                if(xx<0 || xx>n-1 || yy<0 || yy>n-1) continue;
                //判断是否越界
                if (c[xx][yy]!='#' && ! a[xx][yy])
                //判断是否走过
                {
                    a[xx][yy]=true; // (xx,yy)记为走过,避免重复
                    dl[w][1]=xx;
                    dl[w][2]=yy;
                    w++;
                }
                if (xx==h1 && yy==h2) //走到终点时输出并结束广搜
                {
                    cout<<" YES "<<endl; f=1;
                    break;
                }
            }
            if (f==1) break;
            t++;
        }
        if (f==0) cout<<" NO "<<endl; } //当走不到目的地输出 NO
    }
}

```

5. 红与黑【2.5 基本算法之搜索 1818】

有一间长方形的房子,地上铺了红色、黑色两种颜色的正方形瓷砖。你站在其中一块黑色的瓷砖上,只能向相邻的黑色瓷砖移动。请写一个程序,计算你总共能够到达多少块黑色的瓷砖。

输入:

包括多个数据集合。每个数据集合的第 1 行是两个整数 W 和 H,分别表示 x 方向和 y 方向瓷砖的数量。W 和 H 都不超过 20。在接下来的 H 行中,每行包括 W 个字符。每个字符表示一块瓷砖的颜色,规则如下:

(1)“.”:黑色的瓷砖;

(2)“#”:白色的瓷砖;

(3)“@”:黑色的瓷砖,并且你站在这块瓷砖上。该字符在每个数据集合中唯一出现一次。

当在一行中读入的是两个零时,表示输入结束。

输出:

对每个数据集合,分别输出一行,显示你从初始位置出发能到达的瓷砖数(记数时包括初始位置的瓷砖)。

样例输入:

```
6 9
. . . . # .
. . . . . #
. . . . . .
. . . . . .
. . . . . .
. . . . . .
# @ . . . #
. # . . # .
0 0
```

样例输出:

```
45
```

【深搜程序】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
char s;
int dx[]={-1,1,0,0},dy[]={0,0,-1,1}; //四个方向
int pd[101][101];
int x,y,tot=1,n,m,a[1001][1001],xx,yy;
void dfs(int x,int y)
{
    for(int i=0;i<4;i++)
    {
        xx=x+dx[i];yy=y+dy[i]; //搜索四个方向
        if(xx>=1&&xx<=n&&yy>=1&&yy<=m&&a[x][y]==a[xx][yy]==
        1&&! pd[xx][yy])
```

```

    {
        pd[xx][yy]=1;
        tot++;
        dfs(xx,yy);
    }
}

int main()
{
    for(int q=1;q++;)
    {
        scanf("%d%d",&m,&n);
        if(n==0&&m==0) break;
        memset(a,0,sizeof(a));
        memset(pd,0,sizeof(pd));
        tot=1;
        for(int i=1;i<=n;i++)
        for(int j=1;j<=m;j++)
        {
            cin>>s;
            if(s=='@')
            {
                x=i,y=j;
                a[i][j]=1;
            }
            if(s=='.' ) a[i][j]=1;
            if(s=='#') a[i][j]=0;
        }
        pd[x][y]=1;
        dfs(x,y);
        cout<<tot<<endl;
    }
    return 0;
}

```

//判断是否越界,是否能走,是否走过
//记为走过的黑砖
//瓷砖数+1
//以此位置继续查找

//输入行列
//初始化:清空数组
//瓷砖数初始化为1
//把起点定为@位置
//起点记为不可走,避免回到起点
//"."记为可走
//"#”记为不可走
//起点记为不可走,避免回到起点
//从起点开始搜索

【广搜程序】

```

#include<iostream>
using namespace std;
int dx[4]={-1,0,1,0},dy[4]={0,1,0,-1}; //四个方向
int bz[1001][1001]={0},n=1,m=1,num[10001]={0},tot; string s[1001];
void gsy(int p,int q)
{
    int x,y,t,w,i;
    int h[1001][3];
    bz[p][q]=0;tot=1;
    t=0; w=1; h[1][1]=p; h[1][2]=q;
    do
    {

```

```

t++;
for(i=0;i<=3;i++) //向四个方向搜索
{
    x=h[t][1]+dx[i]; y=h[t][2]+dy[i];
    if ((x>=0)&&(x<=n)&&(y>=0)&&(y<=m)&&(bz[x][y]==1))
    { //判断是否越界及是否可走
        w++; tot++;
        h[w][1]=x;
        h[w][2]=y;
        bz[x][y]=0; //已走过即为不可走
    }
}
} while (t<w);
cout<<tot<<endl; //输出走过的黑砖的个数
}

int main()
{
    int i,j,xxx,yyy;
    for(;;)
    {
        cin>>m>>n; if (m==0&&n==0) break;
        for(i=0;i<=1000;i++)
            for(j=0;j<=1000;j++) bz[i][j]=0;
        for(i=1;i<=n;i++)
        {
            cin>>s[i];
            for(j=0;j<=m-1;j++)
                if (s[i][j]=='.' || s[i][j]=='@')
                    bz[i][j]=1; //起点和“.”记为可走
        }
        for(i=1;i<=n;i++)
        {
            for(j=0;j<=m-1;j++)
                if (s[i][j]=='@') gsy(i,j);
        }
    }
    return 0;
}

```

6. 棋盘问题[2.5 基本算法之搜索 323]

在一个给定形状的棋盘(形状可能是不规则的)上面摆放棋子,棋子没有区别。要求摆放时任意的两个棋子不能放在棋盘中的同一行或者同一列,请编程求解对于给定形状和大小的棋盘,摆放 k 个棋子的所有可行的摆放方案 C 。

输入:

含有多组测试数据。

每组数据的第 1 行是两个正整数, n k ($n \leq 8$, $k \leq n$), 用一个空格隔开, 表示了将在一个 $n \times n$ 的矩阵内描述棋盘, 以及摆放棋子的数目。

当为 -1 -1 时表示输入结束。

随后的 n 行描述了棋盘的形状:每行有 n 个字符,其中“#”表示棋盘区域,表示空白区域(数据保证不出现多余的空白行或者空白列)。

输出:

对于每一组数据,给出一行输出,输出摆放的方案数目 C (数据保证 $C < 2^{31}$)。

样例输入:

```
2 1
#.
.#
4 4
...#
..#.
.#..
#...
-1-1
```

样例输出:

```
2
1
```

【参考程序】

```
#include<cstring>
#include<iostream>
using namespace std;
void js(int,int);
char a[9][9];
bool b[9];
int n,k,ans;
int main()
{
    while (cin>>n>>k)                //循环输入 n,k
    {
        if (n== -1 && k== -1) return 0;
        memset(b,true,sizeof(b));
        for (int i=1;i<=n;++i)
            for (int j=1;j<=n;++j) cin>>a[i][j];
        ans=0;
        js(1,0);                        //从第一行开始搜索,落子数初始化为 1
        cout<<ans<<endl;
    }
}

void js(int x,int y)
{
    if (y==k)                          //判断是否落满子
    {
        ans++;                          //方案数+1
        return;
    }
    for (int i=x;i<=n;++i)              //从 x 行开始,x 行是上一行+1
        for (int j=1;j<=n;++j)          //当前行搜索每 1 列
```

```

if (a[i][j]=='#' && b[j])           //判断当前点是否能落子及是否落过子
{
    b[j]=false;
    js(i+1,y+1);                   //自调用函数,搜索下一行
    b[j]=true;                     //回溯法,收回上一枚棋子,重新搜索
}
return;
}

```

7. 取石子游戏【2.5 基本算法之搜索 6266】

有两堆石子,两个人轮流去取。每次取的时候,只能从较多的那堆石子里取,并且取的数目必须是较少的那堆石子数目的整数倍,最后谁能够把一堆石子取空谁就算赢。

比如初始的时候两堆石子的数目是 25 和 7。

25 7	-->	11 7	-->	4 7	-->	4 3	-->	1 3	-->	1 0
	选手 1 取		选手 2 取		选手 1 取		选手 2 取		选手 1 取	

最后选手 1(先取的)获胜,在取的过程中选手 2 都只有唯一的一种取法。

给定初始时石子的数目,如果两个人都采取最优策略,请问先手能否获胜。

输入:

包含多数数据。每组数据一行,包含两个正整数 a 和 b ,表示初始时石子的数目。

输入以两个 0 表示结束。

输出:

如果先手胜,输出“win”,否则输出“lose”。

样例输入:

34 12

15 24

0 0

样例输出:

win

lose

提示:

假设石子数目为 (a, b) 且 $a \geq b$, 如果 $[a/b] \geq 2$ 则先手必胜, 如果 $[a/b] < 2$, 那么先手只有唯一的一种取法。 $[a/b]$ 表示 a 除以 b 取整后的值。

【参考程序 1】

```

#include<iostream>
using namespace std;
bool js(long long a, long long b)
{
    if(a/b >= 2 || a == b) return true;           //是否满足先手必胜的条件
    else return ! js(b, a-b);                   //两人交换
}
int main()
{
    long long a, b;
    while(cin >> a >> b)
    {
        if(a == 0 || b == 0) return 0;           //终止条件
        if(b > a) swap(a, b);                   //把石子多的那堆放在前面
    }
}

```

```

    if(js(a,b))cout<<" win "<<endl;
    else cout<<" lose "<<endl;
}
}

```

【参考程序 2】

```

#include<iostream>
using namespace std;
bool dfs(int a,int b)
{
    if(a<b) swap(a,b);
    if(a%b==0) return true;
    for(int k=a/b;k>=1;k--) //题目提示
        if(! dfs(a-b*k,b))
            return true;
    return false;
}
int main()
{
    int a,b;
    while(cin>>a>>b)
    {
        if(a==0&& b==0) break; //如果读入 0 0 则结束
        if(dfs(a,b)) cout<<" win "<<endl;
        else cout<<" lose "<<endl;
    }
}

```

8. 马走日【2.5 基本算法之搜索 8465】

马在中国象棋以日字形规则移动。

请编写一段程序,给定 $n * m$ 大小的棋盘,以及马的初始位置 (x,y) ,要求不能重复经过棋盘上的同一个点,计算马可以有多少途径遍历棋盘上的所有点。

输入:

第 1 行为整数 $T(T < 10)$,表示测试数据组数。

每一组测试数据包含一行,为四个整数,分别为棋盘的大小以及初始位置坐标 n,m,x,y ($0 \leq x \leq n-1, 0 \leq y \leq m-1, m < 10, n < 10$)。

输出:

每组测试数据包含一行,为一个整数,表示马能遍历棋盘的途径总数,0 为无法遍历一次。

样例输入:

```

1
5 4 0 0

```

样例输出:

```

32

```

【参考程序】

```

#include<cstdio>
using namespace std;
int fx[8]={1,1,2,2,-1,-1,-2,-2};
int fy[8]={2,-2,1,-1,2,-2,1,-1}; //八个方向

```

```

int ans,N,n,m,sx,sy;
bool visited[101][101];
void work(int x,int y,int tot)
{
    if(tot==n*m) ans++;           //到达终点后方案数+1
    else
    for(int i=0;i<8;++i)
    {
        int X=x+fx[i],Y=y+fy[i];
        if((X>=0)&&(X<n)&&(Y>=0)&&(Y<m)&&(! visited[X][Y]))
            //判断是否越界及是否已走过
        {
            visited[x][y]=1;
            work(X,Y,tot+1);
            visited[x][y]=0;       //回溯法,退回上一步重新搜索
        }
    }
}

int main()
{
    scanf("%d",&N);
    for(int i=1;i<=N;++i)
    {
        scanf("%d%d%d%d",&n,&m,&sx,&sy);
        work(sx,sy,1);           //从(sx,sy,1)开始搜索
        printf("%d\n",ans);
        ans=0;
    }
}

```

9. 单词接龙【2.5 基本算法之搜索 8783】

单词接龙是一个与我们经常玩的成语接龙相类似的游戏,现在我们已知一组单词,且给定一个开头的字母,要求出以这个字母开头的最长的“龙”(每个单词都最多在“龙”中出现两次),在两个单词相连时,其重合部分合为一部分,例如 *beast* 和 *astonish*,如果接成一条龙则变为 *beastonish*,另外相邻的两部分不能存在包含关系,例如 *at* 和 *atide* 间不能相连。

输入:

第 1 行为一个单独的整数 $n(n \leq 20)$,表示单词数,以下 n 行每行有一个单词(只含有大写或小写字母,长度不超过 20),输入的最后一行为一个单个字符,表示“龙”开头的字母。你可以假定以此字母开头的“龙”一定存在。

输出:

只需输出以此字母开头的最长的“龙”的长度。

样例输入:

```

5
at
touch
cheat
choose
tact
a

```


样例输出:

23

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
char str[101][106],st;
int jl[101],n,tot=0,maxx=0,l[101],n2;
void dfs(int x,int y)
{
    if(tot>maxx) maxx=tot;           //出解
    for(int i=1;i<=n2;i++)
    {
        if(x==1)
        {
            if(str[i][0]!=st) continue; //必须为特定字母开头
            l[i]=1; tot+=jl[i];
            dfs(x+1,i);
            l[i]=0; tot-=jl[i];
        }
        if (l[i]==1) continue;
        int c=1,k=jl[y]-1;
        char a[1001]={'\0'},b[1001]={'\0'};
        while(c<=jl[i]&& c<jl[y])
        {
            strncpy(a,str[i],c);           //作用:单词从开头向右复制 c 个字母到 a 数组
            strncpy(b,&str[y][k--],c);
            if(strcmp(a,b)==0)             //判断两个字符串是否相等
            {
                tot+=jl[i]-c;
                l[i]=1;
                dfs(x+1,i);
                tot-=jl[i]-c;
                l[i]=0;
                break;                     //最小 c 才有最大龙
            }
            c++;
        }
    }
}

int main()
{
    cin>>n;
    n2=2*n;
    for(int i=1;i<=n;i++)
```

```

{
    cin>>str[i];
    jl[i]=strlen(str[i]);
}
for(int i=n+1;i<=n2;i++) //复制一次数组,因为每个单词用两遍
{
    strcpy(str[i],str[i-n]); //复制
    jl[i]=jl[i-n]; //数组扩大一倍
}
cin>>st;
dfs(1,0); // 处理第一个单词,上一个单词为 0(没有)
cout<<maxx<<endl;
}

```

10. 分成互质组【2.5 基本算法之搜索 7834】

给定 n 个正整数,将它们分组,使得每组中任意两个数互质。至少要分成多少个组?

输入:

第 1 行是一个正整数 $n(1 \leq n \leq 10)$;

第 2 行是 n 个不大于 10,000 的正整数。

输出:

一个正整数,即最少需要的组数。

样例输入:

6

14 20 33 117 143 175

样例输出:

3

【参考程序】

```

#include<cstdio>
#include<cstdlib>
#include<cmath>
#include<algorithm> //sort 调用库
using namespace std;
int n,tot,Whose[11],a[11];
int Min=0,Max=0x7ffffff;
int c[11];
int gcd(int x,int y) //最大公因数
{
    int r=x%y;
    while (r)
    {
        x=y;
        y=r;
        r=x%y; //辗转相除法
    }
    return y;
}
int main()
{

```

```

scanf("%d",&n);
for (int i=1;i<=n;i++) scanf("%d",&a[i]);
sort(a+1,a+1+n);           //从小到大快排一遍
for (int i=1;i<=n;i++)
{
    if (!c[i])               //判断 a[i] 是否使用过
    {
        Min++;               //每找得到一次,组数+1
        for (int j=n;j>=i+1;j--) //从大到小更优
        {
            if (gcd(a[i],a[j])==1&&!c[j])
            {
                //判断 a[i] * a[j] 互质且 a[j] 未被使用过
                a[i] *= a[j]; //包含本组所有质因子
                c[j]=1;       //记录 a[j] 被使用过
            }
        }
    }
}
printf("%d",Min);
}

```

11. 放苹果【2.5 基本算法之搜索 666】

把 m 个同样的苹果放在 n 个同样的盘子里,允许有的盘子空着不放,问:共有多少种不同的分法(用 k 表示)? 注意:5,1,1 和 1,5,1 是同一种分法。

输入:

第 1 行是测试数据的数目 t ($0 \leq t \leq 20$)。以下每行均包含两个整数 m 和 n ($1 \leq m, n \leq 10$),以空格分开。

输出:

对输入的每组数据 m 和 n ,用一行输出相应的 k 。

样例输入:

```

1
7 3

```

样例输出:

```

8

```

【参考程序】

```

#include<iostream>
using namespace std;
int x(int,int);
int a[11]={1},ans,n,m;
int main()
{
    int k;
    cin>>k;
    for (int i=1;i<=k;++i)
    {
        ans=0;
        cin>>m>>n;
        x(m,1);
    }
}

```

```

        cout<<ans<<endl;
    }
    return 0;
}
int x(int s,int k)
{
    for (int i=a[k-1];i<=s;++i)
    if (i<=m)
    {
        s-=i;
        a[k]=i;
        if (s==0&& k<=n) ans++; //判断是否满足条件
        else x(s,k+1);          //自调用函数
        s+=i;                    //回溯
    }
    return 0;
}

```

第六章 贪心算法

1. An Easy Problem【4.6 算法之贪心 1455 poj2453】

题意:给定一个正整数 n ,求最小的、比 n 大的正整数 m ,使得 m 与 n 的二进制表示中有相同数目的 1。

举个例子,假如给定的 n 为 78,其二进制表示为 1001110,包含 4 个 1,那么最小的比 n 大的并且二进制表示中只包含 4 个 1 的数是 83,其二进制是 1010011,因此 83 就是答案。

输入:

若干行,每行一个数 $n(1 \leq n \leq 1000000)$,输入“0”则结束。

输出:

若干行对应的值。

样例输入:

1
2
3
4
78
0

样例输出:

2
4
5
8
83

【参考程序 1】

```
#include<iostream>
using namespace std;
int main()
{
    int n,ans,s,t;
    while (cin>>n)
    {
        if (n==0) return 0;           //终止条件
        ans=0;
        t=n;
        while (t>0)
        {
            if (t%2==1) ans++;        //ans 储存 n 的二进制形式的 1 的个数
            t/=2;
        }
        s=0;
        for (;;) //循环找寻最小的比 n 大二进制形式的 1 的个数与 n 相等的数
        {
            t=++n;                      //枚举
            while (t>0)
            {
                if (t%2==1) s++;        //s 表示新数为二进制时 1 的个数
                t/=2;
            }
            if (ans==s)
```

```

        {
            //若找到第一个比 n 大且二进制形式 1 的个数与 n 相等,输出并跳出循环
            cout<<n<<endl;
            break;
        }
        s=0;
    }
}

```

【参考程序 2】

```

#include<cstdio>
#include<iostream>
using namespace std;
int AlphGo(int n)
{
    int k=0;
    while (n>0)
    {
        if (n%2==1) ++k;           //用 k 统计 n 为二进制时 1 的个数
        n/=2;
    }
    return k;                      //返回 1 的个数
}
int main()
{
    int n;
    while (cin>>n)                //循环输入 n
    {
        if (n==0) break;          //结束条件
        int y=AlphGo(n),i=n+1;    //y 为 n 二进制时 1 的个数
        while (AlphGo(i) != y) i++;
        //当 i 为二进制时 1 的个数与 n 不相同,则继续寻找,最早找到的就为最小数
        cout<<i<<endl;
    }
}

```

2. 最大子矩阵【4.6 算法之贪心 1768】

已知矩阵的大小定义为矩阵中所有元素的和。给定一个矩阵,你的任务是找到最大的非空(大小至少是 1×1)子矩阵。

比如,如下 4×4 的矩阵

```

0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2

```

的最大子矩阵是

```

9 2
-4 1
-1 8

```

这个子矩阵的大小是 15。

输入:

一个 $n \times n$ 的矩阵。输入的第 1 行给出 n ($0 < n \leq 100$)。再后面的若干行中,依次(首先从左到右给出第 1 行的 n 个整数,再从左到右给出第 2 行的 n 个整数……)给出矩阵中的 n^2 个整数,整数之间由空白字符分隔(空格或者空行)。已知矩阵中整数的范围都在 $[-127, 127]$ 。

输出:

最大子矩阵的大小。

样例输入:

```
4
0   -2  -7   0
9    2  -6   2
-4   1  -4   1
-1   8   0  -2
```

样例输出:

```
15
```

【参考程序】

```
#include<iostream>
using namespace std;
int a[101][101];
int main()
{
    int n,ans,max=-1000000000;    //max 储存最大子矩阵的值
    cin>>n;
    for (int i=1;i<=n;++i)
        for (int j=1;j<=n;++j)
        {
            cin>>a[i][j];
            a[i][j]+=a[i][j-1];    //a[i][j]储存第 i 行前 j 个数字的和
        }
    for (int i=0;i<=n-1;++i)
        for (int j=i+1;j<=n;++j)
        {
            ans=0;
            for (int k=1;k<=n;++k)
            {
                ans+=a[k][j]-a[k][i]; //ans 累加第 k 行第 i 个到第 j 个数字的和
                if (ans>max) max=ans; //当找到更大的子矩阵和,存入最大值
                if (ans<0) ans=0;    //如果 ans 小于 0,清零,重新累加
            }
        }
    cout<<max<<endl;    //输出储存在 max 的最大值
    return 0;
}
```

3. 金银岛【4.6 算法之贪心 1797】

某天 KID 利用飞行器飞到了一个金银岛上,上面有许多珍贵的金属,KID 虽然更喜欢各种宝石的艺术品,可是也不拒绝这样珍贵的金属。但是他只带着一个口袋,口袋至多只能装重量为 w 的物品。岛上金属有 s 个种类,每种金属重量不同,分别为 $n_1, n_2, \dots, n_s$,同时每个种类的金属总的价值也不同,分别为 $v_1, v_2, \dots, v_s$ 。KID 想一次带走价值尽可能多的金属,问:他最多能带走价值多少的金属? 注意:金属是可以被任意分割的,并且金属的价值

和其重量成正比。

输入：

第 1 行是测试数据的组数 k ，后面跟着 k 组输入。

每组测试数据占 3 行，第 1 行是一个正整数 w ($1 \leq w \leq 10,000$)，表示口袋承重上限。

第 2 行是一个正整数 s ($1 \leq s \leq 100$)，表示金属种类。第 3 行有 $2s$ 个正整数，分别为 $n_1, v_1, n_2, v_2, \dots, n_s, v_s$ 分别为第一种，第二种， $\dots$ ，第 s 种金属的总重量和总价值 ($1 \leq n_i \leq 10,000, 1 \leq v_i \leq 10,000$)。

输出：

k 行，每行输出对应一个输入。输出应精确到小数点后 2 位。

样例输入：

样例输出：

```
2
50
4
10 100 50 30 7 34 87 100
10000
5
1 43 43 323 35 45 43 54 87 43
```

```
171.93
508.00
```

【参考程序】

```
#include <cstdio>
double n[101], v[101], c[101];
using namespace std;
void qsort (int, int);
int main()
{
    int k, w, s, i;
    double max;
    scanf("%d", &k); //有 k 个测试数据
    while(k--)
    {
        scanf("%d %d", &w, &s);
        max = 0;
        for (i = 1; i <= s; ++i)
        {
            scanf("%lf %lf", &n[i], &v[i]);
            c[i] = v[i] / n[i]; //c[i] 储存第 i 种金属的单位量价值
        }
        qsort(1, s); //将所有金属的单位量价值从大到小排序
        i = 1;
        while (w - n[i] >= 0 & i <= s) //将目前最大价值的金属装入口袋
        {
            w -= n[i], max += v[i];
            i++; //当金属重量小于剩下的容量则加入价值，减去用过的容量
        }
        if (i <= s & w < n[i]) max += w * c[i];
        //如果口袋还没满，尽可能装入目前最大价值的金属的数目
        printf("%.2lf\n", max);
    }
}
```



```

    return 0;
}

//如果用快速排序函数(sort),要用结构体
void qsort (int l,int r)    //快速排序
{
    int i=l,j=r;
    double mid=c[(l+r)/2],p;
    do
    {
        while (c[i]>mid) i++;
        while (c[j]<mid) j--;
        if (i<=j)
        {
            p=n[i]; n[i]=n[j]; n[j]=p;
            p=v[i]; v[i]=v[j]; v[j]=p;
            p=c[i]; c[i]=c[j]; c[j]=p;
            i++,j--;
        }
    } while (i<=j);
    if (l<j) qsort(l,j);
    if (i<r) qsort(i,r);
}

```

4. 装箱问题【4.6 算法之贪心 19】

一个工厂制造的产品形状都是长方体,它们的高度都是 h ,长和宽都相等,一共有六个型号,他们的长宽尺寸表示分别为 $1 \times 1, 2 \times 2, 3 \times 3, 4 \times 4, 5 \times 5, 6 \times 6$ 。这些产品通常使用一个 $6 \times 6 \times h$ 的长方体包裹包装然后邮寄给客户。因为邮费很贵,所以工厂要想方设法地减小每个订单运送时的包裹数量。他们很需要有一个好的程序帮他们解决这个问题从而节省费用。现在这个程序由你来设计。

输入:

包括几行,每一行代表一个订单。每个订单里的一行包括六个整数,中间用空格隔开,分别为 1×1 至 6×6 这六种产品的数量。输入文件将以 6 个 0 组成的一行结尾。

输出:

除了输入的最后一行 6 个 0 以外,输入文件里每一行对应着输出文件的一行,每一行输出一个整数代表对应的订单所需的最小包裹数。

样例输入:

```

0 0 4 0 0 1
7 5 1 0 0 0
0 0 0 0 0 0

```

样例输出:

```

2
1

```

【参考程序】

```

#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<cstring>

```

```
#include<algorithm>
using namespace std;
int aa[4]={0,5,3,1};
/* 表示 3*3 的产品分别是 4k,4k+1,4k+2,4k+3 时,为 3*3 产品打开的新箱子中剩余
2*2 的空位数 */
int main()
{
    int a=1,b,c,d,e,f,x,y,n;      //a,b,c,d,e,f 表示每种商品个数,n 表示需要的箱子
    while(1)
    {
        cin>>a>>b>>c>>d>>e>>f;
        if (a==0&& b==0&& c==0&& d==0&& e==0&& f==0) break;
        //结束条件
        n=f+e+d+(c+3)/4;          //(c+3)/4 是因为 6*6*1 可以放 4 个 3*3
        y=5*d+aa[c%4];
        /* y 是当前箱子的剩余空间能放 2*2 箱子的个数,没有出现 e,f 是因为 5*5,6*6
        放不下 2*2,5*d 是只放 d 个 4*4 的箱子时能放 2*2 的箱子的个数 */
        if(b>y) n+=(b-y+8)/9;      //如果不够放,增加高度
        x=36*n-36*f-25*e-16*d-9*c-4*b;
        //x 是当前箱子的剩余空间能放 1*1 箱子的个数
        if(a>x) n+=(a-x+35)/36;    //求出最少需要的箱子
        cout<<n<<endl;
    }
    return 0;
}
```

5. Ride to Office【4.6 算法之贪心 2404poj1922】

题意:起点与终点相隔 4500 米。现 Charley 需要从起点骑车到终点。但是,他有个习惯,沿途需要有人陪伴,即以相同的速度,与另外一个人一起骑。而当他遇到以更快的速度骑车的人时,他会以相应的速度跟上这个更快的人。先给定所有与 Charley 同路的人各自的速度与出发时间,问 Charley 以这种方式跟人,骑完 4500 米需要多少时间。得出的结果若是小数,则向上取整。

输入:

若干组数据,每组数据第 1 行 $n(1 \leq n \leq 10000)$,当 n 为 0 时,表示输入结束,接着输入 n 行数据,每行 2 个数据,表示速度 v 和出发时间 t ,如果 $t < 0$,表示陪伴人提早出发了。

输出:

对应若干行数据,每行输出 1 个数,表示最快到达的时间。

样例输入:

```
4
20 0
25 -155
27 190
30 240
2
21 0
22 34
0
```

样例输出:

780

771

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
int n,v,t;
int main()
{   while (cin>>n)           //多组数据
    {
        if (n==0) break;      //结束条件
        int ans=999999,s=0;    //ans 存最短时间
        for (int i=1; i<=n; ++i)
        {
            scanf("%d%d",&v,&t);
            if (t>=0)          //t<0 的情况(即某辆自行车早出发)无需考虑
            {
                double Time=(4500/(v*10.0/36.0)); //求某辆车行驶到办公室的时间
                if (Time-(int)Time>0.0) Time++;    //将所求的时间向上取整
                s=Time+t;                          //加上迟出发的时间,即到达时间(不一定是最优解)
                if (ans>s) ans=s;
                //取多辆自行车处理后的最小值,即为 Charley 到办公室时间的最优解
            }
        }
        printf("%d\n",ans);    //输出最优解
    }
}
```

6. 书架【4.6 算法之贪心 2407】

John 最近买了一个书架用来存放奶牛养殖书籍,但书架很快被存满了,只剩最顶层有空余。

John 共有 N 头奶牛($1 \leq N \leq 20,000$),每头奶牛有自己的高度 H_i ($1 \leq H_i \leq 10,000$), N 头奶牛的总高度为 S 。书架高度为 B ($1 \leq B \leq S < 2,000,000,007$)。

为了到达书架顶层,奶牛可以踩着其他奶牛的背,像叠罗汉一样,直到他们的总高度不低于书架高度。当然若奶牛越多则危险性越大。为了帮助 John 到达书架顶层,找出使用奶牛数目最少的解决方案吧。

输入:

第 1 行:空格隔开的整数 N 和 B 。

第 2~ $N+1$ 行:第 $i+1$ 行为整数 H_i 。

输出:

能达到书架高度所使用奶牛的最少数目

样例输入:

6 40

6

18

11

13

19

11

样例输出:

3

【参考程序】

```

#include<iostream>
#include<algorithm>
using namespace std;
int a[20001];
int main()
{
    int b,n,s=0,t=0;
    cin>>n>>b;
    for (int i=1;i<=n;++i) cin>>a[i];
    sort(a+1,a+n+1);           //将奶牛的高度从矮到高排序
    while (s<b)                 //当奶牛高度达到要求停止循环
    {
        t++;                   //累加奶牛的数量
        s+=a[n-t+1];           //累加奶牛的高度
    }
    cout<<t<<endl;
    return 0;
}

```

7. 电池的寿命【4.6 算法之贪心 2469】

小 S 新买了一个掌上游戏机,这个游戏机由两节 5 号电池供电。为了保证能够长时间玩游戏,他买了很多 5 号电池,这些电池的生产商不同,质量也有差异,因而使用寿命也有所不同,有的能使用 5 个小时,有的可能就只能使用 3 个小时。显然如果他只有两个电池一个能用 5 小时,一个能用 3 小时,那么他只能玩 3 个小时的游戏,有一个电池剩下的电量无法使用,但是如果他有更多的电池,就可以更加充分地利用它们,比如 he 有三个电池,分别能用 3 小时、3 小时、5 小时, he 可以先使用两节能用 3 个小时的电池,使用半个小时后再把其中一个换成能使用 5 个小时的电池,两个半小时后再把剩下的一节电池换成刚才换下的电池(那个电池还能用 2.5 个小时),这样总共就可以使用 5.5 个小时,没有一点浪费。

现在已知电池的数量和电池能够使用的时间,请你找一种方案使得使用时间尽可能的长。

输入:

包含多组数据。每组数据包括两行,第 1 行是一个整数 $N(2 \leq N \leq 1000)$,表示电池的数目,接下来一行是 N 个正整数表示电池能使用的时间。

输出:

对每组数据输出一行,表示电池能使用的时间,保留到小数点后 1 位。

样例输入:

2

3 5

3

3 3 5

样例输出:

3.0

5.5

【参考程序】

```

#include<cstdio>
#include<iostream>
using namespace std;
int main()
{
    int n,max,a;
    double s;
    while (cin>>n)                //当输入 n 时继续循环
    {
        max=0,s=0;
        for (int i=1;i<=n;++i)
        {
            cin>>a;
            s+=a;
            if (a>max) max=a;      //寻找最长时间并累加除最长时间外的所有时间
        }
        if (s-max<max) printf("%.1lf\n",s-max);
        /* 若除最长时间外的所有时间的和小于最长时间,则电池所用的最多时间为除最长时间外的所有时间的和 */
        else printf("%.1lf\n",s/2);
        //电池所用的最多时间为电池所用最多时间的一半(两个电池才能用游戏机)
    }
    return 0;
}

```

8. 寻找平面上的极大点【4.6 算法之贪心 2704】

在一个平面上,如果有两个点 (x,y) , (a,b) ,如果说 (x,y) 支配了 (a,b) ,这是指 $x \geq a$, $y \geq b$;用图形来看就是 (a,b) 坐落在以 (x,y) 为右上角的一个无限的区域内。

给定 n 个点的集合,一定存在若干个点的集合,它们不会被集合中的任何一点所支配,这些点叫做极大点。

编程找出所有的极大点,按照 x 坐标由小到大,输出极大点的坐标。

本题规定: n 不超过 100,并且不考虑点的坐标为负数的情况。

输入:

输入包括两行,第 1 行是正整数 n ,表示是点数;

第 2 行包含 n 个点的坐标,坐标值都是整数,坐标范围从 0~100,输入数据中不存在坐标相同的点。

输出:

按 x 轴坐标最小到大的顺序输出所有极大点。

输出格式为: $(x_1,y_1),(x_2,y_2),\dots,(x_k,y_k)$ 。

注意:输出的每个点之间有“,”分隔,最后一个点之后没有“,”,少输出和多输出都会被判错。

样例输入:

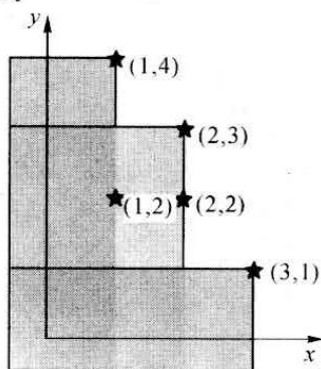
5

1 2 2 2 3 1 2 3 1 4

样例输出:

(1,4),(2,3),(3,1)

提示:



【参考程序】

```
#include<iostream>
#include<cstdio>
using namespace std;
int ans[101][3],a[101][101];
int mx,my,t,m,n,x,y;
int main()
{
    cin>>n;
    for (int i=1; i<=n; i++)
    {
        scanf("%d%d",&x,&y);
        a[x][y]=1;
        mx=max(mx,x);
        my=max(my,y);
    }
    for (int i=mx; i>=0; i--)
        for (int j=my; j>=0; j--)
        {
            if (a[i][j]==1)
            {
                m=j;
                t++;
                ans[t][1]=i;
                ans[t][2]=j;
            }
        }
    for (int i=t; i>=1; i--)
        if (i==1) cout<<'('<<ans[i][1]<<','<<ans[i][2]<<')';
        else cout<<'('<<ans[i][1]<<','<<ans[i][2]<<"),";
    return 0;
}
```

9. 最小新整数【4.6 算法之贪心 3528】

给定一个十进制正整数 n ($0 < n < 1,000,000,000$), 每个数位上数字均不为 0。n 的位数为 m 。

现在从 m 位中删除 k 位 ($0 < k < m$), 求生成的新整数最小为多少?

例如: $n=9,128,456,k=2$,则生成的新整数最小为 12,456。

输入:

第 1 行是一个整数 t , 表示有 t 组数据;

接下来 t 行, 每一行表示一组测试数据, 每组测试数据包含两个数字 n,k 。

输出:

t 行, 每行一个数字, 表示从 n 中删除 k 位后得到的最小整数。

样例输入:

```
2
9128456 2
1444 3
```

样例输出:

```
12456
1
```

【参考程序】

```
#include<iostream>
#include<cstring>
using namespace std;
int main()
{
    int k,t,m;
    string n;
    cin>>t;
    for (int l=1;l<=t;++l)        //有 t 组数据
    {
        cin>>n>>k;
        m=n.size();
        for (int i=1;i<=k;++i)
        {
            for (int j=0;j<m-1;++j)
            if (n[j]>n[j+1])        //找出第一个下降子序列的头一个数字
            {
                for (int x=j;x<m-1;++x) n[x]=n[x+1];
                //将位数向前移一位,删除这个数字
                break;
            }
            m--;
            //数字位数减 1
        }

        for (int i=0;i<m;++i) cout<<n[i];
        cout<<endl;
    }
    return 0;
}
```

10. Crossing River【4.6 算法之贪心 702】

题意: 几个人过河, 每次过两人一人回, 速度由慢者决定, 问过河所需最短时间。

输入:

t 组数据, 每组数据第 1 行输入 n , 第 2 行输入 n 个数, 表示每个人过河的时间。

输出:

t 行数据,每行 1 个数,表示每组过河最少时间。

样例输入:

```
1
4
1 2 5 10
```

样例输出:

```
17
```

【参考程序】

```
#include<cstdio>
#include<algorithm>
using namespace std;
int a[1001];
int main()
{
    int t,n,s;
    scanf("%d",&t);          //t 组数据
    for (int l=1;l<=t;++l)
    {
        scanf("%d",&n);
        for (int i=1;i<=n;++i) scanf("%d",&a[i]);
        sort(a+1,a+n+1);      //将每人的用时从快到慢排序
        s=0;
        while (n>3)  //累加每次将目前最慢与次慢的两个人送过河所需的最短用时
        {
            s+=min(a[1]*2+a[n-1]+a[n],a[1]+a[2]*2+a[n]);
            /* 两种优选策略(选更优的一种),(1)最快和最慢过去,最快回;最快和次慢过去,最快回;(2)最快和次慢过去,最快回;最慢和次慢过去,次慢回。 */
            n-=2;
        }
        if (n==1) s+=a[1];      //若只有 1 人,用时为此人的用时
        else if (n==2) s+=a[2]; //若只剩 2 人,用时则为慢者的用时
        else s+=a[1]+a[2]+a[3]; //若还剩下 3 人,则用时为 3 人的用时和
        printf("%d\n",s);
    }
    return 0;
}
```

11. 接水问题【1.9 编程基础之顺序查找 15】Noip2010 普及组第 2 题

学校里有一个水房,水房里一共装有 m 个龙头可供同学们打开水,每个龙头每秒钟的供水量相等,均为 1。

现在有 n 名同学准备接水,他们的初始接水顺序已经确定。将这些同学按接水顺序 $1 \sim n$ 编号, i 号同学的接水量为 w_i 。接水开始时, $1 \sim m$ 号同学各占一个水龙头,并同时打开水龙头接水。当其中某名同学 j 完成其接水量 w_j 后,下一名排队等候接水的同学 k 马上接替 j 同学的位置开始接水。这个换人的过程是瞬间完成的,且没有任何水的浪费。即 j 同学第 x 秒结束时完成接水,则 k 同学第 $x+1$ 秒立刻开始接水。若当前接水人数 n' 不足 m ,则只有 n' 个龙头供水,其他 $m-n'$ 个龙头关闭。

现在给出 n 名同学的接水量,按照上述接水规则,问:所有同学都接完水需要多少秒?

输入:

第 1 行 2 个整数 n 和 m , 用一个空格隔开, 分别表示接水人数和龙头个数;

第 2 行 n 个整数 $w_1, w_2, \dots, w_n$, 每两个整数之间用一个空格隔开, w_i 表示 i 号同学的接水量。

输出:

只有一行, 1 个整数, 表示接水所需的总时间。

样例输入:

样例 #1:

5 3

4 4 1 2 1

样例 #2:

8 4

23 71 87 32 70 93 80 76

样例输出:

样例 #1:

4

样例 #2:

163

提示:

输入输出样例 1 解释:

第 1 秒, 3 人接水。第 1 秒结束时, 1, 2, 3 号同学每人的已接水量为 1, 3 号同学接完水, 4 号同学接替 3 号同学开始接水。

第 2 秒, 3 人接水。第 2 秒结束时, 1, 2 号同学每人的已接水量为 2, 4 号同学的已接水量为 1。

第 3 秒, 3 人接水。第 3 秒结束时, 1, 2 号同学每人的已接水量为 3, 4 号同学的已接水量为 2。4 号同学接完水, 5 号同学接替 4 号同学开始接水。

第 4 秒, 3 人接水。第 4 秒结束时, 1, 2 号同学每人的已接水量为 4, 5 号同学的已接水量为 1。1, 2, 5 号同学接完水, 即所有人完成接水。

总接水时间为 4 秒。

【参考程序】

```
#include<stdio>
using namespace std;
int main()
{
    int a[10001], s[101], n, m, i, j, k, min, max = -100000; //max 为接水的时间
    scanf("%d%d", &n, &m);
    for (i = 1; i <= n; ++i) scanf("%d", &a[i]);
    for (i = 1; i <= m; ++i) s[i] = 0; //每个水龙头的节水时间都初始化为 0
    for (i = 1; i <= n; ++i)
    {
        min = 100000; //min 为最短的时间
        for (j = 1; j <= m; ++j)
            if (s[j] < min)
            {
                min = s[j]; k = j; //找出最小时间的序号
            }
        s[k] += a[i]; //让下一个人在当前用时最短的水龙头接水
    }
    for (i = 1; i <= m; ++i)
        if (s[i] > max) max = s[i]; //接水时间最多的水龙头就是接水的最少总用时
    printf("%d\n", max);
    return 0;
}
```

第七章 分治算法

1. 2011【2.4 基本算法之分治 2991】

已知长度最大为 200 位的正整数 n , 请求出 2011^n 的后四位。

输入:

第 1 行为一个正整数 $k(k \leq 200)$, 代表有 k 组数据, 接下来的 k 行, 每行都有一个正整数 n , n 的位数不大于 200。

输出:

每一个 n 的结果为一个整数, 占一行, 若不足 4 位, 去除高位多余的 0。

样例输入:

3
5
28
792

样例输出:

1051
81
5521

【参考程序】

```
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
using namespace std;
char n[210];
int k, Min=2147483647;
int main()
{
    scanf("%d", &k);
    for (int i=1; i<=k; i++) //有 k 组数据
    {
        scanf("%s", n);
        int lenc=strlen(n), B=0;
        for (int j=lenc-4; j<lenc; j++) if (j>=0) B=B*10+n[j]-'0';
        int C=2011;
        int j=1;
        do
        {
            if (j*2<=B) //类似快速幂
                j*=2, C=(C*C)%10000;
        } while (j*2<=B);
        for (; j<B; j++)
            C=(C*2011)%10000; //计算后四位
        printf("%d\n", C); //输出后四位
    }
    return 0;
}
```

2. 输出前 k 大的数【2.4 基本算法之分治 7617】

给定一个数组,统计前 k 大的数并且把这 k 个数从大到小输出。

输入:

第 1 行包含一个整数 n ($n < 100,000$),表示数组的大小;

第 2 行包含 n 个整数,表示数组的元素,整数之间以一个空格分开。每个整数的绝对值不超过 $100,000,000$ 。

第 3 行包含一个整数 k ($k < n$)。

输出:

从大到小输出前 k 大的数,每个数占一行。

样例输入:

```
10
4 5 6 9 8 7 1 2 3 0
5
```

样例输出:

```
9
8
7
6
5
```

【参考程序】

```
#include<iostream>
using namespace std;
int js(int,int);
int a[100000];
int main()
{
    int n,k;
    cin>>n;
    for (int i=1;i<=n;++i) cin>>a[i];
    js(1,n); //从大到小排序
    cin>>k;
    for (int i=1;i<=k;++i)
        cout<<a[i]<<endl; //前 k 个数即为前 k 大的数
    return 0;
}

int js(int l,int r) //快速排序,二分法(可以用 sort()函数代替)
{
    int i,j,mid;
    i=l;
    j=r;
    mid=a[(l+r)/2]; //将当前序列在中间位置的数定义为分隔数
    do
    {
        while (a[i]>mid) i++;
        while (a[j]<mid) j--;
        if (i<=j)
        {
```

```

        int t;
        t=a[i];a[i]=a[j];a[j]=t;
        i++;
        j--;
        //继续找
    }
} while (i<=j);
if (l<j) js(l,j); //若未到两个数的边界,则递归搜索左右区间
if (r>i) js(i,r);
return 0;
}

```

3. 区间合并【2.4 基本算法之分治 7620】

给定 n 个闭区间 $[a_i, b_i]$, 其中 $i=1, 2, \dots, n$ 。任意两个相邻或相交的闭区间可以合并为一个闭区间。例如, $[1, 2]$ 和 $[2, 3]$ 可以合并为 $[1, 3]$, $[1, 3]$ 和 $[2, 4]$ 可以合并为 $[1, 4]$, 但是 $[1, 2]$ 和 $[3, 4]$ 不可以合并。

我们的任务是判断这些区间是否可以最终合并为一个闭区间, 如果可以, 将这个闭区间输出, 否则输出 no。

输入:

第 1 行为一个整数 $n(3 \leq n \leq 50,000)$, 表示输入区间的数量。

之后 n 行, 在第 i 行上 ($1 \leq i \leq n$), 为两个整数 a_i 和 b_i , 整数之间用一个空格分隔, 表示区间 $[a_i, b_i]$ ($1 \leq a_i \leq b_i \leq 10,000$)。

输出:

一行, 如果这些区间最终可以合并为一个闭区间, 输出这个闭区间的左右边界, 用单个空格隔开; 否则输出 no。

样例输入:

样例输出:

```

5                1 10
5 6
1 5
10 10
6 9
8 10

```

【参考程序】

```

#include<iostream>
using namespace std;
int js(int,int);
int a[50001],b[50001];
int main()
{
    int n;
    cin>>n;
    for (int i=1;i<=n;++i) cin>>a[i]>>b[i];
    js(1,n); //快排
    for (int i=1;i<=n;++i)
    if (b[i]<a[i+1]) //当第 i 个区间的结尾小于下一个区间的开头,则无法合并
    {
        cout<<" no "<<endl; //无法合并,输出"no"
        return 0; //结束程序
    }
}

```

```

else if (b[i]>b[i+1]) b[i+1]=b[i]; //当达成条件可合并时,合并闭区间
cout<<a[1]<<" "<<b[n]<<endl;
return 0;
}
int js(int l,int r)
{
    int i,j,mid;
    i=l;
    j=r;
    mid=a[(l+r)/2]; //将当前序列在中间位置的数定义为分隔数
    do
    {
        while (a[i]<mid) i++; //在左半部份寻找比中间数大的数
        while (a[j]>mid) j--; //在右半部份寻找比中间数大的数
        if (i<=j)
        {
            int t;
            t=a[i];a[i]=a[j];a[j]=t;
            t=b[i];b[i]=b[j];b[j]=t;
            i++;j--;
        }
    }while (i<=j);
    if (l<j) js(l,j); //若未到两个数的边界,则递归搜索左右区间
    if (r>i) js(i,r);
    return 0;
}

```

4. 求排列的逆序数【2.4 基本算法之分治 7622】

在 Internet 上的搜索引擎经常需要对信息进行比较,比如可以通过某个人对一些事物的排名来估计他(或她)对各种不同信息的兴趣,从而实现个性化的服务。

对于不同的排名结果可以用逆序来评价它们之间的差异。考虑 $1, 2, \dots, n$ 的排列 $i_1, i_2, \dots, i_n$, 如果其中存在 j, k , 满足 $j < k$ 且 $i_j > i_k$, 那么就称 (i_j, i_k) 是这个排列的一个逆序。

一个排列含有逆序的个数称为这个排列的逆序数。例如排列 263451 含有 8 个逆序 $(2, 1), (6, 3), (6, 4), (6, 5), (6, 1), (3, 1), (4, 1), (5, 1)$, 因此该排列的逆序数就是 8。显然, 由 $1, 2, \dots, n$ 构成的所有 $n!$ 个排列中, 最小的逆序数是 0, 对应的排列就是 $1, 2, \dots, n$; 最大的逆序数是 $n(n-1)/2$, 对应的排列就是 $n, (n-1), \dots, 2, 1$ 。逆序数越大的排列与原始排列的差异度就越大。

现给定 $1, 2, \dots, n$ 的一个排列, 求它的逆序数。

输入:

第 1 行是一个整数 n , 表示该排列有 $n(n \leq 100,000)$ 个数;

第 2 行是 n 个不同的正整数, 之间以空格隔开, 表示该排列。

输出:

该排列的逆序数。

样例输入:

```

6
2 6 3 4 5 1

```

样例输出:

```

8

```

【参考程序】

```

#include<iostream>
using namespace std;
void mer(long long,long long);
long long a[999999],r[999999],ans=0;
int main()
{
    long long n,i;
    cin>>n;
    for(i=1;i<=n;i++)
        cin>>a[i];
    mer(1,n);           //进行归并排序
    cout<<ans<<endl;
    return 0;
}

void mer(long long s,long long t)    //归并排序
{
    long long m,i,j,k;
    if(s==t) return;               //递归边界
    m=(s+t)/2;
    mer(s,m);                       //递归搜索左右区间
    mer(m+1,t);
    i=s; j=m+1; k=s;
    while(i<=m&& j<=t)              //两个子序列从小到大合并,直到有一列结束
    {
        if(a[i]<=a[j])               //选择较小的一个入新队列
        { r[k]=a[i];i++;k++; }
        else
        {
            ans=ans+j-k;              //记录逆序对的个数
            r[k]=a[j];j++;k++;
        }
    }
    while(i<=m)
    {
        r[k]=a[i];
        i++; k++;
    }
    while(j<=t)
    {
        r[k]=a[j];
        j++; k++;
    }
    for(i=s;i<=t;i++) a[i]=r[i];    //把合并后的有序数据重新放回 a 数组
}

```

5. 一元三次方程求解【2.4 基本算法之分治 7891】

有形如： $ax^3 + bx^2 + cx + d = 0$ 这样的 一个一元三次方程。

给出该方程中各项的系数(a,b,c,d 均为实数)，并约定该方程存在三个不同实根(根的范围在 $-100 \sim 100$ 之间)，且根与根之差的绝对值 ≥ 1 。要求由小到大依次在同一行输出这三个实根(根与根之间留有空格)，并精确到小数点后 2 位。

输入：

一行，包含四个实数 a,b,c,d,相邻两个数之间用单个空格隔开。

输出：

一行，包含三个实数，为该方程的三个实根，按从小到大顺序排列，相邻两个数之间用单个空格隔开，精确到小数点后 2 位。

样例输入：

1.0 -5.0 -4.0 20.0

样例输出：

-2.00 2.00 5.00

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
int x;
double a,b,c,d,x1,x2,xx,f(double x);
int main()
{
    scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
    for(x=-100;x<=100;x++) //枚举法
    {
        x1=x;x2=x+1; //根的区间
        if(f(x1)==0)printf("%.2lf",x1); //如果 x1 正好输出
        else if(f(x1)*f(x2)<0)
        {
            while(x2-x1>=0.001) //提高精度
            {
                xx=(x2+x1)/2;
                if(f(x1)*f(xx)<=0)x2=xx; //右指针往左
                else x1=xx; //左指针往右
            }
            printf("%.2lf",x1);
        }
    }
    return 0;
}
double f(double x)
{
    return x*x*x+a*x*x+b*x+c+d; //返回这个一元三次方程
}
```

6. 统计数字【2.4 基本算法之分治 7909】Noip2007 提高组第1题

某次科研调查时得到了 n 个自然数,每个数均不超过 $1,500,000,000(1.5 \times 10^9)$ 。已知不相同的数不超过 $10,000$ 个,现在需要统计这些自然数各自出现的次数,并按照自然数从小到大的顺序输出统计结果。

输入:

第1行是整数 n ,表示自然数的个数;

第 $2 \sim n+1$ 每行一个自然数。

注意:

40%的数据满足: $1 \leq n \leq 1000$;

80%的数据满足: $1 \leq n \leq 50,000$;

100%的数据满足: $1 \leq n \leq 200,000$,每个数均不超过 $1,500,000,000(1.5 \times 10^9)$ 。

输出:

包含 m 行(m 为 n 个自然数中不相同数的个数),按照自然数从小到大的顺序输出。每行输出两个整数,分别是自然数和该数出现的次数,其间用一个空格隔开。

样例输入:

```
8
2
4
2
4
5
100
2
100
```

样例输出:

```
2 3
4 2
5 1
100 2
```

【参考程序】

```
#include<iostream>
#include<algorithm>
#include<cstdio>
using namespace std;
int a[1000001],n,js=1,tot=1;
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    sort(a+1,a+n+1);           //从小到大排序(快速排序)
    while(tot<=n)
    {
        int s=a[tot],k;
        k=tot;
        while(s==a[tot])
            tot++;
        //统计相同的数
```



```

printf("%d ",a[tot-1]);
printf("%d\n",tot-k); //k 为未增值的 tot 的值,相减为 a[tot-1]出现的次数
}
return 0;
}

```

7. 查找最接近的元素【1.11 编程基础之二分查找 01】

在一个非降序列中,查找与给定值最接近的元素。

输入:

第 1 行包含一个整数 $n(1 \leq n \leq 100,000)$,为非降序列长度;

第 2 行包含 n 个整数,为非降序列各元素。所有元素的大小均在 $0 \sim 1,000,000,000$ 之间。

第 3 行包含一个整数 $m(1 \leq m \leq 10,000)$,为要询问的给定值个数。

接下来 m 行,每行一个整数,为要询问最接近元素的给定值。所有给定值的大小均在 $0 \sim 1,000,000,000$ 之间。

输出:

m 行,每行一个整数,为最接近相应给定值的元素值,保持输入顺序。若有多个值满足条件,输出最小的一个。

样例输入:

```

3
2 5 8
2
10
5

```

样例输出:

```

8
5

```

【参考程序】

```

#include<cstdio>
#include<cmath>
using namespace std;
int n,m,x,l,r,mid,a[100001];
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i) scanf("%d",&a[i]);
    scanf("%d",&m);
    while(m-->0) //有 m 组数据
    {
        scanf("%d",&x),l=1,r=n;
        while(l<r-1) //二分查找
        {
            mid=(l+r)/2; //将当前序列在中间位置的数定义为分隔数
            if(a[mid]>x) r=mid;
            else l=mid;
        }
        if(abs(a[l]-x)<=abs(a[r]-x))
            printf("%d\n",a[l]); //找到最接近的输出
        else printf("%d\n",a[r]);
    }
    return 0;
}

```

8. 二分法求函数的零点【1.11 编程基础之二分查找 02】

有函数： $f(x) = x^5 - 15 * x^4 + 85 * x^3 - 225 * x^2 + 274 * x - 121$

已知 $f(1.5) > 0$, $f(2.4) < 0$ 且方程 $f(x) = 0$ 在区间 $[1.5, 2.4]$ 有且只有一个根, 请用二分法求出该根。

输入:

无。

输出:

该方程在区间 $[1.5, 2.4]$ 中的根。要求四舍五入到小数点后 6 位。

样例输入:

无

样例输出:

不提供

【参考程序】

```
#include <stdio>
using namespace std;
double f(double x) //模拟式子
{
    return x * x * x * x * x - 15.0 * x * x * x * x + 85.0 * x * x * x - 225.0 * x * x + 274.0 * x - 121.0;
}
int main()
{
    double l = 1.5, r = 2.4;
    while(l < r - 0.000001) //二分查找
    {
        double m = (l + r) / 2.0; //将当前序列在中间位置的数定义为分隔数
        if(f(m) > 0.0) l = m; //前半查找
        if(f(m) < 0.0) r = m; //后半查找
    }
    if(f(l) == 0.0) printf("%.6lf", l);
    else printf("%.6lf", r);
    return 0;
}
```

9. 网线主管【1.11 编程基础之二分查找 04】

仙境的居民们决定举办一场程序设计区域赛。裁判委员会完全由自愿组成, 他们承诺要组织一次史上最公正的比赛。他们决定将选手的电脑用星形拓扑结构连接在一起, 即将它们全部连到一个单一的中心服务器。为了组织这个完全公正的比赛, 裁判委员会主席提出要将所有选手的电脑等距离地围绕在服务器周围放置。

为购买网线, 裁判委员会联系了当地的一个网络解决方案提供商, 要求能够提供一定数量的等长网线。裁判委员会希望网线越长越好, 这样选手们之间的距离可以尽可能远一些。

该公司的网线主管承接了这个任务。他知道库存中每条网线的长度(精确到厘米), 并且只要告诉他所需的网线长度(精确到厘米), 他都能够完成对网线的切割工作。但是, 这次, 所需的网线长度并不知道, 这让网线主管不知所措。

你需要编写一个程序, 帮助网线主管确定一个最长的网线长度, 并且按此长度对库存中的网线进行切割, 能够得到指定数量的网线。

输入:

第 1 行包含两个整数 N 和 K , 以单个空格隔开。 $N(1 \leq N \leq 10,000)$ 是库存中的网

线数, $K(1 \leq K \leq 10,000)$ 是需要的网线数量。

接下来 N 行, 每行一个数, 为库存中每条网线的长度(单位: 米)。所有网线的长度至少 1m, 至多 100km。输入中的所有长度都精确到厘米, 即保留到小数点后两位。

输出:

网线主管能够从库存的网线中切出指定数量的网线的最长长度(单位: 米)。必须精确到厘米, 即保留到小数点后两位。

若无法得到长度至少为 1cm 的指定数量的网线, 则必须输出 "0.00" (不包含引号)。

样例输入:

```
4 11
8.02
7.43
4.57
5.39
```

样例输出:

```
2.00
```

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cmath>
using namespace std;
double n,k,a[100086],maxx=-2147483647;
double x;
int main()
{
    scanf("%lf%lf",&n,&k);
    for (int i=1;i<=n;i++)
    {
        scanf("%lf",&x);
        a[i]=x;
        maxx=max(maxx,a[i]);           //找出最长的网线
    }
    double l=0,r=maxx;
    while (l+0.001<r)                   //二分查找
    {
        double Mid=(l+r)/2.0;          //将当前序列在中间位置的数定义为分隔数
        int ans=0;
        for (int i=1;i<=n;i++) ans+=(a[i]/Mid);
        if (ans>=k) l=Mid;
        else r=Mid;                     //处理左右端点
    }
    int ans=0,an=0;
    for (int i=1;i<=n;i++) an+=(a[i]/r),ans+=(a[i]/l);
    if (an>=k) l=r;
    printf("%.2lf",floor(l*100)/100.0); //处理输出(保留两位)
    return 0;
}
```

10. 月度开销【1.11 编程基础之二分查找 06】

农夫约翰是一个精明的会计师。他意识到自己可能没有足够的钱来维持农场的运转了。他计算出并记录下了接下来 N ($1 \leq N \leq 100,000$) 天里每天需要的开销。

约翰打算为连续的 M ($1 \leq M \leq N$) 个财政周期创建预算案,他把一个财政周期命名为 fajo 月。每个 fajo 月包含一天或连续的多天,每天被恰好包含在一个 fajo 月里。

约翰的目标是合理安排每个 fajo 月包含的天数,使得开销最多的 fajo 月的开销尽可能少。

输入:

第 1 行包含两个整数 N, M , 用单个空格隔开;

接下来 N 行, 每行包含一个 $1 \sim 10,000$ 之间的整数, 按顺序给出接下来 N 天里每天的开销。

输出:

一个整数, 即最大月度开销的最小值。

样例输入:

```
7 5
100
400
300
100
500
101
400
```

样例输出:

```
500
```

提示:

若约翰将前两天作为一个月, 第三、第四两天作为一个月, 最后三天各作为一个月, 则最大月度开销为 500。其他任何分配方案都会比这个值更大。

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include<cmath>
#include<string>
using namespace std;
int n,m,a[100086],l=0;
int r;
//假设最大月开销为 mid,统计需要分成多少个月,然后看月的个数是否太多或太少
int pd(int X)
{
    int money=0,fajo_month=1;
    for (int i=1;i<=n;i++)
    {
        if (money+a[i]<=X) money+=a[i]; //把第 i 天归入到当前第 fajo_month 月
        else //可以独立成一个月
        {
            money=a[i];
        }
    }
}
```

```

        fajo_month++;           //开始一个新的月
    }
}
return (fajo_month>m);
}
int main()
{
    scanf("%d%d",&n,&m);
    r=1000000001;
    for (int i=1;i<=n;i++)
        scanf("%d",&a[i]),l=max(l,a[i]);
    while (l<r-1)
    {
        int Mid=(l+r)/2;
        if (pd(Mid)) l=Mid;
        else r=Mid;
    }
    if (! pd(l)) printf("%d",l);
    else printf("%d",r);
}

```

11. 和为给定数【1.11 编程基础之二分查找 07】

给出若干个整数,询问其中是否有一对数的和等于给定的数。

输入:

第 1 行是整数 n ($0 < n \leq 100,000$),表示有 n 个整数;
 第 2 行是 n 个整数。整数的范围是在 0 到 10^8 之间;
 第 3 行是一个整数 m ($0 \leq m \leq 2^{30}$),表示需要得到的和。

输出:

若存在和为 m 的数对,输出两个整数,小的在前,大的在后,中间用单个空格隔开。若有多个数对满足条件,选择数对中较小的数更小的。若找不到符合要求的数对,输出一行 No。

样例输入:

```

4
2 5 1 4
6

```

样例输出:

```

1 5

```

【参考程序】

```

#include<cstdio>
#include<algorithm>
using namespace std;
int n,a[100001],m;
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)scanf("%d",&a[i]);
    sort(a+1,a+n+1);           //从小到大排序
    scanf("%d",&m);
}

```

```

for(int i=1;i<n; ++i)
{
    int l=i+1,r=n;
    while(l<r-1)
    {
        int mi=(l+r)/2;           //将当前序列在中间位置的数定义为分隔数
        if(a[mi]<m-a[i]) l=mi;
        else r=mi;
    }
    if(a[r]+a[i]==m)
    {
        printf("%d %d ",a[i],a[r]);    //输出符合条件的数对
        return 0;
    }
    if(a[l]+a[i]==m)
    {
        printf("%d %d ",a[i],a[l]);    //输出符合条件的数对
        return 0;
    }
}
printf(" No ");           //若没有则输出 No
return 0;
}

```

12. 不重复地输出数【1.11 编程基础之二分查找 08】

输入 n 个数,从小到大将它们输出,重复的数只输出一次。保证不同的数不超过 500 个。

输入:

第 1 行是一个整数 $n(1 \leq n \leq 100,000)$;

之后 n 行,每行一个整数。整数大小在 `int` 范围内。

输出:

一行,从小到大不重复地输出这些数,相邻两个数之间用单个空格隔开。

样例输入:

```

5
2 4 4 5 1

```

样例输出:

```

1 2 4 5

```

【参考程序】

```

#include<cstdio>
using namespace std;
int a[100001],n;
void qs(int l,int r)           //快速排序
{
    int i=l,j=r,m=a[(l+r)/2]; //将当前序列在中间位置的数定义为分隔数
    do
    {
        while(a[i]<m) i++;      //在左半部份寻找比中间数大的数
        while(a[j]>m) j--;      //在右半部份寻找比中间数小的数
        if(i<=j)
            //若找到一组与排序目标不一致的数对,则交换它们

```

```

    {
        int t=a[i]; a[i]=a[j]; a[j]=t;
        i++; j--; //继续找
    }
} while(i<=j);
if(l<j) qs(l,j); //若未到两个数的边界,则递归搜索左右区间
if(i<r) qs(i,r);
}
int main()
{
    scanf("%d",&n);
    for(int i=1;i<=n;++i)
        scanf("%d",&a[i]);    qs(1,n);
    int i=1;
    while(i<=n)
    {
        while(a[i]==a[i+1]) i++; //跳到不重复的数
        printf("%d ",a[i++]); //处理输出
    }
    return 0;
}

```

13. 膨胀的木棍【1.11 编程基础之二分查找 09】

当长度为 L 的一根细木棍的温度升高 n 度,它会膨胀到新的长度 $L'=(1+n * C) * L$,其中 C 是热膨胀系数。

当一根细木棍被嵌在两堵墙之间被加热,它将膨胀形成弓形的弧,而这个弓形的弦恰好是未加热前木棍的原始位置。

你的任务是计算木棍中心的偏移距离。

输入:

三个非负实数:木棍初始长度(单位:毫米),温度变化(单位:度),以及材料的热膨胀系数。

保证木棍不会膨胀到超过原始长度的 1.5 倍。

输出:

木棍中心的偏移距离(单位:毫米),保留到小数点后第三位。

样例输入:

1000 100 0.0001

样例输出:

61.329

【参考程序】

```

#include <cmath>
#include <cstdio>
using namespace std;
double N,C,L;
int main()
{
    scanf("%lf%lf%lf",&L,&N,&C);
    double minv=0,maxv=acos(-1.0),midv;
    double L2=(1+N * C) * L; //膨胀后新的长度

```

```
while(maxv-minv>1e-12)           //二分查找
{
    midv=(minv+maxv)/2;           //将当前序列在中间位置的数定义为分隔数
    if(2 * L2/L>midv/sin(midv/2))
        minv=midv;
    else
        maxv=midv;
}
printf("%.3lf\n",L2/midv*(1-cos(midv/2)));
return 0;
}
```

14. 河中跳房子【1.11 编程基础之二分查找 10】

每年奶牛们都要举办各种特殊版本的跳房子比赛,包括在河里从一个岩石跳到另一个岩石。这项激动人心的活动在一条长长的笔直河道中进行,在起点和离起点 L 远 ($1 \leq L \leq 1,000,000,000$) 的终点处均有一个岩石。在起点和终点之间,有 N ($0 \leq N \leq 50,000$) 个岩石,每个岩石与起点的距离分别为 D_i ($0 < D_i < L$)。

在比赛过程中,奶牛轮流从起点出发,尝试到达终点,每一步只能从一个岩石跳到另一个岩石。当然,实力不济的奶牛是没有办法完成目标的。

农夫约翰为他的奶牛们感到自豪并且年年都观看这项比赛。但随着时间的推移,看着其他农夫的胆小奶牛们在相距很近的岩石之间缓慢前行,他感到非常厌烦。他计划移走一些岩石,使得从起点到终点的过程中,最短的跳跃距离最长。他可以移走除起点和终点外的至多 M ($0 \leq M \leq N$) 个岩石。

请帮助约翰确定移走这些岩石后,最长可能的最短跳跃距离是多少。

输入:

第 1 行包含三个整数 L, N, M , 相邻两个整数之间用单个空格隔开。

接下来 N 行, 每行一个整数, 表示每个岩石与起点的距离。岩石按与起点距离从近到远给出, 且不会有两个岩石出现在同一个位置。

输出:

一个整数, 最长可能的最短跳跃距离。

样例输入:

```
25 5 2
2
11
14
17
21
```

样例输出:

```
4
```

提示:

在移除位于 2 和 14 的两个岩石之后, 最短跳跃距离为 4 (17~21 或 21~25)。

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#define Maxx 51000
using namespace std;
int i,j,k,n,m;
```



```
int l,r,mid;
int Ans,L;
int a[Maxx],b[Maxx];
int work(int mid)
{
    k=0;
    for(int i=0;i<=n;++i) b[i]=a[i];
    for(int i=1;i<=n;++i)
    {
        if(b[i]-b[i-1]<mid)
        {
            b[i]=b[i-1];
            k++;
        }
        if(k>m) return k;
    }
    return k;
}

int main()
{
    scanf("%d%d%d",&L,&n,&m);
    for(i=1;i<=n;++i)
        scanf("%d",&a[i]);
    sort(a+1,a+1+n);           //从小到大排序
    n++;
    a[0]=0; a[n]=L;
    l=0; r=L;
    while(l+1<r)
    {
        mid=(l+r)/2;           //将当前序列在中间位置的数定义为分隔数
        if(work(mid)>m) r=mid;
        else l=mid;
    }
    if (work(r)<=m) cout<<r;   //如果移走的石头个数符合要求,输出
    else cout<<l;
    return 0;
}
```

第八章 广度优先搜索算法

1. Dungeon Master[2.5 基本算法之搜索 1253]

题目大意:这题是一个三维的迷宫题目,其中用“.”表示空地,“#”表示障碍物,“S”表示起点,“E”表示终点,求从起点到终点的最小移动次数,解法和二维的类似,只是在行动时除了东南西北移动外还多了上下。可以上下左右前后移动,每次都只能移到相邻的空位,每次需要花费一分钟,求从起点到终点最少要多久。

样例输入:

```

3 4 5
S . . . .
. # # # .
. # # . .
# # # . #

# # # # #
# # # # #
# # . # #
# # . . .

# # # # #
# # # # #
# . # # #
# # # # E

1 3 3
S # #
# E #
# # #

0 0 0
    
```

样例输出:

```

Escaped in 11 minute(s).
Trapped!
    
```

【题目分析】

对于题目给出数据的含义就是输入 l, r, c , 分别代表迷宫有 l 层, 每层长宽分别是 c, r 。数据 1 可以这样移动:

```

(1,1,1)→(1,1,2)→(1,1,3)→(1,1,4)→(1,1,5)→(1,2,5)→
(1,3,5)→(1,3,4)→(1,4,4)→(2,4,4)→(2,4,5)→(3,4,5)
    
```

共 11 步就可以到达终点, 数据 2 明显不能到达, 则输出 Trapped。

这题用 BFS 解, 每次去队首元素, 如果是终点则输出结果移动的步数, 否则, 从该点开始分别向东南西北上下移动(如果可以走的话)并继续搜, 如果到队列为空还没搜到解法, 则说明无解。

【参考程序】

```

#include<iostream>                                     //BFS
    
```

```

using namespace std;
int xx[6]={-1,1,0,0,0,0}, //三维的六个方向
    yy[6]={0,0,-1,1,0,0},
    zz[6]={0,0,0,0,-1,1};
int h[27001][5]; //队列
bool a[31][31][31];
int main()
{
    int l,r,c,t,w,x,y,z,fx,fy,fz;
    char ch;
    bool s;
    while (cin>>l>>r>>c) //循环输入 l,r,c
    {
        if (l==0&&.r==0&&.c==0)
            return 0; //若满足条件则终止
        for (x=1;x<=l;++x)
        for (y=1;y<=r;++y)
        for (z=1;z<=c;++z) //三重循环表示三维空间
        {
            cin>>ch;
            if (ch=='S') //寻找并记录起点
            {
                a[x][y][z]=false; //起点无法重复走
                h[1][1]=x; //存下起点 3 个坐标
                h[1][2]=y;
                h[1][3]=z;
                h[1][4]=0; //表示起点到起点步数为 0
            }
            else if (ch=='E') //寻找并记录终点
            {
                a[x][y][z]=true; //标记为可走
                fx=x; //存下终点 3 个坐标
                fy=y;
                fz=z;
            }
            else if (ch=='#') a[x][y][z]=false; //‘#’记为不可走
            else a[x][y][z]=true;
        }
        t=0; w=1;
        s=false;
        do
        {
            t++; //头指针+1
            for (int i=0;i<6;++i)
            {
                x=h[t][1]+xx[i];
                y=h[t][2]+yy[i];

```

```

z=h[t][3]+zz[i];
if (x>0&& x<=l&& y>0&& y<=r&& z>0&& z<=c&& a[x][y][z])
{
    //判断是否越界及是否走过
    a[x][y][z]=false;
    w++; //尾指针+1
    h[w][1]=x;
    h[w][2]=y;
    h[w][3]=z;
    h[w][4]=h[t][4]+1; //到该点的步数即为到上一个点的步数加 1
    if (x==fx&& y==fy&& z==fz) //判断是否到达终点
    {
        cout<<" Escaped in "<<h[w][4]<<" minute(s). "<<endl;
        //第一次找到即为最短路径
        s=true; //表示能够达到终点
        break;
    }
}
}
} while (t<w&& s==false);
if (s==false)
    cout<<" Trapped! " <<endl; //无法到达则直接输出
}
return 0;
}

```

2. Lake Counting【2.5 基本算法之搜索 1388】

题意:有一块 $N \times M$ 的土地,雨后积起了水,有水标记为“W”,干燥为“.”。八连通的积水被认为是连接在一起的。请求出院子里共有多少水洼?

样例输入:

```

10 12
W . . . . . W W .
. W W W . . . . W W W
. . . . W W . . W W .
. . . . . . . W W .
. . . . . . . W . .
. . W . . . . W . .
. W . W . . . W W .
W . W . W . . . . W .
. W . W . . . . W .
. . W . . . . . W .

```

样例输出:

3

【广搜程序】

```

#include<iostream> //本题水洼等同于细胞
using namespace std;
void js(int,int);
int xx[8]={-1,1,0,0,-1,-1,1,1}, //表示 8 个方向
    yy[8]={0,0,-1,1,-1,1,-1,1};

```

```

int n,m,ans=0,h[10001][3];           //h 数组表示队列
char a[101][101];
int main()
{
    cin>>n>>m;
    for (int i=1;i<=n;++i)
        for (int j=1;j<=m;++j) cin>>a[i][j];
    for (int i=1;i<=n;++i)
        for (int j=1;j<=m;++j)
            if (a[i][j]=='W') js(i,j);           //遇到细胞,搜索周围的细胞
    cout<<ans<<endl;
    return 0;
}

void js (int p,int q)
{
    int x,y,t=0,w=1;
    ans++;
    h[1][1]=p; h[1][2]=q;           //第一个细胞入队
    a[p][q]='.';                     //覆盖第一个细胞,避免回到起点
    do
    {
        t++;                         //头指针+1
        for (int i=0;i<8;++i)
        {
            x=h[t][1]+xx[i];         //表示搜到的下一个点的坐标
            y=h[t][2]+yy[i];
            if (x>0&& x<=n&& y>0&& y<=m&& a[x][y]=='W')
            {
                a[x][y]='.';         //判断当前点是否过界及是否是细胞
                w++;                 //覆盖细胞,避免重复搜索
                h[w][1]=x;           //尾指针+1
                h[w][2]=y;           //将搜到的细胞记录横纵坐标并入队
            }
        }
    } while (t<w);                   //直到队空为止
}

```

【深搜程序】

```

#include <iostream>
using namespace std;
int N, M, res;
char field[100][100];
void dfs(int x, int y)
{
    field[x][y]='.';                 //标记为干燥,即为搜索过
    for (int dx=-1; dx<=1; dx++)    //循环遍历 8 个移动方向
    {

```

```

for (int dy = -1; dy <= 1; dy++)
{
    int nx = x + dx, ny = y + dy;           // 移动到(nx,ny)位置
    if (0 <= nx && nx < N && 0 <= ny && ny < M && field[nx][ny] == 'W')
        // 判断是否在院子内,并且是否有积水
        dfs(nx, ny);                       // 搜索水洼的下一个积水
}
}

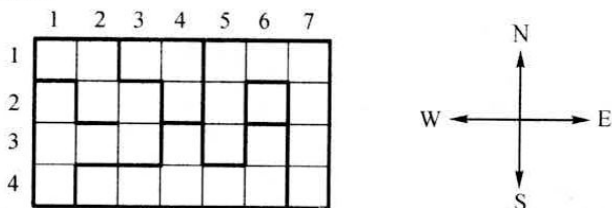
int main()
{
    while (cin >> N >> M)
    {
        int res = 0;
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                cin >> field[i][j];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < M; j++)
                if (field[i][j] == 'W')
                    // 从有'W'的地方开始搜索
                    {
                        dfs(i, j);           // 开始搜索周围的'W'
                        res++;               // res 记录水洼数
                    }
        cout << res << endl;
    }
    return 0;
}

```

3. The Castle【2.5 基本算法之搜索 166 和 1817】

问题描述:

一座城堡被分成 $m \times n$ ($m \leq 50, n \leq 50$) 个方块, 每个方块可有 0~4 堵墙(0 表示无墙)。下面表示出了建筑平面图:



图中的加粗黑线代表墙。几个连通的方块组成房间, 房间与房间之间一定是用黑线(墙)隔开的。

现在要求你编一个程序, 解决以下 2 个问题:

(1) 该城堡中有多少个房间?

(2) 最大的房间有多大?

输入格式:

平面图用一个数字表示一个方块(第 1 个房间用二进制 1011 表示, 0 表示无东墙, 用十进制 11 表示)。

●第1行一个整数 $m(m \leq 50)$, 表示房子南北方向的长度;

第2行一个整数 $n(n \leq 50)$, 表示房子东西方向的长度。

●后面的 m 行, 每行有 n 个整数, 每个整数都表示平面图对应位置的方块的特征。每个方块中墙的特征由数字 $P(0 \leq P \leq 15)$ 来描述。数字 P 是下面的可能取的数字之和:

1(西墙 west)

2(北墙 north)

4(东墙 east)

8(南墙 south)

室内的墙被定义两次: 例如方块 $(1, 1)$ 中的南墙也被位于其南面的方块 $(2, 1)$ 定义了一次。

●建筑中至少有两个房间。

输出格式:

第1行: 一个整数, 表示房间总数;

第2行: 一个整数, 表示最大房间的面积(方块数);

输入样例:

```
4
7
11 6 11 6 3 10 6
7 9 6 13 5 15 5
1 10 12 7 13 7 5
13 11 10 8 10 12 13
```

输出样例:

```
5
9
```

【广搜程序】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
int dx[4]={-1,1,0,0}, //四个方向
    dy[4]={0,0,-1,1};
int a[51][51][4],q[5001][3];
int p,i,j,n,m,tot,maxx=-1;
bool b[51][51];
void bfs(int x,int y)
{
    memset(q,0,sizeof(q));
    tot++; //tot 记录房间数
    int tail=1,head=0,xx,yy;
    q[1][1]=x; q[1][2]=y;
    while (head<tail) //直到队空为止
    {
        head++; //头指针+1
        for (int i=0; i<=3; ++i) //向周围四个方向搜索
        {
            xx=q[head][1]+dx[i]; //移动位置
            yy=q[head][2]+dy[i];
```

```

if (xx>0 && xx<=m && yy>0 && yy<=n)
if (b[xx][yy]==true && a[q[head][1]][q[head][2]][i]==0)
{
    tail++; //判断是否越界以及有无墙以及是否走过
    b[xx][yy]=false; //尾指针+1
    q[tail][1]=xx; //表示已经搜过
    q[tail][2]=yy; //将新搜到的点入队
}
}
}
maxx=max(tail-1,maxx); //每个房间的方块数即为尾指针-1,取最大值
}
int main()
{
    memset(b,true,sizeof(b));
    scanf("%d%d",&m,&n);
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
        {
            scanf("%d",&p);
            if (p>=8){p-=8;a[i][j][1]=1;} //南边有墙
            if (p>=4){p-=4;a[i][j][3]=1;} //东边有墙
            if (p>=2){p-=2;a[i][j][0]=1;} //北边有墙
            if (p>=1){p-=1;a[i][j][2]=1;} //西边有墙
            //从大到小判断是否有墙
        }
    for (i=1; i<=m; ++i)
        for (j=1; j<=n; ++j)
            if (b[i][j]) bfs(i,j); //对未搜索过的方块进行搜索
    printf("%d\n%d\n",tot,maxx);
}

```

【深搜程序】

```

#include<cstdio> //DFS
#include<cstring>
using namespace std;
int s,n,m,sum,maxn=-1,a[101][101];
bool b[101][101];
void dfs(int i,int j)
{
    if (i<1||j<1||i>m||j>n||b[i][j]) return;
    b[i][j]=1; sum++; //判断是否越界
    if (a[i][j]>=8) //模拟每个房间的4面墙
        a[i][j]=a[i][j]-8; else dfs(i+1,j); //判断南边是否有墙
    if (a[i][j]>=4)
        a[i][j]=a[i][j]-4; else dfs(i,j+1); //判断东边是否有墙
    if (a[i][j]>=2)

```



```

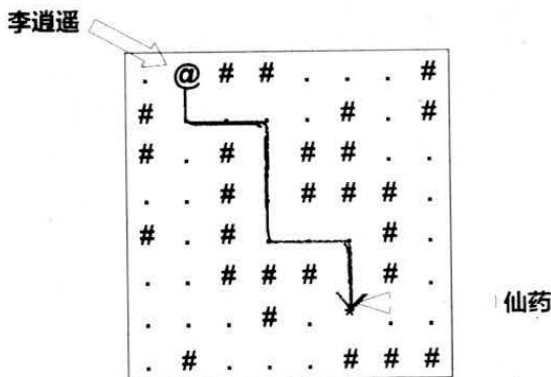
    a[i][j]=a[i][j]-2; else dfs(i-1,j);           //判断北边是否有墙
    if (a[i][j]==0) dfs(i,j-1);                   //判断西边是否有墙
}
int main()
{
    scanf("%d%d",&m,&n);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            scanf("%d",&a[i][j]);
    for (int i=1; i<=m; i++)
        for (int j=1; j<=n; j++)
            if (! b[i][j])                         //判断该方格是否搜索过
            {
                sum=0;                             //sum 记录房间的大小
                s++;
                dfs(i,j);                           //搜索周围的方格
                if (sum>maxn) maxn=sum;              //取城堡中最大房间
            }
    printf("%d\n%d\n",s,maxn);
    return 0;
}

```

4. 仙岛求药【2.5 基本算法之搜索 2727】

少年李逍遥的婶婶病了,王小虎介绍他去一趟仙灵岛,向仙女姐姐要仙丹救婶婶。叛逆但孝顺的李逍遥闯进了仙灵岛,克服了千险万难来到岛的中心,发现仙药摆在了迷阵的深处。迷阵由 $M \times N$ 个方格组成,有的方格内有可以瞬秒李逍遥的怪物,而有的方格内则是安全的。现在李逍遥想尽快找到仙药,显然他应避开有怪物的方格,并经过最少的方格,而且那里会有神秘人物等待着他。现在要求你来帮助他实现这个目标。

下图显示了一个迷阵的样例及李逍遥找到仙药的路线。



输入:

有多组测试数据,每组测试数据以两个非零整数 M 和 N 开始,两者均不大于 20。 M 表示迷阵行数, N 表示迷阵列数。接下来有 M 行,每行包含 N 个字符,不同字符分别代表不同含义:

- (1)“@”:少年李逍遥所在的位置;
- (2)“.”:可以安全通行的方格;
- (3)“#”:有怪物的方格;

(4)“*”：仙药所在位置。

当在一行中读入的是两个零时，表示输入结束。

输出：

对于每组测试数据，分别输出一行，该行包含李逍遥找到仙药需要穿过的最少的方格数目(计数包括初始位置的方块)。如果他不可能找到仙药，则输出-1。

样例输入：

```

8 8
. @ # # . . . #
# . . . . # . #
# . # . # # . .
. . # . # # # .
# . # . . . # .
. . # # # . # .
. . . # . * . .
. # . . . # # #
6 5
. * . # .
. # . . .
. . # # .
. . . . .
. # . . .
. . . . @
9 6
. # . . # .
. # . * . #
. # # # # .
. . # . . .
. . # . . .
. . # . . .
. . # . . .
# . @ . # #
. # . . # .
0 0
    
```

样例输出：

10

8

-1

【参考程序】

```

#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int n,m,qx,qy,l[10001],x,y;
char a[101][101];
bool b[101][101];
int dx[4]={-1,1,0,0},dy[4]={0,0,-1,1}; //四个方向
int p[10001][3]; //队列
    
```

```

int bfs()
{
    memset(l,0,sizeof(l));
    memset(p,0,sizeof(p));
    memset(b,0,sizeof(b));           //清空所有上一个数据中的所有数组
    int t=0,w=1;
    p[1][1]=qx; p[1][2]=qy;         //将起点入队
    while (t<w)
    {
        t++;                         //头指针加 1
        for (int i=0; i<4; i++)      //搜索四个方向
        {
            x=p[t][1]+dx[i];         //移动位置
            y=p[t][2]+dy[i];
            if (x>=0&& x<m&& y>=0&& y<n) //判断是否越界
                if (a[x][y]!='#' && b[x][y]==0) //判断是否可走和是否重复走过
                {
                    w++;              //尾指针加 1
                    p[w][1]=x;        //将新搜到的点入队
                    p[w][2]=y;
                    l[w]=l[t]+1;
                    b[x][y]=1;        //记录,避免重复走
                    if (a[x][y]=='*')  //判断是否到达终点
                    {
                        printf("%d\n",l[w]);
                        return 0;
                    }
                }
        }
    }
    printf("-1\n");
}

int main()
{
    while (cin>>m>>n)               //循环输入 n,m
        if (n!=0&&m!=0)              //判断输入是否结束
        {
            for (int i=0; i<m; i++)
            {
                scanf("%s",a[i]);
                for (int j=0; j<n; j++)
                {
                    if (a[i][j]=='@') //寻找并记录起点
                    {
                        qx=i;
                        qy=j;
                    }
                }
            }
        }
    }
}

```

```

    bfs();
}
return 0;
}

```

5. 走迷宫【2.5 基本算法之搜索 2753】

一个迷宫由 R 行 C 列格子组成,有的格子里有障碍物,不能走;有的格子是空地,可以走。

给定一个迷宫,求从左上角走到右下角最少需要走多少步(数据保证一定能走到)。只能在水平方向或垂直方向走,不能斜着走。

输入:

第 1 行是两个整数, R 和 $C(1 \leq R, C \leq 40)$,代表迷宫的长和宽。

接下来是 R 行,每行 C 个字符,代表整个迷宫。

空地格子用“.”表示,有障碍物的格子用“#”表示。

迷宫左上角和右下角都是“.”。

输出:

从左上角走到右下角至少要经过的步数(即至少要经过的空地格子数)。计算步数要包括起点和终点。

样例输入:

```

5 5
. . # # #
# . . . .
# . # . #
# . # . #
# . # . .

```

样例输出:

9

【广搜程序】

```

#include<iostream>
using namespace std;
int xx[4]={-1,1,0,0},yy[4]={0,0,-1,1}; //四个方向
int h[1001][4];
bool a[41][41];
int main()
{
    int r,c,x,y,i,j,t=0,w=1;
    char k;
    cin>>r>>c;
    for (i=1;i<=r;++i)
        for (j=1;j<=c;++j)
        {
            cin>>k;
            if (k=='.') a[i][j]=true; //判断是否能走
            else a[i][j]=false;
        }
    h[1][1]=1;h[1][2]=1;h[1][3]=1; //左上角的起点即为(1,1)
    a[1][1]=false;
    while (t<w) //直到队空为止

```

```

{
    t++; //头指针+1
    for (i=0;i<4;i++) //搜索当前点的上、下、左、右四个方向
    {
        x=h[t][1]+xx[i]; //移动位置
        y=h[t][2]+yy[i];
        if (x>0&& x<=r&& y>0&& y<=c&& a[x][y])
        { //判断此点是否越界、是否搜索过及是否能走
            w++; //尾指针+1
            h[w][1]=x; //新搜到的点入队
            h[w][2]=y;
            h[w][3]=h[t][3]+1;
            a[x][y]=false; //走过的点不能重复走
            if (x==r&& y==c) //判断是否到达终点,右下角即为(r,c)
            {
                cout<<h[w][3]<<endl;
                return 0; //第一次找到终点即为最短路径
            }
        }
    }
}
//直至队空为止
}

```

【深搜程序】

```

#include<iostream>
#include<cstring>
using namespace std;
int r,c,mi=2147483647;
int xx[4]={-1,1,0,0},yy[4]={0,0,-1,1};
bool b[101][101]={0};
char a;
void dfs(int x,int y,int n)
{
    if(x==r && y==c && n<mi) mi=n; //到达终点(r,c)后取最短路径
    for(int i=0;i<4;i++)
    {
        if(! b[x+xx[i]][y+yy[i]]) //判断下一步是否能走以及是否走过
        {
            b[x][y]=1; //走过的点不能再走
            dfs(x+xx[i],y+yy[i],n+1); //继续搜索下一个点
            b[x][y]=0; //回溯法,退一步重新搜索
        }
    }
}

int main()
{
    memset(b,1,sizeof(b)); //先全部标记为不可走
    cin>>r>>c;
}

```

```

for(int i=1;i<=r;i++)
{
    for(int j=1;j<=c;j++)
    {
        cin>>a;
        if(a=='.') b[i][j]=0;           //标记为可走
    }
}
dfs(1,1,0);                           //从左上角开始搜索,路径初始为 0
cout<<mi+1;
return 0;
}

```

6. 抓住那头牛【2.5 基本算法之搜索 2971】

农夫知道一头牛的位置,想要抓住它。农夫和牛都位于数轴上,农夫起始位于点 $N(0 < N \leq 100,000)$,牛位于点 $K(0 \leq K \leq 100,000)$ 。农夫有两种移动方式:

- (1)从 X 移动到 $X-1$ 或 $X+1$,每次移动花费一分钟;
- (2)从 X 移动到 $2 * X$,每次移动花费一分钟。

假设牛没有意识到农夫的行动,站在原地不动。农夫最少要花多少时间才能抓住牛?

输入:

两个整数, N 和 K 。

输出:

一个整数,农夫抓到牛所要花费的最小分钟数。

样例输入:

5 17

样例输出:

4

【参考程序】

```

#include<iostream>                       //BFS
using namespace std;
int h[1000001][3];
int n,k,x;
bool v[1000001];
int main()
{
    cin>>n>>k;
    h[1][1]=n; v[n]=1;
    int head=0,tail=1;
    do
    {
        head++;                           //头指针+1
        for (int i=1; i<=3; i++)          //三种移动方式
        {
            x=h[head][1];
            if (i==1) x++;                 //向右移动一位
            if (i==2) x--;                 //向左移动一位
            if (i==3) x*=2;                //移动到 2 * x 的位置
            if (x>=0 && x<=100000)         //判断是否越界

```

```

        if (!v[x] || h[x][2] > h[h[head][1]][2] + 1)
        {
            //是否走过或是否步数最少
            tail++; //尾指针+1
            h[tail][1] = x; //将新搜到的点入队
            v[x] = 1;
            h[x][2] = h[h[head][1]][2] + 1; //h[x][2]记录步数
        }
    }
    while (head < tail);
    cout << h[k][2] << endl; //坐标点 k 上存储的步数即为最短路径
    return 0;
}

```

7. 走出迷宫【2.5 基本算法之搜索 6264】

当你站在一个迷宫里的时候,往往会被错综复杂的道路弄得失去方向感,如果你能得到迷宫地图,事情就会变得非常简单。

假设你已经得到了一个 $n * m$ 的迷宫的图纸,请你找出从起点到出口的最短路。

输入:

第 1 行是两个整数 n 和 m ($1 \leq n, m \leq 100$), 表示迷宫的行数和列数。

接下来 n 行, 每行一个长为 m 的字符串, 表示整个迷宫的布局。字符“.”表示空地, “#”表示墙, “S”表示起点, “T”表示出口。

输出:

从起点到出口最少需要走的步数。

样例输入:

```

3 3
S # T
. # .
. . .

```

样例输出:

```
6
```

【广搜程序】

```

#include <cstring>
#include <iostream>
using namespace std;
int h[10001][4], xx[4] = {-1, 1, 0, 0}, yy[4] = {0, 0, -1, 1}; //队列及四个方向
bool a[101][101];
int main()
{
    int n, m, fx, fy, x, y, t = 0, w = 1;
    char c;
    memset(a, true, sizeof(a)); //a 数组全部记为可走
    cin >> n >> m;
    for (int i = 1; i <= n; ++i)
        for (int j = 1; j <= m; ++j)
        {
            cin >> c;

```

```

if (c=='S') //查找并记录起点坐标
{
    h[1][1]=i; //将起点入队
    h[1][2]=j;
    h[1][3]=0; //起点到起点步数为 0
    a[i][j]=false; //起点不可以重复走
}
else if (c=='T') //查找并记录终点坐标
{
    fx=i;
    fy=j;
}
else if (c=='#') a[i][j]=false; //“#”记录为不可走
}
do
{
    t++;
    for (int i=0;i<4;++i) //模拟 4 个移动方向
    {
        x=h[t][1]+xx[i]; //移动位置
        y=h[t][2]+yy[i];
        if (x>0&& x<=n&& y>0&& y<=m&& a[x][y])
        { //判断当前点是否、越界是否走过及是否
可走
            a[x][y]=false; //走过的点不能再走
            w++; //尾指针加 1
            h[w][1]=x; //新搜到的点入队
            h[w][2]=y;
            h[w][3]=h[t][3]+1;
            if (x==fx&& y==fy) //判断是否找到终点
            {
                cout<<h[w][3]<<endl; //第一次找到即为最短路径
                return 0;
            }
        }
    }
} while(t<w);
}

```

【深搜程序】

```

#include<iostream>
#include<cstdio>
#include<algorithm>

#include<cmath>
using namespace std;
int n,m,x,y,f[1001][1001];
char a[1001][1001];

```



```

int dfs(int b)
{
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
    {
        if(f[i][j]==b)                //更新步数为 b 周围四个方向的点
        {
            if(a[i-1][j]=='.') f[i-1][j]=f[i][j]+1;
            if(a[i+1][j]=='.') f[i+1][j]=f[i][j]+1;
            if(a[i][j-1]=='.') f[i][j-1]=f[i][j]+1;
            if(a[i][j+1]=='.') f[i][j+1]=f[i][j]+1;
            //判断四个方向是否可走,也可用 for 循环处理
        }
    }
    if(f[x][y]!=0) cout<<f[x][y]-1;
    else dfs(b+1);
}

int main()
{
    cin>>n>>m;
    for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
    {
        cin>>a[i][j];
        if(a[i][j]=='T')                //遇到终点记录并标记为可走
        {
            a[i][j]='.'; x=i;y=j;
        }
        if(a[i][j]=='S') f[i][j]=1;    //遇到起点初始为 1 种走法
    }
    dfs(1);                            //从步数为 1 的周围点开始搜索
}

```

8. 迷宫问题【2.5 基本算法之搜索 7084】

定义一个二维数组：

$$\text{int maze}[5][5] = \begin{pmatrix} 0, & 1, & 0, & 0, & 0, \\ 0, & 1, & 0, & 1, & 0, \\ 0, & 0, & 0, & 0, & 0, \\ 0, & 1, & 1, & 1, & 0, \\ 0, & 0, & 0, & 1, & 0, \end{pmatrix};$$

它表示一个迷宫,其中的 1 表示墙壁,0 表示可以走的路,只能横着走或竖着走,不能斜着走,要求编程找出从左上角到右下角的最短路线。

输入：

一个 5×5 的二维数组,表示一个迷宫。数据保证有唯一解。

输出：

左上角到右下角的最短路径,格式如样例所示。

样例输入：

0 1 0 0 0

```
0 1 0 1 0
0 0 0 0 0
0 1 1 1 0
0 0 0 1 0
```

样例输出:

```
(0, 0)
(1, 0)
(2, 0)
(2, 1)
(2, 2)
(2, 3)
(2, 4)
(3, 4)
(4, 4)
```

【广搜程序】

```
#include<iostream>
using namespace std;
int h[37][3], b[26];
int xx[4]={-1,1,0,0},yy[4]={0,0,-1,1};    //四个方向
bool a[6][6];
int main()
{
    int t=0,w=1,x,y,k=0;
    for (int i=0;i<5;++i)
        for (int j=0;j<5;++j) cin>>a[i][j];
    h[1][1]=0,h[1][2]=0;
    a[1][1]=1;
    do
    {
        t++;                                //头指针+1
        for (int i=0;i<4;++i)              //模拟搜索的四个方向
        {
            x=h[t][1]+xx[i];
            y=h[t][2]+yy[i];
            if (x>=0&& x<5&& y>=0&& y<5&& a[x][y]==0)
            {
                a[x][y]=1;                  //判断是否越界及是否走过
                w++;                          //避免走重复的点
                h[w][0]=t;                    //尾指针+1
                h[w][1]=x;                    //记录上一步
                h[w][2]=y;
                if (x==4&& y==4) break;      //到达终点,跳出循环
            }
        }
    } while (t<w);                          //直至队空为止
    while (w>=1)
    {
```

```

    k++;
    b[k]=w; //记录坐标
    w=h[w][0]; //循环替换前驱
}
for (int i=k;i>=1;--i)
    cout<<"("<<h[b[i]][1]<<","<<h[b[i]][2]<<")"<<endl;
return 0;
}

```

【深搜程序】

```

#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
using namespace std;
int a[1001][1001],
    b[1001][1001],
    dx[]={1,-1,0,0}, //四个方向
    dy[]={0,0,1,-1};
int tx,ty,tot=0;
void print(int a,int aa)
{
    if(a==0&&aa==0)
    {
        cout<<"(0,0)"<<endl;
        return;
    } // (0,0)时特殊处理输出,递归结束
    print(b[a][aa]/5,b[a][aa]%5); //返回 i,j,递归处理前驱
    cout<<"("<<a<<","<<aa<<")"<<endl; //每次输出一坐标
}
void dfs(int a[][1001],int i,int j)
{
    if(a[i][j]==99999999) return; //如果不可走则直接返回
    for(int k=0;k<=3;k++) //向周围四个方向搜索
    {
        tx=i+dx[k]; ty=j+dy[k]; //移动位置
        if(a[tx][ty]!=99999999&&0<=tx&&tx<=4&&0<=ty&&ty<=4) //判断是否可走,是否越界
        {
            tot=a[i][j]+1;
            if(tot<a[tx][ty]||a[tx][ty]==0)
            {
                a[tx][ty]=tot; //tot 表示到当前的点的最短距离
                b[tx][ty]=i*5+j; //利用 i*5+j 特殊记录 i,j 前驱
                dfs(a,tx,ty);
            }
        }
    }
}
return ;

```

```

}
int main()
{
    for(int i=0;i<5;i++)
        for(int j=0;j<5;j++)
        {
            cin>>a[i][j];
            if(a[i][j]==1) a[i][j]=99999999; //遇到障碍标记为不可走
        }
    dfs(a,0,0); //开始搜索
    print(4,4); //开始输出到(4,4)的路径
}

```

9. 献给阿尔吉侬的花束【2.5 基本算法之搜索 7218】

阿尔吉侬是一只聪明又慵懒的小白鼠，它最擅长的就是走各种各样的迷宫。今天它要挑战一个非常大的迷宫，研究员们为了鼓励阿尔吉侬尽快到达终点，就在终点放了一块阿尔吉侬最喜欢的奶酪。现在研究员们想知道，如果阿尔吉侬足够聪明，它最少需要多少时间就能吃到奶酪。

迷宫用一个 $R \times C$ 的字符矩阵来表示。字符“S”表示阿尔吉侬所在的位置，字符“E”表示奶酪所在的位置，字符“#”表示墙壁，字符“.”表示可以通行。阿尔吉侬在 1 个单位时间内可以从当前的位置走到它上下左右四个方向上的任意一个位置，但不能走出地图边界。

输入：

第 1 行是一个正整数 $T(1 \leq T \leq 10)$ ，表示一共有 T 组数据。

每 1 组数据的第 1 行包含了两个用空格分开的正整数 R 和 $C(2 \leq R, C \leq 200)$ ，表示地图是一个 $R \times C$ 的矩阵。

接下来的 R 行描述了地图的具体内容，每 1 行包含了 C 个字符。字符含义如题目描述中所述，且保证有且仅有一个 S 和 E。

输出：

对于每组数据，输出阿尔吉侬吃到奶酪的最少单位时间。若阿尔吉侬无法吃到奶酪，则输出“oop!”(只输出引号里面的内容，不输出引号)。每组数据的输出结果占一行。

样例输入：

```

3
3 4
. S . .
# # # .
. . E .
3 4
. S . .
. E . .
. . . .
3 4
. S . .
# # # #
. . E .

```

样例输出：

```

5
1
oop!

```

【参考程序】

```

#include<cstring>
#include<iostream>
using namespace std;
int h[40001][4],xx[4]={-1,1,0,0},yy[4]={0,0,-1,1}; //队列及四个方向
bool a[201][201];
int main()
{
    int k,r,c,fx,fy,x,y,t,w;
    char cs;
    bool s;
    cin>>k;
    for (int l=1;l<=k;++l)
    {
        cin>>r>>c;
        memset(a,false,sizeof(a)); //先全部初始为不可走
        for (int i=1;i<=r;++i)
        for (int j=1;j<=c;++j)
        {
            cin>>cs;
            if (cs=='S') //寻找并记录起点
            {
                h[1][1]=i; //将起点入队
                h[1][2]=j;
                h[1][3]=0; //起点到起点步数为0
            }
            else if (cs=='E') //寻找并记录终点
            {
                a[i][j]=true; //终点记录为可走
                fx=i;
                fy=j;
            }
            else if (cs=='.') a[i][j]=true; //“.”记录为可走
        }
        t=0;
        w=1;
        s=false;
        do
        {
            t++; //头指针+1
            for (int i=0;i<4;++i)
            {
                x=h[t][1]+xx[i]; //移动位置
                y=h[t][2]+yy[i];
                if (x>0&& x<=r&& y>0&& y<=c&& a[x][y])
                { //判断是否过界、是否走过
                    a[x][y]=false;
                }
            }
        } while (!s);
    }
}

```

```

w++; //尾指针+1
h[w][1]=x; //将新的点入队
h[w][2]=y;
h[w][3]=h[t][3]+1; //记录到该点的步数
if (x==fx&& y==fy) //到达终点,输出
{
    cout<<h[w][3]<<endl; //第一次找到即为最短时间
    s=true;
}
}
}
} while (t<w&& s==false);
if (s==false)
    cout<<" oop! "<<endl; //无法到达则输出“oop!”
}
return 0;
}

```

10. Knight Moves【2.5 基本算法之搜索 917】

题意:输入 n 代表有个 $n \times n$ 的棋盘,输入开始位置的坐标和结束位置的坐标,问:一个骑士朝棋盘的八个方向走马字步,从开始坐标到结束坐标可以经过多少步。

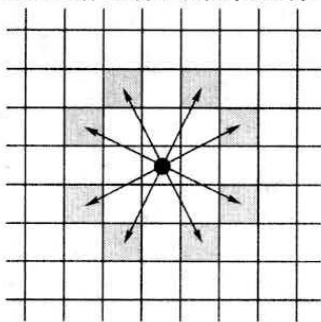


Figure 1: Possible knight moves on the board.

输入:

首先输入一个 n , 表示测试样例的个数。

每个测试样例有三行。第 1 行是棋盘的大小 $l(4 \leq l \leq 300)$, 第 2 行和第 3 行分别表示马的起始位置和目标位置 $(0 \sim l-1)$ 。

输出:

马移动的最小步数, 起始位置和目标位置相同时输出 0。

样例输入:

```

3
8
0 0
7 0
100
0 0
30 50
10
1 1
1 1

```

样例输出:

5
28
0

【参考程序】

```
#include<stdio>
#include<cmath>
#include<cstring>
using namespace std;
int fx[8]={1,1,2,2,-1,-1,-2,-2};
int fy[8]={2,-2,1,-1,2,-2,1,-1}; //八个方向
int N,n,sx,sy,ex,ey,h[5000001][3]; //h 为队列
bool visited[1001][1001];
int work(int sx,int sy,int ex,int ey)
{
    if((sx==ex)&&(sy==ey))return 0; //若起点与终点重复则直接退出
    memset(visited,0,sizeof(visited)); //标记为全部未走过
    int t=0,w=1;
    h[1][0]=sx,h[1][1]=sy,h[1][2]=0; //将起点入队
    do
    {
        t++; //头指针+1
        int x=h[t][0],y=h[t][1];
        for(int i=0;i<8;++i) //模拟搜索八个方向
        {
            int X=x+fx[i],Y=y+fy[i]; //移动位置
            if((X>=0)&&(X<n)&&(Y>=0)&&(Y<n)&&!visited[X][Y]) //判断是否越界、是否走过
            {
                w++;
                h[w][0]=X; //将新搜到的点入队
                h[w][1]=Y;
                h[w][2]=h[t][2]+1; //记录新搜到的点的步数
                visited[X][Y]=1; //标记已走过
                if((X==ex)&&(Y==ey))return h[w][2]; //到达终点则返回
            }
        }
    }while(t<w); //直到队列为空
}

int main()
{
    scanf("%d",&N); //n 组数据
    for(int i=1;i<=N;++i)
    {
        scanf("%d%d%d%d%d",&n,&sx,&sy,&ex,&ey); //棋盘大小 n * n
        printf("%d\n",work(sx,sy,ex,ey));
    }
}
```

第九章 动态规划

第一节 动态规划的基本模型

1. 最长上升子序列【2.6 基本算法之动态规划 1759】

一个数的序列 b_i , 当 $b_1 < b_2 < \dots < b_s$ 的时候, 我们称这个序列是上升的。对于给定的一个序列 $(a_1, a_2, \dots, a_N)$, 我们可以得到一些上升的子序列 $(a_{i_1}, a_{i_2}, \dots, a_{i_K})$, 这里 $1 \leq i_1 < i_2 < \dots < i_K \leq N$ 。比如, 对于序列 $(1, 7, 3, 5, 9, 4, 8)$, 有它的一些上升子序列, 如 $(1, 7), (3, 4, 8)$ 等等。这些子序列中最长的长度是 4, 比如子序列 $(1, 3, 5, 8)$ 。

你的任务就是对于给定的序列, 求出最长上升子序列的长度。

输入:

第 1 行是序列的长度 $N (1 \leq N \leq 1000)$;

第 2 行给出序列中的 N 个整数, 这些整数的取值范围都在 $0 \sim 10000$ 。

输出:

最长上升子序列的长度。

样例输入:

```
7
1 7 3 5 9 4 8
```

样例输出:

```
4
```

【参考程序】

```
#include<iostream> //本题即为求最长不下降序列
#include<algorithm>
using namespace std;
int i=0,j,k,l,b[1001][1001],s,n;
int main()
{
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>s; //输入序列的初始值
        b[i][1]=s; //b[i][1]记录当前数 s
        b[i][2]=1; //b[i][2]记录序列长度,初始为 1
        b[i][3]=0; //b[i][3]记录序列的下一个数,默认为 0
    }
    for(i=n-1;i>=1;i--) //求最长不下降序列
    {
        l=k=0;
        for(j=i+1;j<=n;j++)
            if(b[j][1]>b[i][1]&& b[j][2]>l)
                //判断后面的数是否大于当前数且是否是更优的解(序列长度更长)
```



```

{
    l=b[j][2];           //l 存储当前的最优解
    k=j;                 //k 暂时存储当前下一个满足序列的数的序号
}
if(l>0)                 //判断是否找到最优解
{
    b[i][2]=l+1;         //当前序列长度+1
    b[i][3]=k;           //表示该序列下一个数为 k
}
}
k=1;
for(j=1;j<=n;j++)      //求最长不下降序列的起始位置
    if(b[j][2]>b[k][2]) k=j; //寻找下一个存储序列长度更长的数
                                //序列长度不断递增,所得数列为最长不下降子序列
cout<<b[k][2]<<endl;    //输出这个数
return 0;
}

```

2. 最大子矩阵【2.6 基本算法之动态规划 1768】

已知矩阵的大小定义为矩阵中所有元素的和。给定一个矩阵,你的任务是找到最大的非空(大小至少是 1×1)子矩阵。

比如,如下 4×4 的矩阵。

```

0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2

```

它的最大子矩阵是

```

9 2
-4 1
-1 8

```

这个子矩阵的大小是 15。

输入:

一个 $N \times N$ 的矩阵。输入的第 1 行给出 N ($0 < N \leq 100$)。再后面的若干行中,依次(首先从左到右给出第 1 行的 N 个整数,再从左到右给出第 2 行的 N 个整数……)给出矩阵中的 N^2 个整数,整数之间由空白字符分隔(空格或者空行)。已知矩阵中整数的范围都在 $[-127, 127]$ 。

输出:

输出最大子矩阵的大小。

样例输入:

```

4
0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2

```

样例输出:

15

【参考程序 1】

```
#include<iostream>
#include<cstdio>
using namespace std;
int n,i,j,fir,las,sum,a[100][100],f[100][100],ans=-1000000;
//ans 储存最大子矩阵的值

int main()
{
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    ans=a[1][1];
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            f[i][j]=f[i][j-1]+a[i][j]; //f[i][j]存储第 i 行前 j 个数字的和
    for(fir=1;fir<=n;fir++) //开头
        for(las=fir;las<=n;las++) //结尾
        {
            sum=0;
            for(i=1;i<=n;i++)
            {
                sum+=f[i][las]-f[i][fir-1]; //储存第 i 行 fir 列到 las 列之和
                ans=max(ans,sum); //ans 求最大值
                sum=max(sum,0); //如果 sum 小于 0,则清零,避免子矩阵和为负数
            }
        }
    printf("%d",ans); //输出储存在 ans 的最大值
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstring>
#include<cmath>
#include<cstdio>
using namespace std;
int a[1001][101];
int b[10001],sum,maxx,tot,m,n;
int hhhh(int a[],int n);
int main()
{
    cin>>n;
    for (int i=1;i<=n;i++)
        for (int j=1;j<=n;j++)
            cin>>a[i][j];
    for (int i=1;i<=n;i++)
        for (int j=i;j<=n;j++)
```

```

{
    memset(b,0,sizeof(b));           //每处理一次数组清 0
    for (int q=0;q<=n;q++)
        for (int w=i;w<=j;w++)
            b[q]+=a[w][q];           //纵向列之和
    sum=hhhh(b,n);
    if(maxx<sum) maxx=sum;           //求最大的一个子矩阵和
}
cout<<maxx<<endl;
return 0;
}
int hhhh(int a[],int n)
{
    tot=0,m=0;
    for (int i=1;i<=n;i++)
    {
        tot+=a[i];                   //模拟加上子矩阵的每一行
        if (tot<0) tot=0;             //如果是负数则归 0,避免子矩阵和为负数
        if (tot>m) m=tot;             //取子矩阵最大的和
    }
    return m;                         //返回最大值 m
}

```

3. 登山【2.6 基本算法之动态规划 1996】

五一到了,ACM 队组织大家去登山观光,队员们发现山上有 N 个景点,并且决定按照顺序来浏览这些景点,即每次所浏览景点的编号都要大于前一个浏览景点的编号。同时队员们还有另一个登山习惯,就是不连续浏览海拔相同的两个景点,并且一旦开始下山,就不再向上走了。队员们希望在满足上面条件的同时,尽可能多地浏览景点,你能帮他们找出最多可能浏览的景点数么?

输入:

第 1 行: N ($2 \leq N \leq 1000$) 个景点数;

第 2 行: N 个整数,表示每个景点的海拔。

输出:

最多能浏览的景点数。

样例输入:

8

186 186 150 200 160 130 197 220

样例输出:

4

【参考程序】

```

#include<cstdio>
using namespace std;
int a[10001],b[10001],c[10001],i,j,n,max=0;
int main()
{
    scanf("%d",&n);                 //读入景点数
    for (i=1;i<=n;i++)               //读每个景点的高度
        scanf("%d",&a[i]);

```

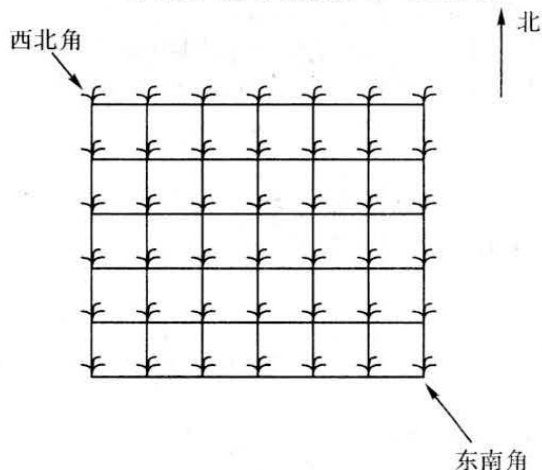
```

for (i=1; i<=n; ++i)          //按照从左往右的顺序计算 b 序列的最大值
{
    b[i]=1;
    for (j=1; j<=i-1; ++j)
        if (a[i]>a[j] && b[j]+1>b[i]) b[i]=b[j]+1;
    //找出编号从小到大、海拔从低到高的序列长度最大值
}
for (i=n; i>=1; --i)          //按照从右往左的顺序计算 c 序列的最大值
{
    c[i]=1;
    for (j=i+1; j<=n; ++j)
        if (a[j]<a[i] && c[j]+1>c[i]) c[i]=c[j]+1;
    //找出编号从大到小、海拔从低到高的序列长度最大值
}
for (i=1; i<=n; ++i)          //计算最多能浏览的景点数(其中一个景点被重复计算)
    if (b[i]+c[i]>max) max=b[i]+c[i];
printf("%d\n", max-1);        //因为其中一个景点被重复计算
return 0;
}

```

4. 摘花生【2.6 基本算法之动态规划 2728】

Hello Kitty 想摘点花生送给她喜欢的米老鼠。她来到一片有网格状道路的矩形花生地(如图),从西北角进去,东南角出来。地里每个道路的交叉点上都种着一株花生苗,上面有若干颗花生,经过一株花生苗就能摘走它上面所有的花生。Hello Kitty 只能向东或向南走,不能向西或向北走。问:Hello Kitty 最多能够摘到多少颗花生。



输入:

第 1 行是一个整数 $T(1 \leq T \leq 100)$, 代表一共有多少组数据。

接下来是 T 组数据。

每组数据的第 1 行是两个整数, 分别代表花生苗的行数 R 和列数 $C(1 \leq R, C \leq 100)$ 。

每组数据的接下来 R 行数据, 从北向南依次描述每行花生苗的情况。每行数据有 C 个整数, 按从西向东的顺序描述了该行每株花生苗上的花生数目 $M(0 \leq M \leq 1000)$ 。

输出:

对每组输入数据, 输出一行, 内容为 Hello Kitty 能摘到得最多的花生颗数。

样例输入:

```
2
2 2
1 1
3 4
2 3
2 3 4
1 6 5
```

样例输出:

```
8
16
```

【参考程序 1】

```
#include<cstdio>
#include<iostream>
using namespace std;
int f[1001][1001],r,c,i,j;
int main()
{
    int t; scanf("%d",&t);
    while (t-->0) //循环输入多组数据
    {
        scanf("%d%d",&r,&c);
        for (i=1; i<=r; ++i)
            for (j=1; j<=c; ++j)
                scanf("%d",&f[i][j]);
        for (i=1; i<=r; ++i)
            for (j=1; j<=c; ++j)
                f[i][j] += max(f[i-1][j],f[i][j-1]);
        //判断左方的方格与上方的方格哪个存储的数较大,即逆推法
        //因为走到当前的点只有两种方法,要么从左往右走,要么从上往下走
        printf("%d\n",f[r][c]);
    }
    return 0;
}
```

【参考程序 2】

```
#include<cstring>
#include<iostream>
using namespace std;
int h[101][101];
int main()
{
    int t,n,m;
    cin>>t;
    for (int l=1;l<=t;++l)
    {
        memset(h,1001,sizeof(h));
        //h 数组初始默认为最大值。当判断边界元素时,直接选择输入数组内的元素
        cin>>n>>m;
        for (int i=1;i<=n;++i)
            for (int j=1;j<=m;++j) cin>>h[i][j];
    }
}
```

```

for (int i=n;i>=1;--i)
    for (int j=m;j>=1;--j)
        //顺推法,判断右方的方格与下方的方格哪个存储的数较大
        //因为走到当前的点只有两种方法,要么从左往右走,要么从上往下走
        if (i<n||j<m) //判断,防止在(n,m)点上累加
            if (h[i+1][j]>=h[i][j+1]) h[i][j]+=h[i+1][j];
            //累加较大值
            else h[i][j]+=h[i][j+1];
        cout<<h[1][1]<<endl; //输出储存在 h[1][1]的最大值
    }
return 0;
}

```

5. 最大上升子序列和【2.6 基本算法之动态规划 3532】

一个数的序列 b_i , 当 $b_1 < b_2 < \dots < b_S$ 的时候, 我们称这个序列是上升的。对于给定的一个序列 $(a_1, a_2, \dots, a_N)$, 我们可以得到一些上升的子序列 $(a_{i_1}, a_{i_2}, \dots, a_{i_K})$, 这里 $1 \leq i_1 < i_2 < \dots < i_K \leq N$ 。比如, 对于序列 $(1, 7, 3, 5, 9, 4, 8)$, 有它的一些上升子序列, 如 $(1, 7)$, $(3, 4, 8)$ 等等。这些子序列中序列和最大为 18, 为子序列 $(1, 3, 5, 9)$ 的和。

你的任务是对于给定的序列, 求出最大上升子序列和。注意: 最长的上升子序列的和不一定是最大的, 比如序列 $(100, 1, 2, 3)$ 的最大上升子序列和为 100, 而最长上升子序列为 $(1, 2, 3)$ 。

输入:

第 1 行是序列的长度 $N (1 \leq N \leq 1000)$;

第 2 行给出序列中的 N 个整数, 这些整数的取值范围都在 $0 \sim 10,000$ 内(可能重复)。

输出:

最大上升子序列和。

样例输入:

7

1 7 3 5 9 4 8

样例输出:

18

【参考程序 1】

```

#include<iostream> //最大上升子序列和,最长上升子序列稍加改变
using namespace std;
int h[1001][3];
int main()
{
    int n,max=0;
    cin>>n;
    for (int i=1;i<=n;++i)
    {
        cin>>h[i][1];
        h[i][2]=h[i][1]; //h[i][2]存储当前序列的下一个数,初始值赋为自身
    }
    for (int i=n-1;i>=1;--i) //从倒数第二个数开始往前搜索
    for (int j=i+1;j<=n;++j)
        if (h[i][1]<h[j][1]&&h[i][2]<h[j][2]+h[i][1]) h[i][2]=h[j][2]+h[i][1];
    //判断当前点是否与 h[i][1]呈上升子序列以及是否是最优解。如果是,则更新
}

```

```

for (int i=1;i<=n;i++)
    if (h[i][2]>max) max=h[i][2]; //寻找最大的上升子序列和
cout<<max<<endl; //输出储存在 max 的最大上升子序列和
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int n,a[100001],tot=-100000,b[100001];
int main()
{
    cin>>n;
    for(int i=1;i<=n;i++)
        cin>>a[i];
    for(int i=1;i<=n;i++)
    {
        b[i]=a[i]; //b[]存储当前最大上升子序列的和,初始值赋为本身
        for(int j=1;j<i;j++)
            if(a[i]>a[j]) //与之前读入的数比较,判断是否存在呈上升趋势
                b[i]=max(b[i],b[j]+a[i]); //判断较大值并记录
    }
    for(int i=1;i<=n;i++)
        if(b[i]>tot) tot=b[i]; //tot 求最大上升子序列和
    cout<<tot;
}

```

6. 怪盗基德的滑翔翼【2.6 基本算法之动态规划 4977】

怪盗基德是一个充满传奇色彩的怪盗,专门以珠宝为目标的超级盗窃犯。而他最为突出的地方,就是他每次都能逃脱中村警部的重重围堵,而这很大程度上多亏了他随身携带的便于操作的滑翔翼。

有一天,怪盗基德像往常一样偷走了一颗珍贵的钻石,不料却被柯南小朋友识破了伪装,而他的滑翔翼的动力装置也被柯南踢出的足球破坏了。不得已,怪盗基德只能操作受损的滑翔翼逃脱。

假设城市中一共有 N 幢建筑排成一条线,每幢建筑的高度各不相同。初始时,怪盗基德可以在任何一幢建筑的顶端。他可以选择一个方向逃跑,但是不能中途改变方向(因为中村警部会在后面追击)。因为滑翔翼动力装置受损,他只能往下滑行(即:只能从较高的建筑滑翔到较低的建筑)。他希望尽可能多地经过不同建筑的顶部,这样可以减缓下降时的冲击力,减少受伤的可能性。请问:他最多可以经过多少幢不同建筑的顶部(包含初始时的建筑)?

输入:

数据第 1 行是一个整数 $K(K \leq 100)$,代表有 K 组测试数据。

每组测试数据包含两行:第 1 行是一个整数 $N(N \leq 100)$,代表有 N 幢建筑;第 2 行包含

N 个不同的整数,每一个对应一幢建筑的高度 $h(0 < h < 10,000)$,按照建筑的排列顺序给出。

输出:

对于每一组测试数据,输出一行,包含一个整数,代表怪盗基德最多可以经过的建筑数量。

样例输入:

```
3
8
300 207 155 299 298 170 158 65
8
65 158 170 298 299 155 207 300
10
2 1 3 4 5 6 7 8 9 10
```

样例输出:

```
6
6
9
```

【参考程序】

//本题怪盗基德可往两个方向飞行,因此寻找两次最长上升序列

```
#include<iostream>
using namespace std;
int h[100][4],k,n,maxx;
int main()
{
    cin>>k;
    for (int l=1;l<=k;++l)
    {
        cin>>n;
        for (int i=1;i<=n;++i)
        {
            cin>>h[i][1];
            h[i][2]=h[i][3]=1;
            //h[i][2]存储从左往右最长上升子序列的长度,初始长度为 1
            //h[i][3]存储从右往左最长不上升子序列的长度,初始长度为 1
        }
        for (int i=n-1;i>=1;--i) //寻找从 h[i][1]出发,能经过的最多的建筑
        for (int j=i+1;j<=n;++j)
            if (h[i][1]<h[j][1]&&h[i][2]<=h[j][2]) h[i][2]=h[j][2]+1;
            //从左到右,求最长上升子序列
            else if (h[i][1]>h[j][1]&&h[i][3]<=h[j][3]) h[i][3]=h[j][3]+1;
            //从右到左,求最长不上升子序列
        maxx=0;
        for (int i=1;i<=n;++i) //寻找能够经过的最多的建筑
        {
            if (h[i][2]>maxx) maxx=h[i][2];
            if (h[i][3]>maxx) maxx=h[i][3];
        }
        cout<<maxx<<endl; //输出储存在 max 的最优值
    }
    return 0;
}
```


7. 最低通行费【2.6 基本算法之动态规划 7614】

一个商人穿过一个 $N \times N$ 的正方形的网格,去参加一个非常重要的商务活动。他要从网格的左上角进,右下角出。每穿越中间 1 个小方格,都要花费 1 个单位时间。商人必须在 $(2N-1)$ 个单位时间穿越出去。而在经过中间的每个小方格时,都需要缴纳一定的费用。

这个商人期望在规定时间内用最少费用穿越出去。请问至少需要多少费用?

注意:不能对角穿越各个小方格(只能向上下左右四个方向移动且不能离开网格)。

输入:

第 1 行是一个整数,表示正方形的宽度 N ($1 \leq N \leq 100$);

后面 N 行,每行 N 个不大于 100 的整数,为网格上每个小方格的费用。

输出:

至少需要的费用。

样例输入:

```
5
1   4   6   8  10
2   5   7  15  17
6   8   9  18  20
10  11  12  19  21
20  23  25  29  33
```

样例输出:

```
109
```

提示:

样例中,最小值为 $109 = 1 + 2 + 5 + 7 + 9 + 12 + 19 + 21 + 33$ 。

【参考程序 1】

```
#include <cstdio>
#include <iostream>
using namespace std;
int n, a[1001][1001], f[1001][1001], i, j;
int main()
{
    scanf("%d", &n);
    for (i=1; i<=n; ++i) //N * N 的正方形
        for (j=1; j<=n; ++j)
        {
            scanf("%d", &a[i][j]);
            f[0][i]=999999; f[j][0]=999999;
            //a 数组边界默认为最大值。当判断边界元素时,直接选择输入数组内的元素
        }
    f[1][1]=a[1][1]; //从左上角开始
    for (i=1; i<=n; ++i)
        for (j=1; j<=n; ++j)
            if (i==1 && j==1) f[i][j]=a[i][j];
            else f[i][j]=min(f[i-1][j], f[i][j-1])+a[i][j];
    //从上行或左列来选最小值, min 求出较小数
    printf("%d\n", f[n][n]); //输出最少费用
    return 0;
}
```

【参考程序 2】

```

#include<iostream>
#include<cstring>
using namespace std;
int main()
{
    int n,i,j,b[200][200],max;
    cin>>n;
    memset(b,122,sizeof(b));
    //b 数组默认为最大值。当判断边界元素时,直接选择输入数组内的元素
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            cin>>b[i][j];
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(i>1||j>1) //判断是否处于起点,避免在起点累加
                if(b[i][j-1]<=b[i-1][j]) //判断向右走还是向下走
                    b[i][j]=b[i][j]+b[i][j-1]; //向右走
                else b[i][j]=b[i][j]+b[i-1][j]; //向下走
    cout<<b[n][n]<<endl; //b[n][n]储存最优解
}

```

8. 三角形最佳路径问题【2.6 基本算法之动态规划 7625】

如下所示的由正整数数字构成的三角形：

```

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

```

从三角形的顶部到底部有很多条不同的路径。对于每条路径,把路径上面的数加起来可以得到一个和,和最大的路径称为最佳路径。你的任务就是求出最佳路径上的数字之和。

注意:路径上的每一步只能从一个数走到下一层上和它最近的下边(正下方)的数或者右边(右下方)的数。

输入:

第 1 行为三角形高度 $h(1 \leq h \leq 100)$,同时也是最底层边的数字的数目。

从第 2 行开始,每行为三角形相应行的数字,中间用空格分隔。

输出:

最佳路径的长度数值。

样例输入:

```

5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

```

样例输出:

30

【参考程序 1】

```

#include<cstdio>                                //逆推法
#include<iostream>
using namespace std;
int a[1001][1001];
int main()
{
    int n;
    scanf("%d",&n);
    for (int i=1; i<=n; i++)
        for (int j=1; j<=i; j++)
            scanf("%d",&a[i][j]);
    for (int i=n-1; i>=1; i--)                    //从倒数第二行逆推到第一行
        for (int j=1; j<=i; j++)
            a[i][j]+=max(a[i+1][j],a[i+1][j+1]);    //从下往上,累加较大值,即逆推法
    printf("%d",a[1][1]);                        //输出储存在 a[1][1]的最终答案,即最大值
    return 0;
}

```

【参考程序 2】

```

#include<cstdio>                                //顺推法
using namespace std;
int a[200][200]={0},n,i,j,max=0;
int main()
{
    scanf("%d",&n);
    for (i=1; i<=n; ++i)
        for (j=1; j<=i; ++j) scanf("%d",&a[i][j]);
    for (i=2; i<=n; ++i)                        //从上往下搜索
        for (j=1; j<=i; ++j)
            if (a[i-1][j-1]>a[i-1][j])            //从上往下,累加较大值,即顺推法
                a[i][j]=a[i-1][j-1];
            else a[i][j]=a[i-1][j];
    for (i=1; i<=n; ++i)
        if (a[n][i]>max) max=a[n][i];            //将最后一行中各个路径进行比较,求最大值
    printf("%d\n",max);
    return 0;
}

```

9. 拦截导弹[2.6 基本算法之动态规划 8780]

某国为了防御敌国的导弹袭击,发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷:虽然它的第一发炮弹能够到达任意的高度,但是以后每一发炮弹都不能高于前一发的高度。某天,雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段,所以只有一套系统,因此有可能不能拦截所有的导弹。

输入导弹依次飞来的高度(雷达给出的高度数据是不大于 30,000 的正整数),计算这套系统最多能拦截多少导弹。

输入:

第 1 行是一个整数 N(不超过 15),表示导弹数;

第 2 行包含 N 个整数,为导弹依次飞来的高度(雷达给出的高度数据是不大于 30,000 的正整数)。

输出:

一个整数,表示最多能拦截的导弹数。

样例输入:

8
389 207 155 300 299 170 158 65

样例输出:

6

【参考程序 1】

```
#include<iostream>                                //最长不上升子序列
using namespace std;
int h[101][101];
int main()
{
    int n,max=0;
    cin>>n;
    for (int i=1;i<=n;++i)
    {
        cin>>h[i][1];
        h[i][2]=1;                                //初始当前系统拦截的导弹数为 1
    }
    for (int i=n-1;i>=1;--i)                        //累计从当前导弹开始,能够接到的最多的导弹
        for (int j=i+1;j<=n;++j)
            if (h[i][1]>=h[j][1]&&h[i][2]<=h[j][2]) h[i][2]=h[j][2]+1;
            //判断当前导弹高度是否小于系统能拦截的导弹高度并找出最优解(高度最高)
    for (int i=1;i<=n;++i)
        if (h[i][2]>max) max=h[i][2];                //寻找导弹数的最大值
    cout<<max<<endl;                                //输出储存在 max 的能够接到的最多的导弹数
    return 0;
}
```

【参考程序 2】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
using namespace std;
int n,maxx,k,x,a[100001],b[100001],c[100001],t,i=1;
int main()
{
    cin>>t;
    while(cin>>a[i])
    {
        maxx=0;
        for(int j=1;j<i;j++)                        //计算前 i-1 个导弹拦截方案
            if(a[j]>=a[i])                        //判断是否可以拦截到这个导弹
                if(b[j]>maxx) maxx=b[j];
        b[i]=maxx+1;                                //在前 i-1 个导弹最佳方案上+1
        if(b[i]>n) n=b[i];                            //求能拦截到的最多导弹数
        i++;
    }
    cout<<n;
}
```

第二节 背包问题

1. 采药【2.6 基本算法之动态规划 1775】

辰辰是个很有潜能、天资聪颖的孩子，他的梦想是成为世界上最伟大的医师。为此，他想拜附近最有威望的医师为师。医师为了判断他的资质，给他出了一个难题。医师把他带到一个到处都是草药的山洞里对他说：“孩子，这个山洞里有一些不同的草药，采每一株都需要一些时间，每一株也有它自身的价值。我会给你一段时间，在这段时间里，你可以采到一些草药。如果你是一个聪明的孩子，你应该可以让采到的草药的总价值最大。”

如果你是辰辰，你能完成这个任务吗？

输入：

第 1 行有两个整数 $T(1 \leq T \leq 1000)$ 和 $M(1 \leq M \leq 100)$ ， T 代表总共能够用来采药的时间， M 代表山洞里的草药的数目。接下来的 M 行每行包括两个在 $1 \sim 100$ 之间（包括 1 和 100）的整数，分别表示采摘某株草药的时间和这株草药的价值。

输出：

只有一行，包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

样例输入：

```
70 3
71 100
69 1
1 2
```

样例输出：

```
3
```

【参考程序】

```
#include<cstdio>
using namespace std;
int f[1001],w[101],c[101];
int main()
{
    int t,m;
    scanf("%d %d",&t,&m);
    for (int i=1;i<=m;++i)
        scanf("%d %d",&w[i],&c[i]); //每种药的摘采时间和价值
    for (int i=1;i<=m;++i) //循环每一种植物
        for (int v=t;v>=w[i];--v) //f[v]表示时间不超过 v 的最大价值
            if (f[v]<f[v-w[i]]+c[i])
                f[v]=f[v-w[i]]+c[i]; //判断将草药放入背包是否是更优解
    printf("%d\n",f[t]); //输出储存在 f[t]里的最优值
    return 0;
}
```

2. 数字组合【2.6 基本算法之动态规划 2985】

有 n 个正整数，找出其中和为 t (t 也是正整数) 的可能的组合方式。如：

$n=5$, 5 个数分别为 1,2,3,4,5, $t=5$;

那么可能的组合有 $5=1+4$ 和 $5=2+3$ 和 $5=5$ 三种组合方式。

输入：

第 1 行是两个正整数 $n(1 \leq n \leq 20)$ 和 $t(1 \leq t \leq 1000)$, 用空格隔开, 其中, 表示正整数的个数, t 为要求的和。

接下来的一行是 n 个正整数, 用空格隔开。

输出:

和为 t 的不同的组合方式的数目。

样例输入:

```
5 5
1 2 3 4 5
```

样例输出:

```
3
```

【参考程序】

```
#include<cstdio>                                //背包问题方案总数
using namespace std;
int a[21], f[1001];
int main()
{
    int n, t;
    scanf("%d %d", &n, &t);
    for (int i=1; i<=n; ++i)
        scanf("%d", &a[i]);
    f[0]=1;                                        //初始化
    for (int i=1; i<=n; ++i)                      //循环每一个正整数
        for (int j=t; j>=a[i]; --j)              //01 背包的方案总数
            f[j] += f[j-a[i]];                    //f[j]表示选择j的总方案数
    printf("%d\n", f[t]);                          //输出储存在 f[t]的方案总数
    return 0;
}
```

3. 宠物小精灵之收服【2.6 基本算法之动态规划 4978】

宠物小精灵是一部讲述小智和他的搭档皮卡丘一起冒险的故事。

一天, 小智和皮卡丘来到了小精灵狩猎场, 里面有很多珍贵的野生宠物小精灵。小智也想收服其中的一些小精灵。然而, 野生的小精灵并不容易收服。对于每一个野生小精灵而言, 小智可能需要使用很多个精灵球才能收服它, 而在收服过程中, 野生小精灵也会对皮卡丘造成一定的伤害(从而减少皮卡丘的体力)。当皮卡丘的体力小于等于 0 时, 小智就必须结束狩猎(因为他需要给皮卡丘疗伤), 而使得皮卡丘体力小于等于 0 的野生小精灵也不会被小智收服。当小智的精灵球用完时, 狩猎也宣告结束。

我们假设小智遇到野生小精灵时有两个选择: 收服它, 或者离开它。如果小智选择了收服, 那么一定会扔出能够收服该小精灵的精灵球, 而皮卡丘也一定会受到相应的伤害; 如果选择离开它, 那么小智不会损失精灵球, 皮卡丘也不会损失体力。

小智的目标有两个: 主要目标是收服尽可能多的野生小精灵; 如果可以收服的小精灵数量一样, 小智希望皮卡丘受到的伤害能够尽量小(剩余体力越大), 因为他们还要继续冒险。

现在已知小智的精灵球数量和皮卡丘的初始体力, 已知每一个小精灵需要的用于收服的精灵球数目和它在被收服过程中会对皮卡丘造成的伤害数目。请问, 小智该如何选择收服哪些小精灵以达到他的目标呢?

输入:

第 1 行包含三个整数: $N(0 < N < 1000)$, $M(0 < M < 500)$, $K(0 < K < 100)$, 分别代表小智的精灵球数量、皮卡丘初始的体力值、野生小精灵的数量。

之后的 K 行, 每一行代表一个野生小精灵, 包括两个整数: 收服该小精灵需要的精灵球

的数量,以及收服过程中对皮卡丘造成的伤害。

输出:

一行,包含两个整数:C,R,分别表示最多收服 C 个小精灵,以及收服 C 个小精灵时皮卡丘的剩余体力值最多为 R。

样例输入 1:

```
10 100 5
7 10
2 40
2 50
1 20
4 20
```

样例输入 2:

```
10 100 5
8 110
12 10
20 10
5 200
1 110
```

样例输出 1:

```
3 30
```

样例输出 2:

```
0 100
```

提示:

对于样例输入 1:小智选择:(7,10) (2,40) (1,20) 这样小智一共收服了 3 个小精灵,皮卡丘受到了 70 点伤害,剩余 $100-70=30$ 点体力,所以输出 3 30。

对于样例输入 2:小智一个小精灵都没法收服,皮卡丘也不会受到任何伤害,所以输出 0 100。

【参考程序 1】

```
#include<iostream>
using namespace std;
int f[1000][500],h[1000][500],a[100],b[100];
int main()
{
    int n,m,k;
    cin>>n>>m>>k;
    for (int i=1;i<=k;++i)
        cin>>a[i]>>b[i];
    for (int i=1;i<=k;++i)
        for (int v=n;v>=a[i];--v)
            for (int u=m;u>=b[i];--u)
                if (f[v][u]<f[v-a[i]][u-b[i]]+1) //判断收服当前小精灵是否是更优解
                {
                    f[v][u]=f[v-a[i]][u-b[i]]+1; //收复的精灵数加 1
                    h[v][u]=h[v-a[i]][u-b[i]]+b[i]; //储存收服当前小精灵后,皮卡丘消耗的体力
                }
    else
        if (f[v][u]==f[v-a[i]][u-b[i]]+1&&h[v][u]>h[v-a[i]][u-b[i]]+b[i])
            h[v][u]=h[v-a[i]][u-b[i]]+b[i]; //判断捕捉的数量相等,体力消耗较少的情况
```

```
cout<<f[n][m]<<" "<<m-h[n][m]<<endl;
/* 输出储存在 f[n][m]里最多收服的小精灵数及皮卡丘的原体力值 m 减去储存在
h[n][m]的皮卡丘消耗的体力值的差 */
return 0;
```

【参考程序 2】

```
#include<cstdio>
int f[1010][1010],n,m,k,i,j,l,a,b;
int main()
{
    scanf("%d%d%d",&n,&m,&k);
    for (l=1; l<=k; ++l)
    {
        scanf("%d%d",&a,&b);
        for (i=n; i>=a; i--)
            for (j=m; j>=b; j--)
                if (f[i-a][j-b]+1>f[i][j])
                    f[i][j]=f[i-a][j-b]+1; //判断是离开还是收服效益更大
    }
    printf("%d ",f[n][m]); //f[n][m]储存收复小精灵的最大数量
    i=m;
    while (i>=0&&f[n][i]==f[n][m]) i--; //循环计算已损失体力值
    printf("%d\n",m-i-1);
    return 0;
}
```

4. 买书【2.6 基本算法之动态规划 6049】

小明手里有 n 元钱全部用来买书,书的价格为 10 元,20 元,50 元,100 元。

问小明有多少种买书方案?(每种书可购买多本)

输入:

一个整数 $n(0 \leq n \leq 1000)$,代表总共钱数。

输出:

一个整数,代表选择方案种数

样例输入 1:

20

样例输出 1:

2

样例输入 2:

15

样例输出 2:

0

样例输入 3:

0

样例输出 3:

0

【参考程序 1】

```
#include<cstdio> //完全背包问题方案总数
#include<iostream>
using namespace std;
int a[4]={10,20,50,100}; //储存每一种价格
long long f[1001];
int main()
{
    int n;
    scanf("%d",&n);
    f[0]=1;
    for (int i=0;i<4;++i)
```



```

    for (int j=a[i];j<=n;++j)           //完全背包
        f[j]=f[j]+f[j-a[i]];           //累加价格为j的总方案数
    printf("%d\n",f[n]);                 //输出 f[n] 储存的最优值
    return 0;
}

```

【参考程序 2】

```

#include<iostream>                       //背包问题的方案总数
using namespace std;
int n,a[5]={0,10,20,50,100};             //每一种价格
long long f[32768];
int main()
{
    cin>>n;
    f[0]=1;                               //初始值
    for(int i=1;i<=4;i++)
        for(int j=n;j>=a[i];j--)         //f[j]表示选择j的总方案数
            for(int k=1;k<=j/a[i];k++)   //j/a[i]表示当前的钱能买几本书
                f[j]+=f[j-k*a[i]];       //将完全背包转化为 01 背包
    cout<<f[n]<<endl;                    //f[n]为最优解
    return 0;
}

```

5. Charm Bracelet【2.6 基本算法之动态规划 7113】

题意:经典 0—1 背包问题,有 n 个物品,编号为 i 的物品的重量为 $w[i]$,价值为 $c[i]$,现在要从这些物品中选一些物品装到一个容量为 m 的背包中,使得背包内物体在总重量不超过 m 的前提下价值尽量大。

输入:

第 1 行包含两个整数, n (物品数量, $n \leq 3500$)和 m (背包容量, $m \leq 12880$)。
接下来 n 行,每行两个整数 $w[i], c[i]$,表示每个物品的重量和价值。

输出:

仅一行,一个数,表示最大总价值。

样例输入:

```

4 6
1 4
2 6
3 12
2 7

```

样例输出:

```

23

```

【参考程序】

```

#include<iostream>                       //01 背包
using namespace std;
int m,n,w[20001],c[20001],f[20001];
int main()
{
    cin>>n>>m;                           //背包容量 m 和物品数量 n
    for(int i=1;i<=n;i++)
        cin>>w[i]>>c[i];                 //每个物品的重量和价值
    for(int i=1;i<=n;i++)                 //设 f[v]表示重量不超过 v 公斤的最大价值

```

```
for(int v=m;v>=w[i];v--) //判断是否将物体放入背包
    if(f[v-w[i]]+c[i]>f[v]) f[v]=f[v-w[i]]+c[i];
cout<<f[m]; //f[m]为最优解
return 0;
}
```

6. 装箱问题【2.6 基本算法之动态规划 8785】

有一个箱子容量为 v (正整数, $0 \leq v \leq 20000$), 同时有 n 个物品 ($0 \leq n \leq 30$), 每个物品有一个体积 (正整数)。

要求 n 个物品中, 任取若干个装入箱内, 使箱子的剩余空间为最小。

输入:

第 1 行是一个整数 v , 表示箱子容量;

第 2 行是一个整数 n , 表示物品数。

接下来 n 行, 每行一个正整数 (不超过 10000), 分别表示这 n 个物品的各自体积。

输出:

一个整数, 表示箱子剩余空间。

样例输入:

```
24
6
8
3
12
7
9
7
```

样例输出:

```
0
```

【参考程序】

```
#include<cstdio>
using namespace std;
int w[31],f[20001];
int main()
{
    int v,n;
    scanf("%d %d",&v,&n);
    for (int i=1;i<=n;++i)
        scanf("%d",&w[i]); //物品体积
    for (int i=1;i<=n;++i)
        for (int j=v;j>=w[i];--j) //表示不超过 j 体积的最大体积
            if (f[j]<f[j-w[i]]+w[i])
                f[j]=f[j-w[i]]+w[i];
    //寻找不超过 j 体积的最大体积并判断装入箱子是否是最优解
    printf("%d\n",v-f[v]); //f[v]为装入箱内的最大体积,v-f[v]求出箱内的剩余体积
    return 0;
}
```

7. 开餐馆【2.6 基本算法之动态规划 6045】

信息学院的同学小明毕业之后打算创业开餐馆, 现在共有 n 个地点可供选择。小明打算从中选择合适的位置开设一些餐馆。这 n 个地点排列在同一条直线上。我们用一个整数

序列 $m_1, m_2, \dots, m_n$ 来表示他们的相对位置。由于地段关系,开餐馆的利润会有所不同。我们用 p_i 表示在 m_i 处开餐馆的利润。为了避免自己的餐馆内部竞争,餐馆之间的距离必须大于 k 。请你帮助小明选择一个总利润最大的方案。

输入:

第 1 行是整数 $T(1 \leq T \leq 1000)$, 表明有 T 组测试数据。紧接着有 T 组连续的测试。每组测试数据有三行:

第 1 行:地点总数 $n(n \leq 100)$, 距离限制 $k(0 < k \leq 1000)$;

第 2 行: n 个地点的位置 $m_1, m_2, \dots, m_n(0 < m_i \leq 1,000,000$ 且为整数,升序排列);

第 3 行: n 个地点的餐馆利润 $p_1, p_2, \dots, p_n(0 < p_i \leq 1000$ 且为整数)。

输出:

对应于每组测试数据可能的最大利润。

样例输入:

```
2
3 11
1 2 15
10 2 30
3 16
1 2 15
10 2 30
```

样例输出:

```
40
30
```

【参考程序 1】

```
#include<cstring>
#include<iostream>
using namespace std;
int a[100],f[1000000];
int main()
{
    int t,n,k,m;
    cin>>t;
    for (int l=1;l<=t;++l)
    {
        memset(a,0,sizeof(a));
        memset(f,0,sizeof(f));          //每次处理之前都清空一次数组
        cin>>n>>k;
        for (int i=1;i<=n;++i)
            cin>>a[i];
        for (int i=1;i<=n;++i)
            cin>>f[a[i]];                //读入开餐馆的位置及利润
        for (int i=1;i<=n;++i)
        {
            m=i-1;
            while (a[i]-a[m]<=k&& m>0) m--;
            //寻找与当前地点距离最近且能同时开餐观的地点
            f[a[i]]=max(f[a[i-1]],f[a[m]]+f[a[i]]);
            //判断在当前地点上开餐馆是否能获得更大的利润
        }
    }
}
```

```

    cout<<f[a[n]]<<endl;    //输出储存在 f[a[n]]中的能够获得的最大利润
}
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int t,k,n,i,j,l,tot[1001];
int main()
{
    scanf("%d",&t);
    for(l=1;l<=t;l++)
    {
        int m[101],f[101],p[101];
        memset(f,0,sizeof(f));
        scanf("%d%d",&n,&k);
        for(i=1;i<=n;i++)
            scanf("%d",&m[i]);
        for(i=1;i<=n;i++)
            scanf("%d",&p[i]);
        f[1]=p[1];    //初始开第 1 家餐馆的利益为 p[1]
        for(i=2;i<=n;i++)
            for(j=0;j<i;j++)
                if((m[i]-m[j]>k)||(!j)) //距离大于 k 或 j=0 时比较只开第 i 家餐馆的利益
                    f[i]=max(f[i-1],f[j]+p[i]);
                //求利益更大的开餐馆方法
        tot[l]=f[n];    //计算总利润
    }
    for(l=1;l<=t;l++)
        printf("%d\n",tot[l]);    //输出总利润
    return 0;
}

```

第三节 动态规划经典题

1. 公共子序列【2.6 基本算法之动态规划 1808】

当且仅当存在严格上升的序列 $\langle i_1, i_2, \dots, i_k \rangle$, 使得对 $j=1, 2, \dots, k$, 有 $x_{i_j} = z_j$ 时, 我们称序列 $Z = \langle z_1, z_2, \dots, z_k \rangle$ 是序列 $X = \langle x_1, x_2, \dots, x_m \rangle$ 的子序列。比如 $Z = \langle a, b, f, c \rangle$ 是 $X = \langle a, b, c, f, b, c \rangle$ 的子序列。

现在给出两个序列 X 和 Y , 你的任务是找到 X 和 Y 的最大公共子序列, 也就是说要找到一个最长的序列 Z , 使得 Z 既是 X 的子序列也是 Y 的子序列。

输入:

包括多组测试数据。每组数据包括一行, 给出两个长度不超过 200 的字符串, 表示两个序列。两个字符串之间由若干个空格隔开。

输出:

对每组输入数据, 输出一行, 给出两个序列的最大公共子序列的长度。

样例输入:

```
abcfbc          abfcab
programming     contest
abcd            mnp
```

样例输出:

```
4
2
0
```

【参考程序】

```
#include <cstring>
#include <iostream>
using namespace std;
int f[201][201];
int main()
{
    int la, lb;
    char a[201], b[201];
    while (cin >> a >> b)
    {
        la = strlen(a);
        lb = strlen(b);           // la, lb 存储 a, b 字符串的长度
        for (int i = 1; i <= la; ++i)
            for (int j = 1; j <= lb; ++j)
            {
                if (a[i-1] == b[j-1]) f[i][j] = f[i-1][j-1] + 1;
                /* 如果 a[i-1], b[j-1] 相等, 那么到 a[i-1], b[j-1] 的最长公共子序列的长
                   度为到 a[i-2], b[j-2] 的最长公共子序列的长度加 1 */
                else f[i][j] = max(f[i-1][j], f[i][j-1]);
                /* 否则, 到 a[i-1], b[j-1] 的最长公共子序列的长度为到 a[i-2], b[j-1] 与
                   到 a[i-1], b[j-2] 的最长公共子序列的长度中较长的长度 */
            }
        cout << f[la][lb] << endl;    // 输出储存在 f[la][lb] 中的最优值
    }
    return 0;
}
```

2. 计算字符串距离【2.6 基本算法之动态规划 2988】

对于两个不同的字符串,我们有一套操作方法来把他们变得相同,具体方法为:

修改一个字符(如把“a”替换为“b”);

删除一个字符(如把“traveling”变为“travelng”)。

比如对于“abcdefg”和“abcdef”两个字符串来说,我们认为可以通过增加或减少一个“g”的方式来达到目的。无论增加还是减少“g”,我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。

给定任意两个字符串,写出一个算法来计算出它们的距离。

输入:

第1行有一个整数n。表示测试数据的组数。

接下来共n行,每行两个字符串,用空格隔开,表示要计算距离的两个字符串。

字符串长度不超过1000。

输出:

针对每一组测试数据输出一个整数,是两个字符串的距离。

样例输入:

```
3
abcdefg abcdef
ab ab
mnlkj jlknm
```

样例输出:

```
1
0
4
```

【参考程序】

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int f[5000][5000];
char a[5000],b[5000];
int i,j,k,m,n,u;
int main()
{
    cin>>u;
    for(int k=1;k<=u;k++)
    {
        scanf("%s%s",a,b);
        m=strlen(a);
        n=strlen(b);
        for(i=1;i<=m;i++) f[i][0]=i;
        for(i=1;i<=n;i++) f[0][i]=i;           //边界的初始化
        for(i=1;i<=m;i++)
            for(j=1;j<=n;j++)
                if(a[i-1]==b[j-1]) f[i][j]=f[i-1][j-1];           //判断头尾是否要改
                else f[i][j]=min(min(f[i-1][j],f[i][j-1]),f[i-1][j-1])+1;
                //改 a[i]为 b[i]、a[i]后插入 b[j-1]、删除 a[i]三种中改动次数最少的一种
        cout<<f[m][n]<<endl;           //最终结果
        /* 三种:f[i-1][j-1]+1    表示改 a[i]为 b[j]
            f[i-1][j]+1        表示删 a[i]
            f[i][j-1]+1        表示 a[i]后插入 b[j-1] */
    }
}
```

```
return 0;
```

```
}
```

3. 糖果【2.6 基本算法之动态规划 2989】

由于在维护世界和平的事务中做出巨大贡献,Dzx 被赠予糖果公司 2010 年 5 月 23 日当天无限量糖果免费优惠券。在这一天,Dzx 可以从糖果公司的 N 件产品中任意选择若干件带回家享用。糖果公司的 N 件产品每件都包含数量不同的糖果。Dzx 希望他选择的产品包含的糖果总数是 K 的整数倍,这样他才能平均地将糖果分给帮助他维护世界和平的伙伴们。当然,在满足这一条件的基础上,糖果总数越多越好。Dzx 最多能带走多少糖果呢?

注意:Dzx 只能将糖果公司的产品整件带走。

输入:

第 1 行包含两个整数 $N(1 \leq N \leq 100)$ 和 $K(1 \leq K \leq 100)$ 。

以下 N 行每行 1 个整数,表示糖果公司该件产品中包含的糖果数目,不超过 1,000,000。

输出:

符合要求的最多能达到的糖果总数,如果不能达到 K 的倍数这一要求,则出 0。

样例输入:

```
5 7
1
2
3
4
5
```

样例输出:

```
14
```

提示:

Dzx 的选择是 $2+3+4+5=14$,这样糖果总数是 7 的倍数,并且是总数最多的选择。

【参考程序】

```
#include<cstdio>
#include<iostream>
using namespace std;
int f[101][100];
int main()
{
    int n,k,x;
    cin>>n>>k;
    for (int i=1;i<=n;++i)
    {
        scanf("%d",&x);
        f[i][x%k]=x; //存储余数相同的
        for (int j=0;j<k;++j)
            if (f[i-1][j]) //判断是否存在余数为j的组合
                f[i][(j+x)%k]=f[i-1][j]+x; //储存前i件产品中,能够形成的所有的组合
        for (int j=0;j<k;++j)
            f[i][j]=max(f[i][j],f[i-1][j]); //寻找前i件中,能够形成所有组合的最大值
    }
    printf("%d\n",f[n][0]); //输出储存在f[n][0]里的最多能达到的糖果总数
    return 0;
}
```

4. 鸡蛋的硬度【2.6 基本算法之动态规划 7627】

最近 XX 公司举办了一个奇怪的比赛：鸡蛋硬度之王争霸赛。参赛者是来自世界各地的母鸡，比赛的内容是看谁下的蛋最硬，更奇怪的是 XX 公司并不使用什么精密仪器来测量蛋的硬度，他们采用了一种最老土的办法：从高度扔鸡蛋来测试鸡蛋的硬度，如果一次母鸡下的蛋从高楼的第 a 层摔下来没摔破，但是从 $a+1$ 层摔下来时摔破了，那么就说这只母鸡下的鸡蛋的硬度是 a 。你当然可以找出各种理由说明这种方法不科学，比如同一只母鸡下的蛋硬度可能不一样等，但是这不影响 XX 公司的争霸赛，因为他们只是为了吸引大家的眼球，一个个鸡蛋从 100 层的高楼上掉下来的时候，这情景还是能吸引很多人驻足观看的，当然，XX 公司也绝不会忘记在高楼上挂一条幅，写上“XX 公司”的字样——这比赛不过是 XX 公司的一个另类广告而已。

勤于思考的小 A 总是能从一件事情中发现一个数学问题，这件事也不例外。“假如有很多同样硬度的鸡蛋，那么我可以用二分的办法用最少的次数测出鸡蛋的硬度”，小 A 对自己的这个结论感到很满意，不过很快麻烦来了，“但是，假如我的鸡蛋不够用呢，比如我只有 1 个鸡蛋，那么我就不得不从第 1 层楼开始一层一层地扔，最坏情况下我要扔 100 次。如果有 2 个鸡蛋，那么就从 2 层楼开始的地方扔……等，不对，好像应该从 $1/3$ 的地方开始扔才对，嗯，好像也不一定啊……3 个鸡蛋怎么办，4 个，5 个，更多呢……”，和往常一样，小 A 又陷入了一个思维僵局，与其说他是勤于思考，不如说他是喜欢自找麻烦。

好吧，既然麻烦来了，就得有人去解决，小 A 的麻烦就靠你来解决了。

输入：

包括多组数据，每组数据一行，包含两个正整数 n 和 m ($1 \leq n \leq 100, 1 \leq m \leq 10$)，其中 n 表示楼的高度， m 表示你现在拥有的鸡蛋个数，这些鸡蛋硬度相同（即它们从同样高的地方掉下来要么都摔碎要么都不碎），并且小于等于 n 。你可以假定硬度为 x 的鸡蛋从高度小于等于 x 的地方摔无论如何都不会碎（没摔碎的鸡蛋可以继续使用），而只要从比 x 高的地方扔必然会碎。

对每组输入数据，你可以假定鸡蛋的硬度在 0 至 n 之间，即在 $n+1$ 层扔鸡蛋一定会碎。

输出：

对于每一组输入，输出一个整数，表示使用最优策略在最坏情况下所需要的扔鸡蛋次数。

样例输入：

100 1
100 2

样例输出：

100
14

提示：

最优策略指在最坏情况下所需要的扔鸡蛋次数最少的策略。

如果只有一个鸡蛋，你只能从第一层开始扔，在最坏的情况下，鸡蛋的硬度是 100，所以需要扔 100 次。如果采用其他策略，你可能无法测出鸡蛋的硬度（比如你第一次在第二层的地方扔，结果碎了，这时你不能确定硬度是 0 还是 1），即在最坏情况下你需要扔无限次，所以第一组数据的答案是 100。

【参考程序】

```
#include<iostream>
using namespace std;
int f[101][11];
int main()
{
    int n,m;
    while (cin>>n>>m)        //循环输入每组数据
```



```

{
    for (int i=1;i<=n;++i)
        for (int j=1;j<=m;++j)
            f[i][j]=i; //初始值赋为楼层数,即最差的策略时遇到的最坏的情况
    for (int i=1;i<=n;++i)
        for (int k=1;k<=i;++k)
            for (int j=2;j<=m;++j)
                f[i][j]=min(f[i][j],max(f[k-1][j-1],f[i-k][j])+1);
    //当有 i 层楼、j 个鸡蛋时,判断最坏的情况时,与之前求出的次数作比较,储存较小值
    cout<<f[n][m]<<endl; //输出储存在 f[n][m]中最坏情况下次数最少的策略
}
return 0;
}

```

5. 大盗阿福【2.6 基本算法之动态规划 8462】

阿福是一名经验丰富的大盗。趁着月黑风高,阿福打算今晚洗劫一条街上的店铺。

这条街上一共有 N 家店铺,每家店中都有一些现金。阿福事先调查得知,只有当他同时洗劫了两家相邻的店铺时,街上的报警系统才会启动,然后警察就会蜂拥而至。

作为一向谨慎作案的大盗,阿福不愿意冒着被警察追捕的风险行窃。他想知道,在不惊动警察的情况下,他今晚最多可以得到多少现金?

输入:

第 1 行是一个整数 $T(T \leq 50)$,表示一共有 T 组数据。

接下来的每组数据包含两行,第 1 行是一个整数 $N(1 \leq N \leq 100,000)$,表示一共有 N 家店铺。第 2 行是 N 个被空格分开的正整数,表示每一家店铺中的现金数量。每家店铺中的现金数量均不超过 1000。

输出:

对于每组数据,输出一行。该行包含一个整数,表示阿福在不惊动警察的情况下可以得到的现金数量。

样例输入:

```

2
3
1 8 2
4
10 7 6 14

```

样例输出:

```

8
24

```

提示:

对于第 1 组样例,阿福选择第 2 家店铺行窃,获得的现金数量为 8。

对于第 2 组样例,阿福选择第 1 和 4 家店铺行窃,获得的现金数量为 $10+14=24$ 。

【参考程序 1】

```

#include<cstdio>
#include<iostream>
using namespace std;
int a[100001],f[100001];
int main()
{
    int t,n;

```

```
scanf("%d",&t);           //共有 t 组数据要读入
for (int l=1;l<=t;++l)    //循环读入每组数据
{
    scanf("%d",&n);
    for (int i=1;i<=n;++i)
        scanf("%d",&a[i]);
    f[1]=a[1];             //假设先洗劫第一间店铺
    for (int i=2;i<=n;++i)
        f[i]=max(f[i-1],f[i-2]+a[i]); //判断洗劫当前店铺是否是更优解
    printf("%d\n",f[n]);   //f[n]表示洗劫能够得到的最多的现金数
}
return 0;
```

【参考程序 2】

```
#include<iostream>
#include<cstring>
using namespace std;
int a[100001],n,f[100001],maxx=0,t;
int main()
{
    cin>>t;
    while(t-->0)           //用于处理 t 组数据的读入
    {
        memset(a,0,sizeof(a));
        memset(f,0,sizeof(f)); //每次处理之前先清空数组
        cin>>n;
        for(int i=1;i<=n;i++) cin>>a[i];
        maxx=a[1];
        f[1]=a[1];
        f[2]=a[2];          //假设先选前两家
        if(n>1&&a[2]>a[1]) maxx=a[2]; //前两家选现金多的一家
        for(int i=3;i<=n;i++)
        {
            f[i]=max(f[i-2],f[i-3])+a[i]; //选择洗劫先前店铺和当前店铺较大值
            if(f[i]>maxx) maxx=f[i];      //maxx 求其最大值,即最优解
        }
        cout<<maxx<<endl;
    }
    return 0;
```

6. 股票买卖【2.6 基本算法之动态规划 8464】

最近越来越多的人都投身股市,阿福也有点心动了。谨记着“股市有风险,入市需谨慎”,阿福决定先来研究一下简化版的股票买卖问题。

假设阿福已经准确预测出了某只股票在未来 N 天的价格,他希望买卖两次,使得获得的利润最高。为了计算简单起见,利润的计算方式为卖出的价格减去买入的价格。

同一天可以进行多次买卖。但是在第一次买入之后,必须要先卖出,然后才可以第二次买入。现在,阿福想知道他最多可以获得多少利润。

输入:

第 1 行是一个整数 $T(T \leq 50)$, 表示一共有 T 组数据。

接下来的每组数据包含两行, 第 1 行是一个整数 $N(1 \leq N \leq 100,000)$, 表示一共有 N 天。第 2 行是 N 个被空格分开的整数, 表示每天该股票的价格。该股票每天的价格的绝对值均不会超过 1,000,000。

输出:

对于每组数据, 输出一行。该行包含一个整数, 表示阿福能够获得的最大的利润。

样例输入:

```
3
7
5 14 -2 4 9 3 17
6
6 8 7 4 1 -2
4
18 9 5 2
```

样例输出:

```
28
2
0
```

提示:

对于第一组样例, 阿福可以第 1 次在第 1 天买入(价格为 5), 然后在第 2 天卖出(价格为 14)。第 2 次在第 3 天买入(价格为 -2), 然后在第 7 天卖出(价格为 17)。一共获得的利润是 $(14-5)+(17-(-2))=28$ 。

对于第二组样例, 阿福可以第 1 次在第 1 天买入(价格为 6), 然后在第 2 天卖出(价格为 8)。第 2 次仍然在第 2 天买入, 然后在第 2 天卖出。一共获得的利润是 $8-6=2$ 。

对于第三组样例, 由于价格一直在下跌, 阿福可以随便选择一天买入之后迅速卖出。获得的最大利润为 0。

提示:

经典算法 Baidu 搜索, 深刻体会。

【参考程序】

```
#include<cstdio>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn=100000+100;
int price[maxn];
int mmin[maxn],mmax[maxn];
int li[maxn],ri[maxn];
int main()
{
    int t;
    scanf("%d",&t);
    while(t--){
        int n;
        scanf("%d",&n);
        for(int i=1;i<=n;i++) scanf("%d",price+i);
        mmin[1]=price[1]; //假设先买下第一天的股票
        li[1]=0;
        for(int i=2;i<=n;i++) //从第二天的股票开始选择
```

```

{
    mmin[i]=min(mmin[i-1],price[i]);
    //比较前一天购买股票方案和当前股票的较小值
    li[i]=max(li[i-1],price[i]-mmin[i]);
    //比较前一天收回股票方案和当前股票利润的较大值
}
mmax[n]=price[n]; //假设购买第 n 天的股票
ri[n]=0;
for(int i=n-1;i>=1;i--)
{
    mmax[i]=max(mmax[i+1],price[i]);
    ri[i]=max(ri[i+1],mmax[i]-price[i]);
}
//同上,即相反处理一遍
int ans=0;
for(int i=1;i<=n;i++)
    ans=max(ans,li[i]+ri[i]); //两次处理后取最大值
printf("%d\n",ans);
}
return 0;
}

```

7. 鸣人的影分身【2.6 基本算法之动态规划 8467】

在火影忍者的世界里,令敌人捉摸不透是非常关键的。我们的主角漩涡鸣人所拥有的一个招数——多重影分身之术——就是一个很好的例子。

影分身是由鸣人身体的查克拉能量制造的,使用的查克拉越多,制造出的影分身越强。

针对不同的作战情况,鸣人可以选择制造出各种强度的影分身,有的用来佯攻,有的用来发起致命一击。

那么问题来了,假设鸣人的查克拉能量为 M ,他影分身的个数最多为 N ,那么制造影分身时有多少种(用 K 表示)不同的分配方法?(影分身可以被分配到 0 点查克拉能量)

输入:

第 1 行是测试数据的数目 $t(0 \leq t \leq 20)$ 。以下每行均包含两个整数 M 和 $N(1 \leq M, N \leq 10)$,以空格分开。

输出:

对输入的每组数据 M 和 N ,用一行输出相应的 K 。

样例输入:

```

1
7 3

```

样例输出:

```

8

```

【参考程序】

```

#include<iostream>
using namespace std;
int f[11][11];
int main()
{
    int t,m,n;
    cin>>t; //用于处理 t 组的读入
    for (int l=1;l<=t;++l)

```

```

{
    cin >> m >> n;
    for (int i=0; i<=m; ++i)
        for (int j=0; j<=n; ++j)
            if (i==0 || j==1) f[i][j]=1;
                //当查克拉为 0 或影分身为 1 时,只有一种分配方法
            else if (j>i) f[i][j]=f[i][i];
                //当影分身多于查克拉时,分配方法就等于影分身等于查克拉时的分配方法
            else f[i][j]=f[i-j][j]+f[i][j-1];
                //否则,分配方法就等于每个影分身至少分配一个查克拉的分配方法加上至少
                //一个影分身分配的查克拉为 0 的分配方法的和
    cout << f[m][n] << endl;
}
return 0;
}

```

8. 数的划分【2.6 基本算法之动态规划 8787】

将整数 n 分成 k 份,且每份不能为空,任意两份不能相同(不考虑顺序)。

例如: $n=7, k=3$,下面三种分法被认为是相同的。

1,1,5; 1,5,1; 5,1,1;

问:有多少种不同的分法?

输入:

两个整数 $n, k (6 \leq n \leq 200, 2 \leq k \leq 6)$,中间用单个空格隔开。

输出:

一个整数,即不同的分法。

样例输入:

7 3

样例输出:

4

提示:

四种分法为:1,1,5;1,2,4;1,3,3;2,2,3。

【参考程序】

```

#include <cstdio>                                //放苹果类似
using namespace std;
int f[201][7], i, j, n, k;
int main()
{
    scanf("%d%d", &n, &k);
    for (i=1; i<=n; ++i)
        for (j=1; j<=k; ++j)
        {
            if (i<j) f[i][j]=0;                    //不够分时没有分法
            else if (j==1) f[i][j]=1;
            else if (i==j) f[i][j]=1;                //i=1 或 j=i 时只有一种拆法
            else f[i][j]=f[i-1][j-1]+f[i-j][j];    //递推式
        }
    printf("%d\n", f[n][k]);                        //f[n][k]储存最多分法
    return 0;
}

```

9. Maximum sum[2.6 基本算法之动态规划 1481]

题意:对于给定的整数序列 $A=\{a_1, a_2, \dots, a_n\}$, 找出两个不重合连续子段, 使得两子段中所有数字的和最大。我们如下定义函数 $d(A)$:

$$d(A) = \max_{1 \leq s_1 \leq t_1 \leq s_2 \leq t_2 \leq n} \left\{ \sum_{i=s_1}^{t_1} a_i + \sum_{j=s_2}^{t_2} a_j \right\}$$

我们的目标就是求出 $d(A)$ 。

输入:

第 1 行是一个整数 $T(\leq 30)$, 代表一共有多少组数据。

接下来是 T 组数据。

每组数据包含两行, 第 1 行是一个整数, 代表数据个数 $n(2 \leq n \leq 50,000)$, 第 2 行是 n 个整数 $a_1, a_2, \dots, a_n(|a_i| \leq 10,000)$ 。

输出:

输出一个整数, 就是 $d(A)$ 的值。

样例输入:

```
1
10
1 -1 2 2 3 -3 4 -4 5 -5
```

样例输出:

```
13
```

提示:

就是求最大子段和问题, 样例取 $\{2, 2, 3, -3, 4\}$ 和 $\{5\}$, Baidu 搜 POJ 2479 Maximum sum, 可获得大量经典最大子段和问题的题目解析, 本题 $O(n^2)$ 算法超时, 必须用 $O(n)$ 算法。

【参考程序】

```
#include<cstdio>
#include<cstring>
#define MAXN 50005 //把 MAXN 赋为 50005
int num[MAXN], lsum[MAXN], rsum[MAXN];
int n;
int main(){
    int t;
    scanf("%d", &t);
    while(t--){
        int i;
        scanf("%d", &n);
        for(i=1; i<=n; i++){
            scanf("%d", &num[i]);
            memset(lsum, 0, sizeof(lsum));
            memset(rsum, 0, sizeof(rsum)); //每次处理之前清空数组
            lsum[1]=num[1]; //假设为 1
            for(i=2; i<=n; i++){
                if(lsum[i-1]>0){
                    lsum[i]=lsum[i-1]+num[i];
                }
                else
                    lsum[i]=num[i];
            }
            rsum[n]=num[n];
```

```

for(i=n-1 ; i >= 1 ; i--){
    if(rsum[i+1] > 0){
        rsum[i]=rsum[i+1]+num[i] ;
    }
    else
        rsum[i]=num[i] ;
}
int nmax=-1000000 ;           //用于下文求最大值
int temp=-1000000 ;
for(i=1 ; i < n ; i++){
    if(temp < lsum[i])
        temp=lsum[i] ;
    if(nmax < temp+rsum[i+1])    //通过比较求最大值
        nmax=temp+rsum[i+1];
}
printf("%d\n" , nmax) ;
}
return 0;
}

```

10. 最长上升公共子序列【2.6 基本算法之动态规划 2000】

给定两个整数序列,写一个程序求它们的最长上升公共子序列。

当满足以下条件的时候,我们将长度 N 的序列 $S_1, S_2, \dots, S_N$ 称为长度为 M 的序列 $A_1, A_2, \dots, A_M$ 的上升子序列:

存在 $1 \leq i_1 < i_2 < \dots < i_N \leq M$, 使得对所有 $1 \leq j \leq N$, 均有 $S_j = A_{i_j}$, 且对于所有的 $1 \leq j < N$, 均有 $S_j < S_{j+1}$ 。

输入:

每个序列用两行表示,第 1 行是长度 M ($1 \leq M \leq 500$), 第 2 行是该序列的 M 个整数 A_i ($-2^{31} \leq A_i < 2^{31}$)

输出:

第 1 行,输出两个序列的最长上升公共子序列的长度 L ;

第 2 行,输出该子序列。

如果有不止一个符合条件的子序列,则输出任何一个即可。

样例输入:

```

5
1 4 2 5 -12
4
-12 1 2 4

```

样例输出:

```

2
1 4

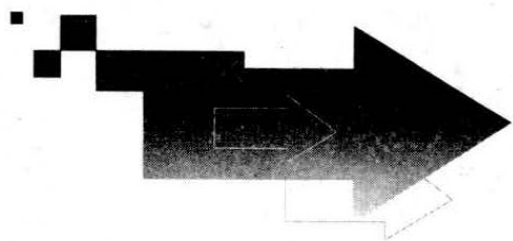
```

提示:

经典算法 Baidu 搜索,深刻体会。

第三部分

数据结构



第一章 栈

1. 表达式括号匹配(stack)

假设一个表达式有英文字母(小写)、运算符(+, -, *, /)和左右小(圆)括号构成,以“@”作为表达式的结束符。请编写一个程序检查表达式中的左右圆括号是否匹配,若匹配,则返回“YES”;否则返回“NO”。表达式长度小于 255,左圆括号少于 20 个。

输入:

输入文件 stack.in 包括一行数据,即表达式。

输出:

输出文件 stack.out 包括一行,即“YES”或“NO”。

样例输入:

样例 1:

$2 * (x + y) / (1 - x) @$

样例 2:

$(25 + x) * (a * (a + b + b) @$

样例输出:

样例 1:

YES

样例 2:

NO

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
char a[10001];
int i=0, tot=0;                                     //tot 用来判断括号是否匹配
int main()
{
    scanf("%s", &a);
    while (a[i] != '@')                             //读入终止条件
    {
        if (a[i] == '(') tot++;                     //如果是左括号,就入栈
        if (a[i] == ')')                             //如果是右括号
        {
            if (tot >= 0) tot--;                     //栈顶元素出栈
            else break;                             //tot<0 表示存在无法匹配的右括号
        }
        i++;
    }
    if (tot != 0) cout << "NO ";
    else cout << "YES ";                            //当 tot 的值为 0 时(也就是没有多余的括号),说明匹配
}
```

【参考程序 2】

```
#include<iostream>
using namespace std;
int js();
string s;
int main()
```

```

{
    getline(cin,s);           //读入字符串
    if (js()==true) cout<<" YES\n "<<endl; //匹配成功
    else cout<<" NO\n "<<endl; //匹配失败
    return 0;
}

int js()
{
    int top=0;                //初始化
    int i=0;
    while (s[i]! = '@')      //语句以@结束
    {
        if (s[i]=='(') top++; //如果是“(”,累加
        else if (s[i]==')')   //如果是“)”(其余字符忽略)
        {
            if (top>0) top--; //如果有“(”与之匹配,累计数-1
            else return false; //否则无法匹配,程序直接结束
        }
        i++;                  //下一个字符
    }
    if (top! =0) return false; //如果还有未匹配的“(”,匹配失败,返回 false
    else return true;         //否则匹配成功,返回 true
}

```

2. 括号匹配检验(check)

假设表达式中允许包含两种括号:圆括号和方括号,其嵌套的顺序随意,如([]())或[([])]等为正确的匹配,[(])或([] ()或(()))均为错误的匹配。

现在的问题是,要求检验一个给定表达式中的括号是否正确匹配。

输入一个只包含圆括号和方括号的字符串,判断字符串中的括号是否匹配,匹配就输出“OK”,不匹配就输出“Wrong”。如:输入一个字符串:[([])],输出:OK。

输入:

输入仅一行字符(字符个数小于 255)。

输出:

匹配就输出“OK”,不匹配就输出“Wrong”。

样例输入:

[(])

样例输出:

Wrong

【参考程序 1】

```

#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
char a[10001]={0};
int main()
{
    int i,n,t,j,k; string s; bool f;
    cin>>s; t=0;

```

```

for(j=0;j<=s.size()-1;j++)
{
    if (s[j]=='('||s[j]=='[') a[++t]=s[j];
        //如果是左括号,就进栈
    if (s[j]==')') if (a[t]=='(') t--; else t++;
    if (s[j]==']') if (a[t]=='[') t--; else t++;
        //不匹配就出栈
}
if (t==0) cout<<" OK "<<endl; //栈空,匹配成功
else cout<<" Wrong "<<endl; //否则有未匹配的括号或匹配顺序不正确
return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<string>
#include<stack> //使用 STL 中的模板
using namespace std;
int n;
string s;
bool check(string s)
{
    stack<char>p; //p 为栈
    p.push('#'); //栈底为#,避免越界
    for (int i=0; i<s.size(); i++)
    {
        char c=s[i];
        if (c==')') //是右括号
        {
            if (p.top()!='(') //没有与它匹配的“(”,该串不匹配
                return false;
            else p.pop(); //能匹配,出栈
        }
        else if (c==']')
        {
            if (p.top()!='[') //没有与它匹配的“[”,该串不匹配
                return false;
            else p.pop(); //能匹配,出栈
        }
        else
            p.push(c); //是左括号则入栈
    }
    return (p.size()==1); //只剩栈底的“#”,说明全部匹配
}

int main()
{
    cin>>s;
    if (check(s))
        cout<<" OK "<<endl; //匹配成功
}

```

```

else
    cout<<" Wrong "<<endl; //匹配失败
return 0;
}

```

3. 字符串匹配问题(strs)

字符串中只含有括号 (), [], <>, {}, 判断输入的字符串中括号是否匹配。如果括号有互相包含的形式,从内到外必须是<>, (), [], {}, 例如:输入 [()], 输出 YES; 而输入 ([]), ([]) 都应该输出 NO。

输入:

文件的第 1 行为一个整数 n, 表示以下有多少个由括号组成的字符串。接下来的 n 行, 每行都是一个由括号组成的长度不超过 255 的字符串。

输出:

在输出文件中有 n 行, 每行都是 YES 或 NO。

样例输入:

```

5
{{{}}<><>()()[][]
{{{}}{}}<><><><>>(( ))(( ))[][][][]
{{{}}{}}<><><><>>(( ))(( ))[][][][]
{<>}{[]}<<<<>><<<<>>>((<>))(( ))[(<>)][][]
><>{[]}<<<<>><<<<>>>((<>))(( ))[(<>)][][]

```

标例输出:

```

YES
YES
YES
YES
NO

```

【参考程序】

```

#include<algorithm>
#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<cstring>
#include<string>
using namespace std;
int a[10001]={0},b[10001]={0}; char s[10001];
void gsy(int a[])
{
    int i;
    for(i=0;i<strlen(s);i++)
    { //将括号转换成数字,以比较括号级别
        if (s[i]=='{') a[i+1]=1;
        if (s[i]=='[') a[i+1]=2;
        if (s[i]=='(') a[i+1]=3;
        if (s[i]=='<') a[i+1]=4;
        if (s[i]=='>') a[i+1]=5;
        if (s[i]==')') a[i+1]=6;
    }
}

```

```

        if (s[i]=='') a[i+1]=7;
        if (s[i]=='') a[i+1]=8;
    }
}
int main()
{
    int i,n,t=0,m,j,k;
    cin>>n;
    for(i=1;i<=n;i++)
    {
        cin>>s; t=0;
        gsy(a);
        for(j=1;j<=strlen(s);j++)
        {
            if (a[j]<=4)                //是左括号
            if (a[j]>=b[t]) b[++t]=a[j]; //如果括号级别顺序符合,左括号进栈
            else break;                  //否则匹配失败
            if (a[j]>=5)                  //右括号进栈
            if (a[j]+b[t]==9) t--;        //匹配成功,出栈
            else t++;
        }
        if (t==0) cout<<" YES "<<endl;
        else cout<<" NO "<<endl;
    }
    return 0;
}

```

4. 计算(calc)

小明在你的帮助下,破密了 Ferrari 设的密码门,正要往前走,突然又出现了一个密码门,门上有一个算式,其中只有“(”,“)”,“0-9”,“+”,“-”,“*”,“/”,“^”,求出的值就是密码。小明数学学得不好,还需你帮他的忙。(“/”用整数除法)

输入:

输入文件 calc.in 共 1 行,为一个算式。

输出:

输出文件 calc.out 共 1 行,就是密码。

输入输出样例:

calc.in	calc.out
1+(3+2)*(7^2+6*9)/(2)	258

限制:

100%的数据满足:算式长度 ≤ 30 其中所有数据在 $2^{31}-1$ 的范围内。

【参考程序 1】

```

#include<stdio.h>
#include<string.h>
#include<stack>
using namespace std;
int f(char c)                //运算优先级判断

```

```

{
    if (c=='+' || c=='-') return 1;           //加减的优先级最低
    if (c=='*' || c=='/') return 2;          //乘除的优先级比加减高,比乘方低
    if (c=='^') return 3;                    //乘方的优先级最高
    if (c=='(') return 0;
}
void js(stack<int> &s1, stack<char> &s2) //退栈并计算当前结果
{
    int x,y,ans=1,i;
    char ch;
    x=s1.top(); s1.pop();
    y=s1.top(); s1.pop();                    //取出栈顶两个操作数
    ch=s2.top(); s2.pop();
    if(ch=='*') s1.push(y * x);
    if(ch=='/') s1.push(y / x);
    if(ch=='^')
    {
        for(int i=0; i < x; i++)
            ans *= y;                        //幂计算
        s1.push(ans);
    }
    if(ch=='+') s1.push(y+x);
    if(ch=='-') s1.push(y-x);                //将计算结果压入栈中
}
int main()
{
    int n;
    char s[50];
    scanf("%d\n", &n);
    while(n--)
    {
        stack<int> si;                       //c++自带的 STL 栈
        stack<char> sc;                       //建立操作数和运算符两个栈
        gets(s);
        int len=strlen(s), i, flag=0, temp=0;
        for(i=0; i < len; i++)
        {
            if(s[i] >='0' && s[i] <='9') //判断当前字符是否为数字
            {
                temp=temp * 10+s[i]-'0';
                flag=1;
            }
            else
            {
                if(flag) si.push(temp), temp=0, flag=0; //操作数入栈
                if(s[i]=='(') {sc.push(s[i]); continue;}
                if(s[i]==')')

```

```

        {
            while (sc.top() != '(') js(si,sc); //先计算括号中的表达式
            sc.pop();
            continue;
        }
    if(sc.empty()) sc.push(s[i]); //运算符栈入栈
    else
    {
        while(! sc.empty() && f(sc.top()) >= f(s[i])) js(si,sc);
            //计算优先级判断 出栈并计算
        sc.push(s[i]); //运算符栈入栈
    }
}
}
if(flag) si.push(temp), temp=0, flag=0;
while(! sc.empty()) js(si,sc);
printf("%d\n", si.top()); //输出栈顶的最终结果
si.pop();
}
}

```

【参考程序 2】

```

#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn=50;
int n;
int top;
int v[maxn];
char s[maxn], c[maxn]; //v 为操作数栈,c 为运算符栈
void Push(int x, int flag) //入栈 ,flag=0 表示运算符,flag=1 表示操作数
{
    if(! flag) c[++top]=x;
    else v[top]=x;
}
int Pow(int x, int n) //乘方计算
{
    int sum=1;
    for(int i=1; i<=n; ++i) sum *= x;
    return sum;
}
void Pop() //出栈并计算结果
{
    if(c[top]=='+') v[top-1]+=v[top];
}

```



```

if(c[top]=='-') v[top-1]-=v[top];
if(c[top]=='*') v[top-1]*=v[top];
if(c[top]=='/') v[top-1]/=v[top];
if(c[top]=='^') v[top-1]=Pow(v[top-1], v[top]);
//根据运算符计算

--top;
}
int Opr(char x) //运算优先级的判断
{
    if(x=='+' || x=='-') return 0;
    if(x=='*' || x=='/') return 1;
    if(x=='^') return 2;

    //优先级顺序:乘方>乘除>加减

    return -1;
}
bool Check(char x)
{
    if(Opr(x) <= Opr(c[top])) return true;
    return false;
}
int main()
{
    scanf("%s", s);
    n=strlen(s);
    s[n]='\0';
    Push('(', 0); //手写栈
    for(int i=0; i<=n; ++i)
    {
        while(s[i]=='(') Push(s[i++], 0);
        int x=0;
        while(isdigit(s[i])) //使用 STL 函数判断是否为数字
            x=x*10+s[i++]-'0';
        Push(x, 1); //操作数入栈
        while(true)
        {
            if(s[i]==')')
            {
                //先计算括号中的表达式
                while(c[top]!='(') Pop();
                v[top-1]=v[top-1];
            }
            else
            {
                while(Check(s[i])) Pop(); //根据优先级计算
                Push(s[i], 0); //运算符入栈
                break;
            }
            ++i;
        }
    }
}

```

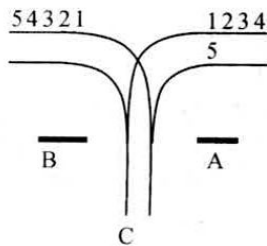
```

        if(i > n) break;
    }
}
printf("%d", v[0]);           //输出栈顶的最终计算结果
}

```

5. 车厢调度(train)

有一个火车站,铁路如图所示,每辆火车从 A 驶入,再从 B 方向驶出,同时它的车厢可以重新组合。假设从 A 方向驶来的火车有 n 节 ($n \leq 1000$),分别按照顺序编号为 $1, 2, 3, \dots, n$ 。假定在进入车站前,每节车厢之间都不是连着的,并且它们可以自行移动到 B 处的铁轨上。另外假定车站 C 可以停放任意多节车厢。但是一旦进入车站 C,它就不能再回到 A 方向的铁轨上了,并且一旦当它进入 B 方向的铁轨,它就不能再回到车站 C。



负责车厢调度的工作人员需要知道能否使它以 $a_1, a_2, \dots, a_n$ 的顺序从 B 方向驶出,请来判断能否得到指定的车厢顺序。

输入:

输入文件的第 1 行为一个整数 n ,其中 $n \leq 1000$,表示有 n 节车厢,第 2 行为 n 个数字,表示指定的车厢顺序。

输出:

如果可以得到指定的车厢顺序,则输出一个字符串"YES",否则输出"NO"(注意要大写,不包含引号)。

样例输入:

```

5
5 4 3 2 1

```

样例输出:

```

YES

```

解析:

观察发现,整个调度过程其实是在模拟入栈出栈的过程,而在这个过程中,我们可以分成三种状态:栈前、栈中、栈后。我们可以发现,当某个数字出栈了,说明比它小的数字要么已经出栈了,要么还在栈里,不能是入栈前状态,并且在栈中的顺序是从大到小的(从栈顶往栈底看),比如出 5,那么 $1, 2, 3, 4$ 要么已经在 5 之前出了,要么还在栈中(假如 $1, 3, 4$ 在栈中,从栈顶往栈底看依次为 $4, 3, 1$),不能是入栈前的状态。如果某个数字要出栈,那么当前在栈中的数字都必须小于它,否则就与栈的性质矛盾,不合法,于是我们可以这样解决:

从第一个数字开始扫描, $a[i]$ 表示当前出栈的数字,如果有比 $a[i]$ 大的数字还在栈中,那么就产生矛盾,输出"NO";否则,标记当前数字 $a[i]$ 为栈后状态,那么 $[1, a[i]-1]$ 这些数字如果还没出栈,标记为栈中状态。具体我们可以用 0 表示为确定状态,1 表示栈中状态,2 表示栈后状态。

【参考程序】

```

#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
int a[1005],instk[1005];
int main(){
    int n;
    scanf("%d",&n);
    for(int i=1; i<=n; i++){

```

```

scanf("%d",&a[i]);
instk[i]=0;           //0 表示未确定
}
int ok=1;             //用于判断是否有矛盾出现
for(int i=1; i<=n; i++){
    int x=a[i];
    for(int j=x+1; j<=n; j++){
        if(instk[j]==1) ok=0;
        //如果在栈中,则元素 x 不可能在第 i 次出栈,即不能得到指定的出栈顺序
    }
    instk[x]=2;        //2 表示出栈
    for(int j=1; j<x; j++){ //将 x 入栈必须将 x 前面的所有未确定的元素都入栈
        if(instk[j]==0) instk[j]=1;    //1 表示栈中
    }
}
if(ok) printf(" YES\n ");
else printf(" NO\n ");
return 0;
}

```

6. 中缀表达式值(expr)

输入一个中缀表达式(由 0~9 组成的运算数、加(+)、减(-)、乘(*)、除(/)四种运算符、左右小括号组成。注意“-”也可作为负数的标志,表达式以“@”作为结束符),判断表达式是否合法,如果不合法,请输出“NO”;否则请把表达式转换成后缀形式,再求出后缀表达式的值并输出。注意:必须用栈操作,不能直接输出表达式的值。

输入:

一行,为一个以@结束的字符串。

输出:

如果表达式不合法,则输出“NO”,要求大写。

如果表达式合法,则输出计算结果。

样例输入:

1+2×8-9

样例输出:

8

【参考程序 1】

```

#include<cstdio>
#include<algorithm>
#include<string>
#include<cstring>
#include<cstdlib>
#include<cmath>
#include<iostream>
#include<stack>
#define Int stack<int>
#define Char stack<char>
using namespace std;
int n,m;
Int s1;           //s1 为操作数栈,s2 为运算符栈

```

```

Char s2;                                //c++自带的 STL 栈
inline int lev(char a)                  //运算符优先级
{
    if(a=='+'||a=='-') return 1;
    if(a=='*'||a=='/') return 2;
    if(a=='^') return 3;
    return 0;
}
inline int ksm(int s,int t)
{
    int res=1;
    while(t)                            //快速幂
    {
        if(t&1) res *= s;
        s *= s;
        t >>= 1;
    }
    return res;
}
inline void js(Int &s1,Char &s2)        //出栈并计算
{
    int y=s1.top();s1.pop();
    int x=s1.top();s1.pop();
    char z=s2.top();s2.pop();          //取出后弹出栈
    if(z=='+') s1.push(x+y);           //根据运算符计算
    if(z=='-') s1.push(x-y);
    if(z=='*') s1.push(x*y);
    if(z=='/') s1.push(x/y);
    if(z=='^') s1.push(ksm(x,y));      //将运算结果压入栈中
}
inline int c(int x)
{
    return x! =0;
}
char sr[1000000];
int sum[1000000];
int main()
{
    scanf("%s",sr+1);
    n=strlen(sr+1)-1;                  //-1 是为了忽略掉读入中的 '@'
    for(int i=1;i<=n;++i)
    {
        sum[i]+=sum[i-1];
        if(sr[i]=='(') ++sum[i];
        if(sr[i]==')') --sum[i];      //检查匹配
    }
    bool out=0;
    for(int i=2;i<=n;++i)

```

```

if(c(lev(sr[i]))&&.c(lev(sr[i-1]))) {out=1;break;}
if((n==1&&.c(lev(sr[1])))||sum[n]||out)
{
    printf(" NO ");
    return 0;
}
Int s1;
Char s2;
int tmp=0;
bool flag=0;
for(int i=1;i<=n;++i)
{
    if('0'<=sr[i]&&.sr[i]<='9')
    {
        //判断当前字符是否为数字
        tmp=(tmp<<3)+(tmp<<1)+sr[i]-'0';
        flag=1;
    }
    else
    {
        if(flag) {s1.push(tmp);tmp=flag=0;}
        if(sr[i]=='(') {s2.push(sr[i]);continue;}
        if(sr[i]==')')
        {
            while(s2.top()!= '(') js(s1,s2); //先进行括号中的运算
            s2.pop();
            continue;
        }
        while(! s2.empty()&&.lev(s2.top())>=lev(sr[i])) //运算优先级的判断
            js(s1,s2);
        s2.push(sr[i]); //运算符入栈
    }
}
if(flag) {s1.push(tmp);tmp=flag=0;}
while(! s2.empty()) js(s1,s2);
printf("%d ",s1.top()); //将栈顶的最终结果输出
}

```

【参考程序 2】

```

#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn=50;
int n;
int top;
int v[maxn];
char s[maxn], c[maxn];
//v 为操作数栈,c 为运算符栈

```

```

void Push(int x, int flag)                //入栈 ,flag=0 为运算符入栈,否则操作数入栈
{
    if(! flag) c[++top]=x;
    else v[top]=x;
}
int Pow(int x, int n)                    //乘方
{
    int sum=1;
    for(int i=1; i <=n; ++i) sum *=x;
    return sum;
}
void Pop()                               //弹出栈顶元素并计算
{
    if(top <=0)
    {
        printf(" NO ");
        exit(0);
    }

    //根据运算符计算
    if(c[top]=='+') v[top-1]+=v[top];
    if(c[top]=='-') v[top-1]-=v[top];
    if(c[top]=='*') v[top-1] *=v[top];
    if(c[top]=='/') v[top-1] /=v[top];
    if(c[top]=='^') v[top-1]=Pow(v[top-1], v[top]);
    --top;
}
int Opr(char x)
{
    if(x=='+' || x=='-') return 0;
    if(x=='*' || x=='/') return 1;
    if(x=='^') return 2;
    return -1;
}
bool Check(char x)                      //符号的优先级判断
{
    if(Opr(x) <=Opr(c[top])) return true;
    return false;
}
int main()
{
    scanf("%s", s);
    n=strlen(s)-1;
    s[n]='\0';
    Push('(', 0);
    for(int i=0; i <=n; ++i)
    {
        while(s[i]=='(') Push(s[i++], 0);
        if(! isdigit(s[i]))
        {

```

```
    printf(" NO ");
    return 0;
}
int x=0;
while(isdigit(s[i]))                //使用 c++ STL 中函数判断是否数字
x=x * 10+s[i++]-'0';
Push(x, 1);                        //操作数入栈
while(true)
{
    if(s[i]=='')
    {
        while(c[top] != '(') Pop(); //先计算括号中的表达式
        v[top-1]=v[top--];
    }
    else
    {
        while(Check(s[i])) Pop();   //优先级判断
        Push(s[i], 0);              //运算符入栈
        break;
    }
    ++i;
    if(i > n) break;
}
}
if(top > 0) printf(" NO ");          //表达式不合法
else printf("%d ", v[0]);
}
```

第二章 队列

1. 面积(area)

编程计算由“*”号围成的下列图形的面积。面积计算方法是统计*号所围成的闭合曲线中水平线和垂直线交点的数目。如下图所示,在 10×10 的二维数组中,有“*”围住了15个点,因此面积为15。

样例输入:

```
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 0 0 0
0 0 0 0 1 0 0 1 0 0
0 0 0 0 0 1 0 0 1 0
0 0 1 0 0 0 1 0 1 0
0 1 0 1 0 1 0 0 1 0
0 1 0 0 1 1 0 1 1 0
0 0 1 0 0 0 0 1 0 0
0 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0
```

样例输出:

15

【参考程序】

```
#include<stdio.h>
#include<string.h>
#define MAXN 10
#define MAXM 10
int BFS_II_I(int,int);
int map[12][12],qx[200],qy[200];
int main(void)
{
    int i,j,ans=0;
    memset(map,255,sizeof(map));
    for (i=1;i<=MAXN;i++)
    {
        for (j=1;j<=MAXM;j++)
        {
            scanf("%d",&map[i][j]);
            map[i][j]=-map[i][j]; //map[i][j]表示 i,j 已经访问过或者不能访问
        }
    }
    for (i=1;i<=MAXN;i++) //按矩阵的边缘搜索
    {
        if (0==map[i][1]) BFS_II_I(i,1); //只要不在1围成的圈内
        if (0==map[i][MAXM]) BFS_II_I(i,MAXM);
        //就一定和边缘的其中一个0在同一块全部为0的区域内
        if (0==map[1][i]) BFS_II_I(1,i);
    }
}
```



```

    if (0 == map[MAXM][i]) BFS_II_I(MAXM, i);
}
for (i = 1; i <= MAXN; i++)
    for (j = 1; j <= MAXM; j++)
        if (0 == map[i][j]) ans++; //还没被标记的,就是在圈内的0了
printf("%d\n", ans);
return 0;
}

int BFS_II_I(int x, int y) //从 x, y 开始进行广搜
{
    const int gx[4] = {0, 0, 1, -1}; //上下左右移动后 x 坐标增量
    const int gy[4] = {1, -1, 0, 0}; //y 坐标增量
    int head = 1, tail = 0, tx, ty, count = 0;
    map[x][y] = -1;
    qx[head] = x;
    qy[head] = y;
    for (; head != tail; )
    {
        count++; //找到了一个结点
        tail++;
        tx = qx[tail];
        ty = qy[tail]; //队头出队
        for (int i = 0; i < 4; i++) //沿着队头的上、下、左、右四个方向搜索
        {
            x = tx + gx[i];
            y = ty + gy[i];
            if (-1 == map[x][y]) continue; //判断是否没被标记过,且原来不是 1
            map[x][y] = -1; //标记
            head++;
            qx[head] = x;
            qy[head] = y; //把当前点放入队尾
        }
    }
    return count;
}

```

2. 奇怪的电梯(lift)

大楼的每一层楼都可以停电梯,而且第 i 层楼($1 \leq i \leq N$)上有一个数字 K_i ($0 \leq K_i \leq N$)。电梯只有四个按钮:开,关,上,下。上下的层数等于当前楼层上的那个数字。当然,如果不能满足要求,相应的按钮就会失灵。例如:3 3 1 2 5 代表了 K_i ($K_1=3, K_2=3, \dots$),从一楼开始。在一楼,按“上”可以到4楼,按“下”是不起作用的,因为没有-2楼。那么,从A楼到B楼至少要按几次按钮呢?

输入:

共有两行,第1行为三个用空格隔开的正整数,表示 N, A, B ($1 \leq N \leq 200, 1 \leq A, B \leq N$),第2行为 N 个用空格隔开的正整数,表示 K_i 。

输出:

仅一行,即最少按键次数,若无法到达,则输出-1。

样例输入:

5 1 5
3 3 1 2 5

样例输出:

3

【参考程序 1】

```
#include<iostream>
#include<cstring>
using namespace std;
int q[32768],k[32768],a[32768];    //q 为第几层,a 为第几步
bool b[32768];                    //标记
int main()
{
    memset(b,false,sizeof(b));
    int x,y;
    int n;
    cin>>n>>x>>y;
    for(int i=1;i<=n;i++)
        cin>>k[i];
    q[1]=x;                        //队头入队,标记
    b[x]=true;
    int h=0,t=1;
    do                            //广搜
    {
        h++;                      //队首元素出队
        for(int i=-1;i<=1;i++)    //上下两个方向
        {
            if((q[h]+k[q[h]]*i>0&& q[h]+k[q[h]]*i<=n)&&! b[q[h]+k[q[h]]*i])
            {
                //如果走了这一步,没有越界,并且没有走到已经走过的位置
                t++;              //新元素入队
                q[t]=q[h]+k[q[h]]*i; //当前位置入队
                b[q[h]+k[q[h]]*i]=true; //标记
                a[t]=a[h]+1;       //步数加 1
            }
        }
    } while(h!=t&& q[h]!=y);      //队列未空且未到达终点
    if(x==y)cout<<"0 "<<endl;    //起点=终点,直接输出 0
    else
        if(q[h]==y)cout<<a[h]<<endl; //能走到,输出答案
    else
        cout<<"-1 "<<endl;       //不能走到
    return 0;
}
```

【参考程序 2】

```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
```

```

#include <algorithm>
#include <cstdio>
using namespace std;
bool bo[201];
int s[201], h[40001][2], t=0, w=1, n, a, b;
int k[201];
//n,a,b,k 的含义和题意一致
//h 是队列, h[i][0], h[i][1] 分别表示队列中第 i 个元素的按键次数和到达层数,
//t, w 分别为队列的头尾指针, bo[i] 表示第 i 层是否能到达
//s[i] 表示到达 i 层的最少按键次数 (若为 -1 表示到达不了)
int main()
{
    scanf("%d%d%d", &n, &a, &b);
    for (int i=1; i <= n; ++i)
        scanf("%d", &k[i]);
    memset(s, -1, sizeof(s)); //初始化, 将 s 数组都赋成 -1
    memset(bo, false, sizeof(bo)); //将 bo 数组初始化为假, 即刚开始都到达不了
    h[1][1] = a;
    h[1][0] = 0; //将起点入队, 步数为 0
    s[a] = 0;
    bo[a] = true; //因为起点就是 a, 所以 a 可以到达, 并且所需步数为 0
    while (t < w && ! bo[b]) //判断头指针小于尾指针 (队列不为空) 且还未到达终点
    {
        t++; //将头指针指向要出队的元素
        int x = h[t][1] + k[h[t][1]];
        int y = h[t][1] - k[h[t][1]]; //x, y 分别表示按上、下到达的层数
        //广度优先搜索, 第一次访问到的步数必然最少, 也就不需要访问第二次
        if (x > 0 && x <= n && ! bo[x])
        {
            //如果没有越界, 且 x 没到过
            w++;
            bo[x] = 1; //标记到过
            s[x] = h[t][0] + 1; //记录走到过和需要按键次数
            h[w][1] = x;
            h[w][0] = s[x]; //将该层入队
        }
        if (y > 0 && y <= n && ! bo[y]) //对于 y 同理
        {
            w++;
            bo[y] = 1;
            s[y] = h[t][0] + 1;
            h[w][1] = y;
            h[w][0] = s[y];
        }
    }
    printf("%d", s[b]); //输出到达终点需要的最少按键次数
    return 0;
}

```

3. 产生数(Produce)

给出一个整数 $n(n \leq 2000)$ 和 k 个变换规则($k \leq 15$)。规则:

(1) 1 个数字可以变换成另 1 个数字;

(2) 规则中, 右边的数字不能为零。

例如: $n=234, k=2$ 规则为:

$2 \rightarrow 5$

$3 \rightarrow 6$

上面的整数 234 经过变换后可能产生出的整数为(包括原数)234, 534, 264, 564 共 4 种不同的产生数。

求经过任意次的变换(0 次或多次), 能产生出多少个不同的整数。仅要求输出不同整数个数。

输入:

n

k

$x_1 y_1$

$x_2 y_2$

... ..

$x_n y_n$

输出:

一个整数(满足条件的整数个数)。

样例输入:

2 3 4

2

2 5

3 6

样例输出:

4

【参考程序】

```
#include<cstdio>
```

```
#define MAXN 10001
```

```
#define MAXM 16
```

```
int n,k;
```

```
int exist[MAXN];
```

```
int que[MAXN];
```

```
int a[MAXM],b[MAXM];
```

```
int main(void)
```

```
{
```

```
    scanf("%d",&n);
```

```
    scanf("%d",&k);
```

```
    for (int i=0;i<k;i++)
```

```
        scanf("%d%d",&a[i],&b[i]);
```

```
    exist[n]=1;
```

```
    que[1]=n;
```

```
    int head=1;
```

```
    for (int tail=0;head>=tail;)
```

```
    {
```

```
        int x=que[++tail];
```

//第一个数标记

//第一个数入队,它也算一个产生数

//利用广度优先搜索计算

```

int y=x,mod=1;
while (x>0)                                //把队头数的每一位都试着进行变换
{
    int tmp=x%10;                            //取出最后一位
    x/=10;
    for (int i=0;i<k;i++)
        if (a[i]==tmp)                      //这一位数可以变换
        {
            int p=y+(b[i]-tmp)*mod; //p 为变换后的数
            if (exist[p]==0)        //判断是否重复
            {
                exist[p]=1;          //标记
                que[++head]=p;       //把变换后的数存入队列
            }
        }
    mod*=10;
}
printf("%d\n",head);                        //队尾即是答案
return 0;
}

```

4. 家庭问题(family)

有 n 个人,编号为 $1,2,\dots,n$,另外还知道存在 K 个关系。一个关系的表达为二元数组 (α,β) 形式,表示 α,β 为同一家庭的成员。

当 n,k 和 k 个关系给出之后,求出其中共有多少个家庭、最大的家庭中有多少人。

例如: $n=6,k=3$,三个关系为 $(1,2),(1,3),(4,5)$

此时,6 个人组成三个家庭,即: $\{1,2,3\}$ 为一个家庭, $\{4,5\}$ 为一个家庭, $\{6\}$ 单独为一个家庭,第一个家庭的人数为最多。

输入:

文件的第 1 行为 n,k 两个整数($1 \leq n \leq 100$)(用空格分隔);

接下来的 k 行,每行两个整数(用空格分隔)表示关系。

输出:

两个整数(分别表示家庭个数和最大家庭人数)

样例输入:

```

6 3
1 2
1 3
4 5

```

样例输出:

```

3 3

```

【参考程序 1】

```

#include<cstdio>
#include<algorithm>
#include<string>
#include<cstring>
#include<cstdlib>
#include<cmath>

```

```
#include<iostream>
using namespace std;
const int N=23333;
int n,m,t,w;
int h[N];
inline int read()                //读入优化
{
    char c;
    bool flag=false;
    while((c=getchar())>'9' || c<'0')
        if(c=='-')flag=true;
    int res=c-'0';
    while((c=getchar())>='0' && c<='9')
        res=(res<<3)+(res<<1)+c-'0';
    return flag? -res : res;
}
int tot,ans,ans_;
int nex[N],go[N],fir[N];
bool vis[N];
inline void add(int x,int y)      //邻接表
{
    nex[++tot]=fir[x];fir[x]=tot;go[tot]=y;
    nex[++tot]=fir[y];fir[y]=tot;go[tot]=x;
}
int main()
{
    freopen("family.in","r",stdin);
    freopen("family.out","w",stdout);
    n=read();
    m=read();
    for(int i=1;i<=m;++i)
        add(read(),read());      //邻接表存图,将有关系的人两两建一条双向边
    for(int i=1;i<=n;++i)
        if(! vis[i])              //对于每一个点进行一次广搜
        {
            t=0;
            h[w=1]=i;
            vis[i]=1;
            while(t<w)              //广搜求解
            {
                int u=h[++t];
                for(int e=fir[u],v;v=go[e],e=e=nex[e])
                    if(! vis[v])    //判断该点是否访问过
                    {
                        vis[v]=1;    //标记为已访问过
                        h[++w]=v;
                    }
            }
        }
}
```

```

    }
    ans++;
    ans_ = max(ans_, w);
}
printf("%d %d ", ans, ans_);
}

```

【参考程序 2】

```

#include <cmath>
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
const int MAXN = 105;
int q[MAXN];
int n, m, i, j, x, y, z, tot, maxmember;
bool vis[MAXN], g[MAXN][MAXN];
inline void bfs(int x)
{
    int w = 1;
    q[w] = x; vis[x] = 1;
    for(int j = 1; j <= w; j++) //广搜求解
    {
        int k = q[j];
        for(int i = 1; i <= n; i++)
            if (k != i && g[k][i] && ! vis[i])
            {
                q[++w] = i;
                vis[i] = 1; //标记已经访问过
            }
    }
    maxmember = max(maxmember, w); //更新最大家庭人数
}

int main()
{
    scanf("%d%d", &n, &m);
    for(i = 1; i <= m; i++)
    {
        scanf("%d%d", &x, &y);
        g[x][y] = g[y][x] = 1; //邻接矩阵存图,将有关系的人两两连一条双向边
    }
    for(i = 1; i <= n; i++)
        if (! vis[i]) bfs(i), tot++; //对于每一个点,都做一次广搜,否则不能遍历到所有的点
    cout << tot << " " << maxmember << endl;
}

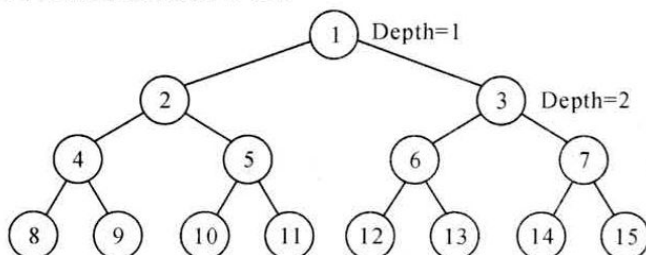
```

第三章 树

第一节 二叉树

1. 小球(drop)

许多的小球一个一个的从一棵满二叉树上掉下来组成 FBT(Full Binary Tree, 满二叉树), 每一时间, 一个正在下降的球第一个访问的是非叶子节点。然后继续下降时, 或者走右子树, 或者走左子树, 直到访问到叶子节点。决定球运动方向的是每个节点的布尔值。最初, 所有的节点都是 false, 当访问到一个节点时, 如果这个节点是 false, 则这个球把它变成 true, 然后从左子树走, 继续它的旅程。如果节点是 true, 则球也会改变它为 false, 而接下来从右子树走。满二叉树的标记方法如下图:



因为所有的节点最初为 false, 所以第一个球将会访问节点 1, 节点 2 和节点 4, 转变节点的布尔值后在节点 8 停止。第二个球将会访问节点 1, 3, 6, 在节点 12 停止。明显地, 第三个球在它停止之前, 会访问节点 1, 2, 5, 在节点 10 停止。

现在你的任务是, 给定 FBT 的深度 D, 和 I, 表示第 I 个小球下落, 你可以假定 I 不超过给定的 FBT 的叶子数, 写一个程序求小球停止时的叶子序号。

输入:

仅一行, 包含两个用空格隔开的整数 D 和 I ($2 \leq D \leq 20, 1 \leq I \leq 524288$)。

输出:

对应输出第 I 个小球下落停止时的叶子序号。

样例输入:

4 2

样例输出:

12

【参考程序 1】

```
#include <iostream>
using namespace std;
int n, k, p;
bool tree[1<<20];
void cyx() //从顶部到底部一次掉落
{
    int i; p=1;
    for(i=2; i<=n; i++)
        if(tree[p]) {tree[p]=0; p=p*2+1;} //往右子树走
        else {tree[p]=1; p*=2;} //往左子树走
}
```



```

}
int main()
{
    int i;
    cin>>n>>k;
    for(i=1;i<=k;i++) cyx();           //一共 k 次掉落
    cout<<p;
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstring>
using namespace std;
bool a[524289];
int main()
{
    memset(a,false,sizeof(a));
    int d,i,k;
    cin>>d>>i;
    for (int j=1;j<=i;++j)           //暴力枚举每一个球的下落情况
    {
        k=1;
        for (int l=1;l<d;++l)
        {
            if (a[k]==false)           //结点的值为 0
            {
                a[k]=true;
                k*=2;                   //向左走
            }
            else                       //结点的值为 1
            {
                a[k]=false;
                k=k*2+1;               //向右走
            }
        }
    }
    cout<<k<<endl;
    return 0;
}

```

【参考程序 3】

```

#include<iostream>
#include<cstdio>
#include<algorithm>
using namespace std;
int D, I, k=1;
int main()
{

```

```

cin >> D >> I;
for (int i=0; i<D-1; i++)           //走 D 次
{ //完全二叉树的其中一个结点为 n,则它的左孩子为 2 * n,右孩子为 2 * n+1
  if (I % 2==0)                      //判断走左子树还是右子树
  {
    k=k * 2+1;
    I /=2;
  }
  else
  {
    k *=2;
    I=(I+1) / 2;
  }
}
cout << k;
}

```

2. 二叉树遍历(flist)

树和二叉树基本上都有先序、中序、后序、按层遍历等遍历顺序,给定中序和其他一种遍历的序列就可以确定一棵二叉树的结构。

假定一棵二叉树一个结点用一个字符描述,现在给出中序和按层遍历的字符串,求该树的先序遍历字符串。

输入:

输入文件 flist.in 共两行,每行是由字母组成的字符串(一行的每个字符都是唯一的),分别表示二叉树的中序遍历和按层遍历的序列。

输出:

输出文件 flist.out 就一行,表示二叉树的先序序列。

样例输入:

DBEAC
ABCDE

样例输出:

ABDEC

【参考程序 1】

```

#include<stdio>
#include<cstring>

int map[255];           //map[ch]表示 ch 在按层遍历中的位置
char s1[255],s2[255];
int l;

void make_tree(int l,int r)           //l 和 r 为在中序遍历中的遍历范围
{
  if (l>r) return;                   //如果访问到空子树,要跳出
  int k=0x7ffff,where;               //按层遍历中,根结点必然比左右儿子靠前
  for (int i=l;i<=r;i++)
  { //在中序遍历的[l,r]中找到一个在按层遍历里最靠前的,即为根
    if (map[s1[i]]<k)                //找到在按层遍历中最靠前的,即为根

```

```

    {
        k=map[s1[i]];
        where=i;                //记录编号
    }

    putchar(s1[where]); //由于是输出先序遍历,刚好和搜索的顺序相同,直接输出
    make_tree(l,where-1); //l~where-1 即为中序遍历的左子树范围
    make_tree(where+1,r); //where+1~n 即为中序遍历的右子树范围
}

int main(void)
{
    gets(s1);
    gets(s2);
    l=strlen(s1);                //记录树的结点总数
    for (int i=0;i<l;i++)
        map[s2[i]]=i;           //记录字符在 s2 中的位置
    make_tree(0,l-1);
    putchar('\n');
    return 0;
}

```

【参考程序 2】

```

#include <string>
#include <iostream>
using namespace std;
string cyf,lzy;                //cyf 为中序遍历,lzy 为按层遍历
void pre(int cyx,int lcx,int gsy) //先序遍历输出
{
    //cyx 和 lcx 分别为 cyf 数组中的左边界和右边界,gsy 为按层遍历中的根位置
    int i,j,root,d1=-1,d2=-1;
    for(i=cyx;i<=lcx;i++)
        if(cyf[i]==lzy[gsy])
            {root=i;break;} //找树根
    cout<<lzy[gsy];          //输出该节点的值
    for(i=gsy+1;i<=lzy.size()-1;i++)
    { //按层遍历子节点一定在序列中当前节点的右边
        if(d1<0 && cyx<root)
        {
            for(j=cyx;j<=root-1;j++)
                if(cyf[j]==lzy[i]) //找左子树的根
                    { d1=i;break; }
        }
        if(d2<0 && lcx>root)
        {
            for(j=root+1;j<=lcx;j++)
                if(cyf[j]==lzy[i]) //找右子树的根
                    { d2=i;break; }
        }
    }
}

```

```

        if(d1>0 && d2>0) break;
    }
    if(cyx<root) pre(cyx,root-1,d1); //递归遍历左子树
    if(lcx>root) pre(root+1,lcx,d2); //递归遍历右子树
}
int main()
{
    cin>>cyf>>lzy;
    pre(0,cyf.size()-1,0);
    return 0;
}

```

3. FBI 树(fbi)

我们可以把由“0”和“1”组成的字符串分为三类:全“0”串称为 B 串,全“1”串称为 I 串,既含“0”又含“1”的串则称为 F 串。

FBI 树是一种二叉树,它的结点类型也包括 F 结点,B 结点和 I 结点三种。由一个长度为 2^N 的“01”串 S 可以构造出一棵 FBI 树 T,递归的构造方法如下:

T 的根结点为 R,其类型与串 S 的类型相同;

若串 S 的长度大于 1,将串 S 从中间分开,分为等长的左右子串 S1 和 S2;由左子串 S1 构造 R 的左子树 T1,由右子串 S2 构造 R 的右子树 T2。

现在给定一个长度为 2^N 的“01”串,请用上述构造方法构造出一棵 FBI 树,并输出它的后序遍历序列。

输入:

输入文件 fbi.in 的第 1 行是一个整数 $N(0 \leq N \leq 10)$,第 2 行是一个长度为 2^N 的“01”串。

输出:

输出文件 fbi.out 包括一行,这一行只包含一个字符串,即 FBI 树的后序遍历序列。

样例输入:

```

3
10001011

```

样例输出:

```

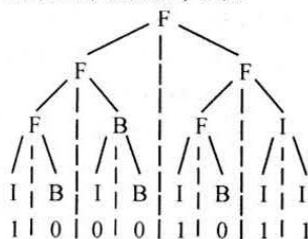
IBFBBBF1BF111FF

```

数据规模:

对于 40% 的数据, $N \leq 2$;

对于 100% 的数据, $N \leq 10$ 。



【参考程序 1】

//两种算法:dfs,逆推

```
#include <map>
```

```
//dfs
```

```
#include <cmath>
```

```
#include <vector>
```

```
#include <cstdio>
```

```
#include <string>
```

```
#include <cstdlib>
```

```
#include <cstring>
```

```
#include <iostream>
```

```
#include <algorithm>
```

```
using namespace std;
```

```
int n;
```

```
char tree[10001], s[10001];
//tree[i]表示结点 i 的类型, s 则是输入的字符串
//把二叉树按层次遍历的顺序依次编号为:1,2,3,...,2n+1-1(该树共有(n+1)层)
//不难看出,这棵树是满二叉树(也是完全二叉树),所以结点 x 的左右儿子编号分别为:
2 * x, 2 * x + 1
```

```
void dfs(int x)
{
    if (x >= 1 << n)
    { //如果是最后一层的结点,直接根据题意从字符串中确定结点的类型
        if (s[x] == '1')
            tree[x] = 'I';
        else
            tree[x] = 'B';

        putchar(tree[x]); //输出后序遍历,最后一层没有孩子,直接把 tree[x] 输出
        return;
    }

    dfs(x * 2);
    dfs(x * 2 + 1); //通过先确定左右儿子的类型,从而确定该节点的类型

    if (tree[x * 2] == tree[x * 2 + 1])
        //如果全是 1 或全是 0 或左右儿子类型都是 F(即左右结点类型相同)
        tree[x] = tree[x * 2]; //那么该节点的类型也和左右儿子相同(只取左儿子即可)
    else
        tree[x] = 'F'; //否则说明两边有 1 也有 0,则该结点类型为 F
    putchar(tree[x]); //dfs 访问的顺序刚好和后序遍历相同,则在 dfs 时就输出答案
    return;
}

int main()
{
    scanf("%d", &n);
    for (int i = 1 << n; i < 1 << (n + 1); ++i)
        cin >> s[i];
    //说明:这棵树的总结点数为 2n+1-1, 即 1 << (n + 1) - 1
    //这里直接把字符串读到 s[2n+1-1]的位置,也就是二叉树最后一层编号的位置,方便
    接下来的操作
    dfs(1); //用 dfs 的方法逐渐确定结点的类型
    putchar('\n');
    return 0;
}
```

```

/*
#include <map>                                //逆推
#include <cmath>
#include <vector>
#include <cstdio>
#include <string>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
//把二叉树按层次遍历的顺序依次编号为:1,2,3,...,2n+1-1(该树共有(n+1)层)
//不难看出,这棵树是满二叉树(也是完全二叉树),所以结点 x 的左右儿子编号分别为:
2 * x, 2 * x + 1
char tree[3001], ch;                        //tree[i]表示结点 i 的类型
int lc[3001], rc[3001];                    //lc[i],rc[i]分别表示结点 i 的左右儿子编号
int n, m;

void print(int k)                            //后序遍历输出答案
{
    if (k == 0) return;                    //如果是空树就不输出
    print(lc[k]);
    print(rc[k]);
    putchar(tree[k]);                      //先访问左右儿子,再访问根,输出答案
}

int main()
{
    scanf("%d", &n);

    m = 1 << n; //m 储存的是二叉树最后一层的第一个结点的编号(该树共有(n+1)层)
    //前 n 层结点数为 2n-1,那么 m 就是 2n-1+1=2n,即 1<=m<=2n
    //那么这一整棵树的结点数就为 2n+1-1,也就是 m * 2 + 1
    for (int i = m; i <= m * 2 - 1; ++i)
        //先根据题意确定最后一层结点的类型
    {
        cin >> ch;
        if (ch == '1')                    //叶子结点根据字符确定该结点的类型
            tree[i] = 'I';
        else
            tree[i] = 'B';
    }
    for (int i = m - 1; i >= 1; --i)        //其次从下往上依次确定剩余结点的类型
    {
        lc[i] = i * 2;
        rc[i] = i * 2 + 1;                //结点 i 的左右儿子编号分别为:2 * i, 2 * i + 1
    }
}

```

```

char x, y;
x=tree[i * 2];
y=tree[i * 2+1];
if (x=='F' || y=='F')
    tree[i]='F';
else if (x != y)
    tree[i]='F';
else
    tree[i]=x;
}
print(1);
putchar('\n');
return 0;
}

```

//x,y 分别为左右儿子的类型
//左右儿子都为 F 或不相同则为 F
//根据左右儿子的类型推出结点 i 的类型
//后序遍历输出答案

【参考程序 2】

```

#include <string>
#include <iostream>
using namespace std;
int n,p;
string s;
char tree[65536];
void cyx(int i)
{
    if(i * 2 <= p * 2 - 1) cyx(i * 2);
    if(i * 2 + 1 <= p * 2 - 1) cyx(i * 2 + 1);
    cout << tree[i];
}
int main()
{
    int i,j;
    cin >> n >> s;
    p = 1 << n;
    for(i=0; i<p; i++)
        if(s[i]=='0') tree[i+p]='B'; //i+p 对应第 i 个叶子节点
        else tree[i+p]='I';
    for(i=n; i>=1; i--)
        for(j=(1<<(i-1)); j<=(1<<i)-1; j++) //从倒数第二层到第一层往上走
            if(tree[j * 2] == tree[j * 2 + 1]) //两个子节点相等则与子节点相等
                tree[j] = tree[j * 2];
            else tree[j] = 'F';
    cyx(1);
    return 0;
}

```

//后序遍历输出
//1<<n 即 2ⁿ
//不相等则为 F
//输出

4. 二叉树输出(btout)

树的凹入表示法主要用于树的屏幕或打印输出,其表示的基本思想是兄弟间等长,一个结点的长度要不小于其子结点的长度。二叉树也可以这样表示,假设叶结点的长度为 1,一

个非叶结点的长度等于它的左右子树的长度之和。

一棵二叉树的一个结点用一个字母表示(无重复),输出时从根结点开始:

每行输出若干个结点字符(相同字符的个数等于该结点长度),

如果该结点有左子树就递归输出左子树;

如果该结点有右子树就递归输出右子树。

假定一棵二叉树一个结点用一个字符描述,现在给出先序和中序遍历的字符串,用树的凹入表示法输出该二叉树。

输入:

输入文件 btout.in 共两行,每行是由字母组成的字符串(一行的每个字符都是唯一的),分别表示二叉树的先序遍历和中序遍历的序列。

输出:

输出文件 btout.out 的行数等于该树的结点数,每行的字母相同。

样例输入:

ABCDEFGF

CBDAFEG

样例输出:

AAAA

BB

C

D

EE

F

G

【参考程序 1】

```
#include <map>
#include <cmath>
#include <vector>
#include <cstdio>
#include <string>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <algorithm>
using namespace std;
int i=-1, a[200];
string s1, s2;
int tree(int l, int r)
{
    //按先序遍历的顺序找,则找的结点和 s1 中的每个字符是一一对应的关系
    int tot=0, k=++i;
    int m=s2.find(s1[k]); //在中序遍历中查找这个字符,则它就为根,它在 s2 的左边部分即为左子树部分,右边部分即为右子树部分
    if (m>l) tot+=tree(l, m-1);
    if (m<r) tot+=tree(m+1, r); //递归先确定左右儿子的长度,再推出根的长度
    if (l==r) tot=1; //如果是叶结点,长度为 1
    a[s1[k]]=tot; //记录下答案
    return tot;
}
```



```

int main()
{
    cin >> s1;                //s1 表示先序遍历
    cin >> s2;                //s2 表示中序遍历
    tree(0, s1.size()-1);    //两个字符串的长度是一样的,所以只要取一个的长度即可
    for (int i=0; i < s1.size(); ++i)
    {
        for (int j=1; j <= a[s1[i]]; ++j)
            putchar(s1[i]);    //s1 恰好就是先序遍历的顺序,按这个顺序输出即可
        putchar('\n');
    }
    return 0;
}

```

【参考程序 2】

```

#include <string>
#include <iostream>
using namespace std;
string pr,ir;
int pre[100001],in[100001],res[100001];
struct node {char data;int left,right;};
node tree[6001];
int cyx(int hxs,int cyf,int lzy,int lcx)    //建树
{
    int i,root=lzy,tmp;
    for(i=lzy;i<=lcx;i++)
        if(in[i]==pre[hxs])
            {root=i;break;}    //找树根
    tree[in[root]].data=pr[in[root]];
    tree[in[root]].left=-1;
    tree[in[root]].right=-1;
    if(root>lzy)
        tree[in[root]].left=
            cyx(hxs+1,hxs+root-lzy,lzy,root-1);    //遍历左子树
    if(root<lcx)
        tree[in[root]].right=
            cyx(hxs+root-lzy+1,cyf,root+1,lcx);    //遍历右子树
    return in[root];
}
int tr(int k)    //计算每一个节点的长度
{
    if(tree[k].left==-1 && tree[k].right==-1)
        { res[k]=1;return 1; }    //叶节点的长度为 1
    int l=0,r=0;
    if(tree[k].left>=0) l=tr(tree[k].left);    //遍历左子树
    if(tree[k].right>=0) r=tr(tree[k].right);    //遍历右子树
    res[k]=l+r;return l+r;    //非叶节点的长度为左右子树的长度和
}

```

```
int main()
{
    int i,j;
    cin>>pr>>ir;
    for(i=0;i<pr.size();i++)
    {
        int k=0;
        for(j=0;j<pr.size();j++)
            if(pr[i]==ir[j])
                { k=j;break; }
        pre[i]=i;in[k]=i;
    }
    cyx(0,pr.size()-1,0,ir.size()-1);
    tr(0);
    for(i=0;i<pr.size();i++)                //输出
    {
        for(j=1;j<=res[i];j++)                //长度是几就输出几个字符
            cout<<pr[i];cout<<endl;
    }
    return 0;
}
```

5. 查找二叉树(tree_a)

已知一棵二叉树用邻接表结构存储,中序查找二叉树中值为 x 的结点,并指出是第几个结点。

输入:

第 1 行 n 为二叉树的结点个数, $n \leq 100$; 第 2 行 x 表示要查找的结点的值; 以下第 1 列数据是各结点的值, 第 2 列数据是左儿子结点编号, 第 3 列数据是右儿子结点编号。

输出:

输出一个数即查找的结点编号。

样例输入:

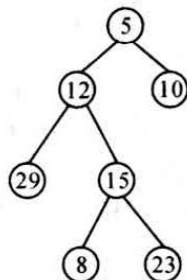
```
7
15
5 2 3
12 4 5
10 0 0
29 0 0
15 6 7
8 0 0
23 0 0
```

样例输出:

```
4
```

【参考程序 1】

```
#include <cmath>
#include <cstdio>
#include <cstring>
#include <cstdlib>
```



```

#include <iostream>
#include <algorithm>
using namespace std;
const int Maxn=110;
int lchild[Maxn], rchild[Maxn], number[Maxn], Answer;
//lchild[i],rchild[i],number[i]表示第 i 个结点的左儿子编号,右儿子编号,结点的值
int n, need; //need 即为题目中的需要查找的值 x
void find(int x) //x 表示目前查找到的结点
{
    //由于是中序查找,所以要按“左儿子、根、右儿子”的顺序遍历
    if (lchild[x]) find(lchild[x]);
    Answer++; //题目要求统计是查询到第几个结点,每次遍历到某个结点时把总个数加一
    if (number[x]==need) //如果此节点的值即为题目要求的值
    {
        printf("%d ", Answer); //那么就输出个数
        exit(0); //直接退出整个程序
    }
    if (rchild[x]) find(rchild[x]);
}
int main()
{
    scanf("%d%d", &n, &need);
    for (int i=1; i <=n; ++i)
        scanf("%d%d%d", &number[i], &lchild[i], &rchild[i]);
    find(1); //从根节点开始中序查找
    return 0;
}

```

【参考程序 2】

```

#include <cstdlib>
#include <iostream>
using namespace std;
int n,x,rt,k=0; //k 统计遍历了几个结点
struct cyx {int data,left,right;};
cyx tree[1001];
bool sh[1001];
void in(int root) //中序遍历查找
{
    if (tree[root].left>0) in(tree[root].left); //中序遍历左子树
    k++;
    if (tree[root].data==x) //查找到对应元素
        {cout<<k;exit(0);} //输出结果并退出程序
    if (tree[root].right>0) in(tree[root].right); //中序遍历右子树
}
int main()
{
    int i;
    cin>>n>>x;
    for(i=1;i<=n;i++)

```

```

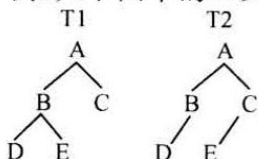
{
    cin>>tree[i].data>>tree[i].left>>tree[i].right;
    sh[tree[i].left]=1;
    sh[tree[i].right]=1;           //这两个节点都不可能是树根
}
for(i=1;i<=n;i++)
    if(! sh[i]) { rt=i;break;}    //找树根
in(rt);                          //查找
return 0;
}

```

6. 对称二叉树(tree_c)

如果二叉树的左右子树的结构是对称的,即两棵子树皆为空,或者皆不空,则称该二叉树是对称的。编程判断给定的二叉树是否对称。

例:如下图中的二叉树 T1 是对称的,T2 是不对称的。



二叉树用顺序结构给出,若读到#则为空,二叉树 T1=ABCDE, T2=ABCD#E,如果二叉树是对称的,输出“Yes”,反之输出“No”。

样例输入:

ABCDE

样例输出:

Yes

【参考程序】

```

#include<iostream>
using namespace std;
int main()
{
    string s;
    int i,j,k,n,m; bool t=true;
    cin>>s;
    n=s.size();
    if(n%2==0)           //如果节点个数为偶数,那么除去根节点后
    {
        cout<<" No "<<endl;
        return 0;        //就剩下奇数个,无法满足两棵子树皆空或皆不空
    }
    for(i=1;i<n;i+=2)
    {
        //判断每对同一节点的子节点,从第二个开始,因为第一个是根节点
        if((s[i]=='#'&& s[i+1]!='#') || (s[i+1]=='#'&& s[i]!='#')) t=false;
        //给出的顺序中,同一个节点的两个子节点是相邻的
        //如果其中恰好有一个是空的,它就不是对称二叉树
    }
    if(t==true) cout<<" Yes "<<endl; //是对称二叉树
    else cout<<" No "<<endl;        //不是对称二叉树
}

```

第二节 堆及其应用

1. 合并果子(fruit)

在一个果园里,多多已经把所有的果子打了下来,而且按果子的不同种类分成了不同的堆。多多决定把所有的果子合成一堆。

每一次合并,多多可以把两堆果子合并到一起,消耗的体力等于两堆果子的重量之和。可以看出,所有的果子经过 $n-1$ 次合并之后,就只剩下一堆了。多多在合并果子时总共消耗的体力等于每次合并所耗体力之和。

因为还要花大力气把这些果子搬回家,所以多多在合并果子时要尽可能地节省体力。假定每个果子重量都为 1,并且已知果子的种类数和每种果子的数目,你的任务是设计出合并的次序方案,使多多耗费的体力最少,并输出这个最小的体力耗费值。

例如有 3 种果子,数目依次为 1,2,9。可以先将 1、2 堆合并,新堆数目为 3,耗费体力为 3。接着,将新堆与原先的第三堆合并,又得到新的堆,数目为 12,耗费体力为 12。所以多多总共耗费体力 $= 3 + 12 = 15$ 。可以证明 15 为最小的体力耗费值。

输入:

输入文件 fruit.in 包括两行,第 1 行是一个整数 n ($1 \leq n \leq 30000$),表示果子的种类数。第 2 行包含 n 个整数,用空格分隔,第 i 个整数 a_i ($1 \leq a_i \leq 20000$) 是第 i 种果子的数目。

输出:

输出文件 fruit.out 包括一行,这一行只包含一个整数,也就是最小的体力耗费值。输入数据保证这个值小于 2^{31} 。

样例输入:

样例 1:

3

1 2 9

样例 2:

10

3 5 1 7 6 4 2 5 4 1

样例输出:

样例 1:

15

样例 2:

120

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include<queue>
#include<cmath>
using namespace std;
int heap[100001], heap_length, n, x, y, i, tot=0;
//heap 为堆数组, heap_length 为堆中元素个数

void put(int t)
{
    int now, next;
    heap[++heap_length]=t; //堆尾部插入新元素
    now=heap_length; //将堆中第 now 个元素向上调整
    while (now>1)
```

```

{
    next=now / 2;
    if (heap[now] >= heap[next]) return;
    swap(heap[now], heap[next]); //如果新元素比父亲节点值小,就交换
    now=next;
}
}

int get()
{
    int next, tot, now;
    tot=heap[1];
    heap[1]=heap[heap_length-1]; //弹出堆顶元素
    now=1; //将堆中第 now 个元素向下调整
    while (2 * now <= heap_length)
    {
        next=2 * now;
        if (next<heap_length&&heap[next+1]<heap[next])
            next++;
        if (heap[now] <= heap[next]) return tot;
        swap(heap[now], heap[next]);
        now=next;
    }
    return tot; //返回堆顶元素
}

int main()
{
    scanf("%d", &n);
    for (i=1; i <= n; i++)
    {
        scanf("%d", &x);
        put(x); //入堆
    }
    for (i=1; i<n; i++)
    {
        x=get();
        y=get();
        tot+=x+y;
        put(x+y);
    }
    printf("%d", tot);
}

```

2. 最小函数值(minval)

有 n 个函数,分别为 $F_1, F_2, \dots, F_n$ 。定义 $F_i(x) = A_i * x^2 + B_i * x + C_i (x \in \mathbf{N}^*)$ 。给定这些 A_i, B_i 和 C_i ,求出所有函数的所有函数值中最小的 m 个(如有重复的要输出多个)。

输入:

第 1 行输入两个正整数 n 和 m 。

以下 n 行每行三个正整数,其中第 i 行的三个数分别位 A_i, B_i 和 C_i 。输入数据保证 $A_i <$

$=10, B_i \leq 100, C_i \leq 10,000$ 。

输出:

输出将这 n 个函数所有可以生成的函数值排序后的前 m 个元素。

这 m 个数应该输出到一行,用空格隔开。

样例输入:

3 1 0

4 5 3

3 4 5

1 7 1

样例输出:

9 12 12 19 25 29 31 44 45 54

规模数据:

$n, m \leq 10,000$

【参考程序】

```
#include <iostream>
```

```
using namespace std;
```

```
int n,m,f[10001][4],size=0,heap[3000001],
```

```
tmp[10001];
```

```
int cyx(int p,int x) //求函数值,f[p][i]表示第p个函数的第i个系数
```

```
{
    return f[p][1] * x * x + f[p][2] * x + f[p][3];
}
```

```
void put(int x) //增加元素
```

```
{
    int i,j,t;
    size++;heap[size]=x;i=size; //堆尾部插入新元素
    while(i>1) //将堆中第i个元素向上调整
    {
        j=i/2;
        if(heap[i]>=heap[j]) return;
        t=heap[i];heap[i]=heap[j];heap[j]=t;
        //如果新元素比父亲节点值小,就交换
        i=j;
    }
}
```

```
int get() //删除最小元素
```

```
{
    int i=1,j,t,res=heap[1];
    heap[1]=heap[size];size--; //弹出堆顶元素
    //将堆中第i个元素向下调整
    while(i*2<=size)
    {
        j=i*2;
        if(j<size && heap[j+1]<heap[j]) j++;
        if(heap[i]<=heap[j]) return res;
    }
}
```

```

        t=heap[i];heap[i]=heap[j];heap[j]=t;
        i=j;
    }
    return res;                                //返回堆顶元素
}

int main()
{
    int i,j,x;
    bool r=1;
    cin>>n>>m;
    for(i=1;i<=n;i++)
        for(j=1;j<=3;j++)
            cin>>f[i][j];
    i=1;                                        //i 为自变量
    while((i-1)*n<m)
    {
        //将 n 个函数的前 k 小的函数值入堆,且 k 为满足  $k*n \leq m$  的最大值
        for(j=1;j<=n;j++)
            put(cyx(j,i)); //入堆
        i++;                                //自变量加 1
    }
    while(r)                                //如果已经确定了前 m 小的函数值
    {
        for(j=1;j<=m;j++) tmp[j]=get();
        //取出前 m 小的函数值
        int mth=tmp[m];                    //mth 储存前 m 小的函数值中最大者
        for(j=1;j<=m;j++) put(tmp[j]);
        //将 tmp 数组里的函数值压回堆中
        r=0;
        for(j=1;j<=n;j++)
            if(cyx(j,i)<mth)                //如果这个新的函数值可以更新前 m 小的值
                r=1;
        if(r) for(j=1;j<=n;j++)
            put(cyx(j,i)); //如果可以更新前 m 小的值,就把自变量为 i 时的函数值入堆
        /* i 为自变量时,如果没有任何函数值可以更新当前的前 m 小的值,那么接下去的函数值
        也一定不能更新前 m 小的值了,此时就确定了前 m 小的函数值,循环结束 */
        i++;                                //自变量的值加 1
    }
    for(i=1;i<=m;i++)
        cout<<get()<<" ";
    return 0;
}

```

3. 看病

有个朋友在医院工作,想请 BSNY 帮忙做个登记系统。具体是这样的,最近来医院看病的人越来越多了,因此很多人要排队,只有当空闲时放一批病人看病。但医院的排队不同于其他地方,因为多数情况下,需要病情严重的人优先看病,所以希望 BSNY 设计系统时,以病情的严重程度作为优先级,判断接下来谁可以去看病。

输入:

第 1 行输入 n , 表示有 n 个操作。

对于每个操作, 首先输入 push 或 pop。

push 的情况, 之后会输入 a_i 和 b_i , 分别表示患者姓名和患者病情优先级。

pop 后面没有输入, 但需要你输出。

输出:

对于 pop 的操作, 输出此时还在排队人中, 优先级最大的患者姓名和优先级。

表示他可以进去看病了。

如果此时没人在排队, 那么输出 "none", 具体可见样例。

样例输入:

```
7
pop
push bob 3
push tom 5
push ella 1
pop
push zkw 4
pop
```

样例输出:

```
none
tom 5
zkw 4
```

数据规模和约定:

$1 \leq n \leq 100,000$, 每个人的优先级都不一样, $0 \leq \text{优先级} \leq 2,000,000,000$ 。

姓名都是小写字母组成的, 长度小于 20。

【算法分析】

本题是一道比较基础的堆的题。主要根据优先级入堆(push), 出堆(pop), 由于同时要记录姓名和优先级, 可以用结构体数组存储, 具体可参考下述代码。

【参考程序——堆操作】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
struct node{
    char nm[25];
    int v;
} hp[100005];
int sz; //hp 为堆数组, sz 为堆中元素个数
void shift_up(int i){ //将堆中第 i 个元素向上调整
    while(i>1 && hp[i/2].v<hp[i].v){
        swap(hp[i/2], hp[i]); //如果新元素比父亲结点值小, 就交换
        i=i/2;
    }
}
void push(node x){
    hp[++sz]=x; //堆数组的尾部推入新元素
```

```

    shift_up(sz);
}
node top() { //返回堆顶元素
    return hp[1];
}
void shift_down(int i) { //将堆中第 i 个元素向下调整
    int j;
    while(i * 2 <= sz) {
        if(i * 2 + 1 <= sz && hp[i * 2 + 1].v > hp[i * 2].v) j = i * 2 + 1;
        else j = i * 2;
        if(hp[j].v > hp[i].v) {
            swap(hp[j], hp[i]);
            i = j;
        } else break;
    }
}
void pop() { //弹出堆顶元素
    hp[1] = hp[sz--];
    shift_down(1);
}
int main() {
    int n;
    scanf("%d", &n);
    sz = 0;
    for(int i = 1; i <= n; i++) {
        char op[25];
        scanf("%s", op);
        if(op[1] == 'o') {
            if(sz == 0) { //如果堆空了,要输出"none"
                printf("none\n");
                continue;
            }
            node tmp = top();
            pop();
            printf("%s %d\n", tmp.nm, tmp.v);
        } else {
            node tmp;
            scanf("%s%d", tmp.nm, &tmp.v);
            push(tmp);
        }
    }
    return 0;
}

```

4. 小明的账单

小明在一次聚会中,不慎遗失了自己的钱包,在接下来的日子,面对小明的将是一系列的补卡手续和堆积的账单……在小明的百般恳求下,老板最终同意延缓账单的支付时间。可老板又提出,必须从目前还没有支付的所有账单中选出面额最大和最小的两张,并把他们付清。还没有支付的账单会被保留到下一天。请你帮他计算出支付的顺序。

输入格式:

第 1 行:一个正整数 $N(N \leq 15,000)$,表示小明补办银联卡总共的天数。

第 2 行到第 $N+1$ 行:每一行描述一天中收到的账单。先是一个非负整数 $M(M \leq 100)$,表示当天收到的账单数,后跟 M 个正整数(都小于 1,000,000,000),表示每张账单的面额。

输入数据保证每天都可以支付两张账单。

输出:

输出共 N 行,每行两个用空格分隔的整数,分别表示当天支付的面额最小和最大的支票的面额。

样例输入:

```
4
3 3 6 5
2 8 2
3 7 1 7
0
```

样例输出:

```
3 6
2 8
1 7
5 7
```

【算法分析】

本题也是堆的基本操作,但要求提高了,既要有大根堆,又要有小根堆,当去掉堆顶元素时,要将此元素标记,防止在另一个堆中又出现。这里, $C++$ 又要推荐一种“福利”,就是红黑树 `set`,它就有功能取得最大值和最小值,也就是一棵树中就能完成最大值最小值的快速搜索,具体操作如下。

【参考程序——set 红黑树实现堆】

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<set>
using namespace std;
multiset<int>st; //定义 int 类型的 set
multiset<int>::iterator it; //定义 set 的迭代器
int main(){
    int n;
    scanf("%d",&n);
    st.clear();
    for(int i=1; i<=n; i++){
        int m;
        scanf("%d",&m);
        for(int j=1; j<=m; j++){
            int a;
            scanf("%d",&a);
            st.insert(a); //插入元素
        }
        printf("%d",*st.begin()); //输出最小元素
        st.erase(st.begin()); //删除 set 中最小元素
        printf("%d\n",*(--st.end())); //输出最大元素
        st.erase(--st.end()); //删除 set 中最大元素
    }
    return 0;
}
```

5. 鱼塘钓鱼(fishing)

有 N 个鱼塘排成一行($N < 100$), 每个鱼塘中有一定数量的鱼, 例如: $N=5$ 时, 如下表:

鱼塘编号	1	2	3	4	5
第 1 分钟能钓到的鱼的数量($1 \cdots 1000$)	10	14	20	16	9
每过 1 分钟钓鱼数的减少量($1 \cdots 100$)	2	4	6	5	3
当前鱼塘到下一个相邻鱼塘需要的时间(单位: 分钟)	3	5	4	4	

即: 在第 1 个鱼塘中钓鱼第 1 分钟内可钓到 10 条鱼, 第 2 分钟内只能钓到 8 条鱼, ……第 5 分钟以后再也钓不到鱼了。从第 1 个鱼塘到第 2 个鱼塘需要 3 分钟, 从第 2 个鱼塘到第 3 个鱼塘需要 5 分钟, ……

编程任务:

给出一个截止时间 T ($T < 1000$), 设计一个钓鱼方案, 从第 1 个鱼塘出发, 希望能钓到最多的鱼。

假设能钓到鱼的数量仅和已钓鱼的次数有关, 且每次钓鱼的时间都是整数分钟。

输入:

输入文件共五行, 分别为:

第 1 行为 N ;

第 2 行为第 1 分钟各个鱼塘能钓到的鱼的数量, 每个数据之间用一个空格隔开;

第 3 行为每过 1 分钟各个鱼塘钓鱼数的减少量, 每个数据之间用一个空格隔开;

第 4 行为当前鱼塘到下一个相邻鱼塘需要的时间, 每个数据之间用一个空格隔开;

第 5 行为截止时间 T ;

输出:

输出文件仅一个整数(不超过 $2^{31}-1$), 表示你的方案能钓到的最多的鱼。

样例输入:

```
5
10 14 20 16 9
2 4 6 5 3
3 5 4 4
1 4
```

样例输出:

```
76
```

【参考程序】

```
#include <iostream>
using namespace std;
int n, lizhiyi[101], chenxian[101],
    chenyuxiang[101], size, heap[100001], T, hlake[100001];
void put(int x, int lake) //增加元素
{
    int i, j, t;
    size++; heap[size] = x;
    hlake[size] = lake; i = size;
    while(i > 1) //将堆中第 i 个元素向上调整
    {
        j = i/2;
        if(heap[i] <= heap[j]) return;
        t = heap[i]; heap[i] = heap[j]; heap[j] = t;
        t = hlake[i]; hlake[i] = hlake[j]; hlake[j] = t;
        //如果新元素比父亲节点值小, 就交换
    }
}
```

```

        i=j;
    }
}

void Heap(int erw)                                //将堆中第 i 个元素向下调整
{
    int i=erw,j,t;
    while(i * 2 <= size)
    {
        j=i * 2;
        if(j < size && heap[j+1] > heap[j]) j++;
        if(heap[i] >= heap[j]) return;
        t=heap[i]; heap[i]=heap[j]; heap[j]=t;
        t=hlake[i]; hlake[i]=hlake[j]; hlake[j]=t;
        i=j;
    }
}

int main()
{
    int i,j,m,p=0,ans=0;
    cin >> n;
    for(i=1; i<=n; i++) cin >> lizhiyi[i];
    for(i=1; i<=n; i++) cin >> chenxian[i];
    for(i=1; i<=n-1; i++) cin >> chenyuxiang[i];
    cin >> m;
    for(i=1; i<=n; i++)                        //枚举最后一个到达的鱼塘
    {
        T=m-p;
        int sum=0; size=0;
        for(j=1; j<=i; j++) put(lizhiyi[j],j);
        while(T>0 && heap[1]>0)                //时间没有结束且这个鱼塘里有鱼
        {
            sum+=heap[1];
            heap[1]-=chenxian[hlake[1]]; //对应的鱼塘鱼的数量减少
            Heap(1);                        //调整
            T--;                            //时间递减 1
        }
        if(sum>ans) ans=sum;                //更新最大值
        p+=chenyuxiang[i];                  //到下一个鱼塘
    }
    cout << ans;
    return 0;
}

```

第四章 图论算法

第一节 图的遍历

1. 铲雪车(snow)

随着白天越来越短夜晚越来越长,我们不得不考虑铲雪问题了。整个城市所有的道路都是双车道,因为城市预算的削减,整个城市只有1辆铲雪车。铲雪车只能把它开过的地方(车道)的雪铲干净,无论哪儿有雪,铲雪车都得从停放的地方出发,游历整个城市的街道。现在的问题是:最少要花多少时间去铲掉所有道路上的雪呢?

输入:

输入数据的第1行表示数据的组数,第2行表示铲雪车的停放坐标(x,y),x,y为整数,单位为米。下面最多有100行,每行给出了一条街道的起点坐标和终点坐标,所有街道都是笔直的,且都是双向一个车道。铲雪车可以在任意交叉口、或任何街道的末尾任意转向,包括转U型弯。铲雪车铲雪时前进速度为20 km/h,不铲雪时前进速度为50 km/h。

保证:铲雪车从起点一定可以到达任何街道。

输出:

铲掉所有街道上的雪并且返回出发点的最短时间,精确到分钟。

样例输入:

```
1
0 0
0 0 10000 10000
5000 -10000 5000 10000
5000 10000 10000 10000
```

样例输出:

```
3:55
```

【注解】

3 小时 55 分钟

【参考程序】

```
#include<iostream>
#include<cmath>
using namespace std;
int main()
{
    long long x,y,x1,y1,x2,y2;
    long double s=0;
    cin>>x>>y;
    while (cin>>x1>>y1>>x2>>y2)
        s+=sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
        //求每条路的长
    s=s*2/1000/20;
        //每条路都要走两次,所以总长*2
    //单位化成 km,所以总长/1000;保证从起点能到达任意一条街道,即每一个单独的点
    都至少有偶数条边相连,所以不会重复经过,由此可知,该图是欧拉图
    //速度保持 20km/h,所以时间为 s/20
```

```

int h=floor(s);           //h 为 s 取整,表示用时 h 小时
s=(s-h)*60;              //s 表示换算后的分钟数
int m=s;                  //m 为 s 强转的整数
if (s-m>=0.5) m++;
    cout<<h<<" ";
if (m<10) cout<<"0 ";   //注意时间的表示
    cout<<m<<endl;
return 0;
}

```

2. 骑马修栅栏(fence)

农民 John 每年有很多栅栏要修理。他总是骑着马穿过每一个栅栏并修复它破损的地方。

John 是一个与其他农民一样懒的人。他讨厌骑马,因此从来不两次经过同一个栅栏。你必须编一个程序,读入栅栏网络的描述,并计算出一条修栅栏的路径,使每个栅栏都恰好被经过一次。John 能从任何一个顶点(即两个栅栏的交点)开始骑马,在任意一个顶点结束。

每一个栅栏连接两个顶点,顶点用 1 到 500 标号(虽然有的农场并没有 500 个顶点)。一个顶点上可连接任意多(≥ 1)个栅栏。所有栅栏都是连通的(也就是你可以从任意一个栅栏到达另外的所有栅栏)。

你的程序必须输出骑马的路径(用路上依次经过的顶点号码表示)。我们如果把输出的路径看成是一个 500 进制的数,那么当存在多组解的情况下,输出 500 进制表示法中最小的一个(也就是输出第一个较小的数,如果还有多组解,输出第二个较小的数,等等)。输入数据保证至少有一个解。

输入:

第 1 行: 一个整数 $F(1 \leq F \leq 1024)$, 表示栅栏的数目;

第 2 到 $F+1$ 行: 每行两个整数 $i, j(1 \leq i, j \leq 500)$ 表示这条栅栏连接 i 与 j 号顶点。

输出:

输出应当有 $F+1$ 行,每行一个整数,依次表示路径经过的顶点号。注意数据可能有多组解,但是只有上面题目要求的那一组解是认为正确的。

样例输入:

```

9
1 2
2 3
3 4
4 2
4 5
2 5
5 6
5 7
4 6

```

样例输出:

```

1
2
3
4
2
5
4
6
5
7

```

【参考程序】

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cmath>
#include <algorithm>

```

```

using namespace std;
#define maxn 2001
int g[maxn][maxn];    //用邻接矩阵储存,g[i][j]表示点i到点j共有多少条栅栏(边)
int du[maxn];          //记录每个点的度,也就是有多少条边与其连接
int circuit[maxn];     //记录欧拉路径
int n, m, tot, x, y, start; //start 记录欧拉路径的起点
                          //n 记录点数(编号最大的点),m 记录边数,tot 记录欧拉路径的总点数
void find(int i)
{
    for (int j=1; j<=n; ++j) //从小到大寻找,符合题目要求
        if (g[i][j])        //如果有边还没走,就从这走
        {
            g[i][j]--;
            g[j][i]--;        //将这条边删去,避免重复
            find(j);
        }
    circuit[++tot]=i;        //记录走过的路径
}

int main()
{
    memset(g, 0, sizeof(g));
    scanf("%d", &m);
    for (int i=1; i<=m; ++i)
    {
        scanf("%d%d", &x, &y);
        g[x][y]++;          //邻接矩阵存图
        g[y][x]++;          //由于两点之间可能有多条边连接,所以要用++
        du[x]++;             //记录入度
        du[y]++;
        n=max(n, max(x, y));
                          //记录最大的编号
    }
    start=1;
    for (int i=1; i<=n; ++i)
        if (du[i] % 2)        //如果图中有奇点(度为奇数)
        {
            start=i;
            break;            //就以编号小的为起点(题目要求),直接 break
        }
    tot=0;
    find(start);              //从起点开始
    for (int i=tot; i>=1; --i)
        printf("%d\n", circuit[i]);
    //因为是深度优先搜索,先搜索再记录,所以答案的顺序是反过来的
    return 0;
}

```


第二节 最短路径算法

1. 信使(msner)

战争时期,前线有 n 个哨所,每个哨所可能会与其他若干个哨所之间有通信联系。信使负责在哨所之间传递信息,当然,这是要花费一定时间的(以天为单位)。指挥部设在第一个哨所。当指挥部下达一个命令后,就派出若干个信使向与指挥部相连的哨所送信。当一个哨所接到信后,这个哨所内的信使们也以同样的方式向其他哨所送信。直至所有 n 个哨所全部接到命令后,送信才算成功。因为准备充足,每个哨所内都安排了足够的信使(如果一个哨所与其他 k 个哨所有通信联系的话,这个哨所内至少会配备 k 个信使)。

现在总指挥请你编一个程序,计算出完成整个送信过程最短需要多少时间。

输入:

输入文件 msner.in,第 1 行有两个整数 $n(1 \leq n \leq 100)$ 和 m ,中间用 1 个空格隔开,分别表示有 n 个哨所和 m 条通信线路。

第 2 至 $m+1$ 行:每行三个整数 i, j, k ,中间用一个空格隔开,表示第 i 个和第 j 个哨所之间存在通信线路,且这条线路要花费 k 天。

输出:

输出文件 msner.out,仅一个整数,表示完成整个送信过程的最短时间。如果不是所有的哨所都能收到信,就输出 -1。

样例输入:

```
4 4
1 2 4
2 3 7
2 4 1
3 4 6
```

样例输出:

```
11
```

【参考程序】

```
#include<iostream>
#include<cstdio>
#include<cstring>
#include<cmath>
using namespace std;
int a[101][3],f[101][101];
int n,i,j,k,x,y,m,s,e;
int main()
{
    cin>>n>>m;
    memset(f,0x7f,sizeof(f));
    for(i=1;i<=m;i++)
    {
        cin>>x>>y>>k;
        f[x][y]=f[y][x]=k;
    }
    for(i=1;i<=n;i++)
        f[i][i]=0;
    for(k=1;k<=n;k++)
```

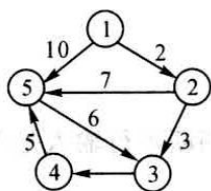
//初始化为很大的值

//邻接矩阵建双向边

//初始化自己到自己的距离为 0

//floyd 找两点间的最短路

样例所对应的图如下:



$$W = \begin{matrix} & \begin{matrix} 1 & 2 & 3 & 4 & 5 \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 2 & - & - & 10 \\ - & 0 & 3 & - & 7 \\ - & - & 0 & 4 & - \\ - & - & - & 0 & 5 \\ - & - & 6 & - & 0 \end{bmatrix} \end{matrix}$$

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<string>
using namespace std;
int g[200][200];
int get(string x)
{
    if(x=="-")return 0x7fff;    //“-”说明这两个点没有直接连通,初始化为很大的值
    if(x[0]=='-'){               //这个字符串的第一个字符是“-”,对应的是负数
        int ans=0;
        for(int i=1;i<x.size();i++) //字符串转为数
            ans=ans*10+x[i]-48;
        return -ans;             //返回这个数的值
    }
    else{
        int ans=0;
        for(int i=0;i<x.size();i++)
            ans=ans*10+x[i]-48;
        return ans;
    }
}

int main()
{
    int n,i,j,k,f;
    string s;
    cin>>n>>f;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        {
            cin>>s;
            g[i][j]=get(s);           //邻接矩阵存单向边
        }
    for(k=1;k<=n;k++)                //floyd 求任意两点间的最短路径
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                g[i][j]=min(g[i][j],g[i][k]+g[k][j]);
    for(i=1;i<=n;i++)                //输出源点到其他每一个点的最短距离
        if(f!=i)
            cout<<"("<<f<<"-> "<<i<<"="<<g[f][i]<<endl;
            //按照题目要求格式输出

    return 0;
}
```

【参考程序 2】

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
#include <algorithm>
using namespace std;

const int MaxInt=0x7fffffff / 3;

int n, v0, dis[81][81];
char s[51];          //n,v0 的意义与题目一致,dis[i][j]表示从 i 到 j 的最短路径

int main()
{
    cin >> n >> v0;
    for (int i=1; i <=n; ++i)
        for (int j=1; j <=n; ++j)
        {
            cin >> s;
            if (strcmp(s, "-") == 0)
                dis[i][j]=MaxInt;          //输入“-”,说明两点之间没有边,值设为一个很大的数
            else
                sscanf(s, "%d", &dis[i][j]); //输入的是数字,就把字符串转化成数字
        }

    for (int k=1; k <=n; ++k)          //用 Floyd 算法求两点间的最短路
        for (int i=1; i <=n; ++i)
            for (int j=1; j <=n; ++j)
                if (i != j && j != k && i != k)
                    dis[i][j]=min(dis[i][j], dis[i][k]+dis[k][j]);

    for (int i=1; i <=n; ++i)
        if (i != v0)                    //输出从源点到各点之间的最短路
            printf("(%d->%d)=%d\n", v0, i, dis[v0][i]);
    return 0;
}

```

4. 热浪(heatwv)

德克萨斯纯朴的民众们这个夏天正在遭受巨大的热浪！他们的德克萨斯长角牛吃起来不错，可是他们并不是很擅长生产富含奶油的乳制品。Farmer John 此时以先天下之忧而忧，后天下之乐而乐的精神，身先士卒地承担起向德克萨斯运送大量的营养冰凉的牛奶的重任，以减轻德克萨斯人忍受酷暑的痛苦。

FJ 已经研究过可以把牛奶从威斯康星运送到德克萨斯州的路线。这些路线包括起始点和终点一共经过 T ($1 \leq T \leq 2,500$) 个城镇，方便地标号为 1 到 T 。除了起点和终点外的每个城镇由两条双向道路连向至少两个其他的城镇。每条道路有一个通过费用(包括

油费、过路费等)。

给定一个地图,包含 C ($1 \leq C \leq 6,200$) 条直接连接 2 个城镇的道路。每条道路由道路的起点 Rs , 终点 Re ($1 \leq Rs \leq T$; $1 \leq Re \leq T$), 和花费 ($1 \leq Ci \leq 1,000$) 组成。求从起始的城镇 Ts ($1 \leq Ts \leq T$) 到终点的城镇 Te ($1 \leq Te \leq T$) 最小的总费用。

输入:

第 1 行: 4 个由空格隔开的整数: T, C, Ts, Te ;

第 2 到第 $C+1$ 行: 第 $i+1$ 行描述第 i 条道路。有 3 个由空格隔开的整数: Rs, Re 和 Ci 。

输出:

一个单独的整数表示从 Ts 到 Te 的最小总费用。数据保证至少存在一条道路。

样例输入:

```
7 11 5 4
2 4 2
1 4 3
7 2 2
3 4 3
5 7 5
7 3 3
6 1 1
6 3 4
2 4 3
5 6 3
7 2 1
```

样例输出:

```
7
```

【样例说明】

$5 \rightarrow 6 \rightarrow 1 \rightarrow 4$ ($3+1+3$)

【参考程序 1】

```
#include<cstdio>
#include<iostream>
#include<cstring>
using namespace std;
int c,rs,re,ci,ts,te,t,i,j,k,tot,v,head,tail;
int num[20001],g[2501][2501],dis[20001],a[2501][2501],team[20001];
bool u[20001];
int main()
{
    scanf("%d%d%d%d",&t,&c,&ts,&te);
    for (i=1; i<=c; ++i)
    {
        scanf("%d%d%d",&rs,&re,&ci);
        g[rs][re]=g[re][rs]=ci;
        a[rs][++num[rs]]=re;      //用邻接矩阵建立双向边
        a[re][++num[re]]=rs;
    }
    for (j=1; j<=t; ++j) dis[j]=99999999;
```

```

memset(team,0,sizeof(team));
memset(u,false,sizeof(u));
dis[ts]=0; team[1]=ts; head=0; tail=1; u[ts]=true;
do //SPFA 求解最短路
{
    head++;
    head=((head-1)%1601)+1; //循环使用队列(节省空间)
    v=team[head];
    u[v]=false;
    for (j=1; j<=num[v]; ++j)
    if (dis[a[v][j]]>dis[v]+g[v][a[v][j]])
    { //如果选择 v->j 为其中一条路径,使得起点到 j 的距离更短,更新
        dis[a[v][j]]=dis[v]+g[v][a[v][j]];
        if(! u[a[v][j]]) //不在队列中(被松弛),入队且标记
        {
            tail++;
            tail=((tail-1)%1601)+1;
            team[tail]=a[v][j];
            u[a[v][j]]=true;
        }
    }
}
}while(head!=tail); //注意循环队列中不能用 head<tail
cout<<dis[te];
}

```

【参考程序 2】

```

#include <cstdio>
#include <cstdlib>
#include <vector>
#include <cstring>
#include <queue>
using namespace std;
const int MAXN=2501;
typedef vector<int> Vec;
Vec Map[MAXN],Val[MAXN]; //不定长数组 stl
int N,M,S,T;

void init()
{
    scanf("%d %d %d %d\n",&N,&M,&S,&T);

    int a,b,v;
    for(int i=1;i<=M;i++)
    {
        scanf("%d %d %d\n",&a,&b,&v);
        Map[a].push_back(b); //申请一个空间 map[a][b]
        Val[a].push_back(v); //val[a][v]
        Map[b].push_back(a); //双向边
    }
}

```

```

        Val[b].push_back(v);
    }
}

int dist[MAXN];
int flag[MAXN];
const int INF=1000000000;
queue<int> Q; //队列 stl
void SPFA()
{
    for(int i=1;i<=N;i++)
        dist[i]=INF,flag[i]=0; //初始化
    dist[S]=0,flag[S]=1; //起点的距离为0,标记为走过
    Q.push(S); //起点入队
    int t,tmp;
    while(! Q.empty()) //spfa 找最短路径
    {
        t=Q.front();
        Q.pop();
        flag[t]=0; //取出队头,标记解除
        for(unsigned int i=0;i<Map[t].size();i++)//枚举与t相连的所有点i
        {
            tmp=dist[t]+Val[t][i];
            if(tmp<dist[Map[t][i]])
            { //如果选择 t->i 为其中一条路径,使得起点到i的距离更短,更新
                dist[Map[t][i]]=tmp;
                if(! flag[Map[t][i]]) //不在队列里(被松弛),入队,标记
                {
                    Q.push(Map[t][i]);
                    flag[Map[t][i]]=1;
                }
            }
        }
    }
    printf("%d\n",dist[T]);
    return;
}

int main()
{
    init();
    SPFA();
    return 0;
}

```

5. 分糖果(candy)

童年的我们,将和朋友分享美好的事物作为自己的快乐。这天,C小朋友得到了 Plenty of candies,将要把这些糖果分给要好的朋友们。已知糖果从一个人传给另一个人需要 1 秒的时间,同一个小朋友不会重复接受糖果。由于糖果足够多,如果某时刻某小朋友接受了糖果,他会将糖果分成若干份,分给那些在他身旁且还没有得到糖果的小朋友们,而且自己会吃一些糖果。由于嘴馋,小朋友们等不及将糖果发完,会在得到糖果后边吃边发。每个小朋友从接受糖果到吃完糖果需要 m 秒的时间。那么,如果第 1 秒 C 小朋友开始发糖,第几秒所有小朋友都吃完了糖呢?

输入:

第 1 行为三个数 n, p, c , 为小朋友数、关系数和 C 小朋友的编号。

第 2 行为一个数 m , 表示小朋友吃糖的时间。

下面 p 行每行两个整数, 表示某两个小朋友在彼此身旁。

输出:

一个数, 为所有小朋友都吃完了糖的时间。

样例输入:

4 3 1

2

1 2

2 3

1 4

样例输出:

5

【样例解释】

第 1 秒, 糖在 1 手上; 第 2 秒, 糖传到了 2, 3 的手中; 第 3 秒, 糖传到了 4 的手中, 此时 1 吃完了; 第 4 秒, 2, 3 吃完了; 第 5 秒, 4 吃完了。所以答案是 5。

【限制】

40% 的数据满足: $1 \leq n \leq 100$;

60% 的数据满足: $1 \leq n \leq 1000$;

100% 的数据满足: $1 \leq n \leq 100,000$;

$m \leq n * (n - 1) / 2$, 不会有同一个关系被描述多次的情况。

【参考程序 1】

/* 由于每个小朋友吃糖的时间相同, 因此其他小朋友在传到糖的最后一个小朋友吃完糖之前一定会把糖吃完。

此时问题的答案就被我们转化为了求最长路(起点到每个点最短路中的最大值)+吃一颗糖的时间 */

```
#include<cstdio>
```

```
#include<queue>
```

```
#include<iostream>
```

```
#include<cstring>
```

```
using namespace std;
```

```
queue<int> Q;
```

```
struct point
```

```
{
```

```
    int next,to;
```

```
}a[2000001];
```

```
int n,p,c,m,x,y,tot,s,w,maxx;
```

```
int dis[1000001],last[1000001];
```



```

bool v[100001];
void INT(int tx,int ty)
{
    tot++;
    a[tot].next=last[tx];
    a[tot].to=ty;
    last[tx]=tot;
}
int main()
{
    memset(dis,127/3,sizeof(dis));
    scanf("%d%d%d%d",&n,&p,&c,&m);
    for (int i=1; i<=p; ++i)
    {
        scanf("%d%d",&x,&y);
        INT(x,y);
        INT(y,x); //邻接表建双向边
    }
    Q.push(c); dis[c]=0; //SPFA 求最短路(利用 C++ STL 模拟队列)
    while (! Q.empty())
    {
        s=Q.front();
        v[s]=false;
        for (int i=last[s]; i; i=a[i].next) //枚举与 s 相连的边
        {
            w=a[i].to; //取出相连的边 w
            if (dis[w]>dis[s]+1)
            {
                dis[w]=dis[s]+1; //更新最短路径
                if (! v[w]) //如果 y 被松弛,同时不在队列中,入队且标记
                {
                    v[w]=true;
                    Q.push(w);
                }
            }
        }
        Q.pop();
    }
    for (int i=1; i<=n; ++i)
        maxx=max(dis[i],maxx); //计算最长路
    printf("%d\n",maxx+m+1); //时间从第 1 秒结束后开始计算
}

```


【参考程序 2】

```

#include <cmath>
#include <string>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;
#define maxn 100001
#define MOD 200001

//分析:我们把每个小朋友看作一个结点,两个小朋友彼此在身旁则连一条双向边,权
//值为 1(下面的程序直接在计算最短路时加上边权 1)
//然后用 SPFA 算法求出源点 C 到其他各点的最短路,取到所有点中最长的最短
//路径(记为 maxx),即为所有小朋友都得到糖果的时间
//由于小朋友可以边吃边传,而时间都是 m,最迟吃完的必然是最迟传到的,即最迟吃
//完的时间就为 maxx+m

struct edge
{
    int next;    //next 表示枚举与某一个点相连的边时,这条边的下一条的编号
    int to;      //to 表示这条边连向的点
}a[200001];

int first[maxn], h[maxn * 2 + 5], dis[maxn];
//first[i]表示枚举与 i 相连的边时,第一个枚举到的边的编号;h 为循环队列;dis[i]表
//示从源点 c 到 i 的最短路

int n, p, c, m;    //n,p,c,m 的含义和题目一致
int head=0, tail=1, num, x, y, maxx=-1;
//head,tail 分别为队列的头尾指针,num 记录总边数,maxx 记录从源点到所有点中最
//长的最短路径

bool exist[maxn];    //exist[i]表示点 i 此时是否在队列中

int add(int x, int y) //连接一条 x→y 的有向边,无向边即为两条有向边
{
    num++;
    a[num].to=y;
    a[num].next=first[x];
    first[x]=num;
}

int main()
{
    memset(dis, 0x7f/3, sizeof(dis)); //所有的最短距离都初始化为一个很大的数

    scanf("%d%d%d%d", &n, &p, &c, &m);
    //输入 n,p,c,m

```

```

for (int i=1; i <=p; ++i)
{
    scanf("%d%d", &x, &y);
    add(x, y);
    add(y, x);                //邻接表建双向边
}

//SPFA 算法求单源最短路径
h[1]=c;                      //把源点入队
dis[c]=1;    //题目中是从第 1 秒开始发糖,所以源点 c 的 dis 初始化为 1
exist[c]=true;                //记录 c 在队列中
do
{
    head=(head % MOD)+1;      //循环队列队头出队
    x=h[head];
    exist[x]=false;           //x 此时不在队列中

    for (int i=first[x]; i; i=a[i].next) //枚举与 x 相连的边
    {
        y=a[i].to;            //取出连向的点为 y
        if (dis[y] > dis[x]+1)
        {
            dis[y]=dis[x]+1;    //更新最短路径
            if (! exist[y])      //如果 y 被松弛,同时不在队列中,入队且标记
            {
                tail=(tail % MOD)+1;
                h[tail]=y;
                exist[y]=true;    //就将 y 入队去修改其他结点的最短路
            }
        }
    }
} while (head != tail);

for (int i=1; i <=n; ++i)
maxx=max(maxx, dis[i]);
//找到从源点到所有点中最长的最短路径,即为所有小朋友都得到糖果的时间

printf("%d", maxx+m);
//由于小朋友可以边吃边传,而时间都是 m,最迟吃完的必然是最迟传到的,即最迟吃
完的时间就为 maxx+m
return 0;
}

```

6. 城市路(Dijkstra)

罗老师被邀请参加一个舞会,是在城市 n ,而罗老师当前所处的城市为 1,附近还有很多城市 $2 \sim n-1$,有些城市之间没有直接相连的路,有些城市之间有直接相连的路,这些路都是双向的,当然也可能有多条。

现在给出直接相邻城市的路长度,罗老师想知道从城市 1 到城市 n 的最短距离。

输入:

输入 n, m , 表示 n 个城市和 m 条路。

接下来 m 行, 每行有 a, b, c , 表示城市 a 与城市 b 有长度为 c 的路。

输出:

输出 $1 \sim n$ 的最短路。

如果 1 到达不了 n , 就输出 -1 。

样例输入:

```
5 5
1 2 20
2 3 30
3 4 20
4 5 20
1 5 100
```

样例输出:

```
90
```

【数据规模和约定】

$1 \leq n \leq 2000$;

$1 \leq m \leq 10,000$;

$0 \leq c \leq 10,000$ 。

【算法分析】

本题是单源最短路简单的应用, 求顶点 1 到顶点 n 的最短路, 并且顶点的规模在 2000 以内, 所以可以用 $O(n^2)$ 的 Dijkstra 求解, 具体代码如下。

【参考程序——Dijkstra 算法】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
const int oo=1000000000;
const int M=2005;
int dis[M],g[M][M],mark[M];
int n,m;
void dij(){
    dis[1]=0; mark[1]=1;
    for(int i=2; i<=n; i++){
        dis[i]=g[1][i]; mark[i]=0;    //首先以 1 为起点,更新其他点到 1 的距离
    }
    for(int i=2; i<=n; i++){
        int id,tmp=oo;
        for(int j=1; j<=n; j++){
            if(mark[j]==0 && dis[j]<tmp){
                tmp=dis[j]; id=j;    //找到离起点最近的点
            }
        }
        mark[id]=1;
        for(int j=1; j<=n; j++){    //使用最近的点更新其他点到起点的距离
            if(dis[j]>dis[id]+g[id][j])
                dis[j]=dis[id]+g[id][j];
        }
    }
}
```

```

    }
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i=1; i<=n; i++){
        for(int j=1; j<=n; j++) g[i][j]=oo;
        g[i][i]=0;           //初始化
    }
    for(int i=1; i<=m; i++){
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        g[a][b]=g[b][a]=min(c, g[a][b]); //双向边,同时如果有重边,取最小边
    }
    dij();
    if(dis[n]>=oo) dis[n]=-1;
    printf("%d\n",dis[n]);
    return 0;
}

```

7. 最短路(Spfa)

给定 M 条边, N 个点的带权无向图。

求 1 到 N 的最短路。

$N \leq 100,000, M \leq 500,000$ 。

输入:

第 1 行: N, M ;

接下来 M 行 3 个正整数: a_i, b_i, c_i 表示 a_i, b_i 之间有一条长度为 c_i 的路, $c_i \leq 1000$ 。

输出:

一个整数, 表示 1 到 N 的最短距离。

样例输入:

```

4 4
1 2 1
2 3 1
3 4 1
2 4 1

```

样例输出:

```

2

```

【样例解释】

注意图中可能有重边和自环, 数据保证 1 到 N 有路径相连。

【算法分析】

本题就是最短路的应用, 相比上面的题目, 数据规模大很多, $O(n^2)$ 和 $O(n * e)$ 的算法都将超时。这种情况下, 平均期望复杂度为 $O(e)$ 的 SPFA 算法就能体现其高效的优点了, 事实上, 这样的求最短路的题, SPFA 都能高效求解, 并且也拥有判负环功能, 所以很多情况下, SPFA 算法登场几率比较高。以下为本题 SPFA 算法的参考代码:

【参考程序——SPFA 算法】

```

#include<cstdio>
#include<cstring>
#include<iostream>

```

```

#include<algorithm>
using namespace std;
const int oo=1000000000;
const int M=100005;
int tot, Next[10 * M], to[10 * M], len[10 * M], head[M];
int n, m;
int d[M], mark[M], Q[10 * M];
void add(int a, int b, int c){
    Next[tot]=head[a], to[tot]=b, len[tot]=c;
    head[a]=tot++;
}
void SPFA(int g){
    for(int i=1; i<=n; i++) {
        d[i]=oo; mark[i]=0;           //初始化
    }
    int st=0, ed=0;
    Q[++ed]=g;
    d[g]=0, mark[g]=1;
    while(st<ed){
        int x=Q[++st];
        mark[x]=0;
        for(int i=head[x]; i!=-1; i=Next[i]){
            int y=to[i], z=len[i];
            if(d[y]>d[x]+z){
                d[y]=d[x]+z;
                if(! mark[y]){           //如果 y 被松弛,同时不在队列中,入队
                    mark[y]=1;
                    Q[++ed]=y;
                }
            }
        }
    }
}

int main(){
    scanf("%d%d", &n, &m);
    tot=0;
    for(int i=1; i<=n; i++) head[i]=-1;
    for(int i=1; i<=m; i++){
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        add(a, b, c);
        add(b, a, c);
    }
    SPFA(1);
    printf("%d\n", d[n]);
    return 0;
}

```

特别的,处理只有一个起点,一个终点的最短路,可以使用堆优化。使用堆替代队列,每次将最小权值的边弹出去松弛起点到其他点的距离,一旦弹出的点为终点,即可结束,因为

堆中已无更短的边来松弛终点了。对于此题,使用堆优化的 SPFA 算法如下:

【参考程序——堆优化的 SPFA 算法】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
#define i2 (i<<1)
#define i3 ((i<<1)|1)
const int M=100005;
const int oo=1000000000;
int n,m,dis[M],pos[M];
int tot,Next[M*10],to[M*10],len[M*10],head[M];
int hp[M],sz;
void put(int x){ //入堆
    hp[++sz]=x; pos[x]=sz;
    int i=sz;
    while(i>>1){
        int a=hp[i], b=hp[i>>1];
        if(dis[a]<dis[b])
            swap(pos[a], pos[b]), swap(hp[i], hp[i>>1]);
        //映射 x 在堆中的位置,因为在更新的时候,堆中的点都有可能更新
        else break;
        i>>=1;
    }
}

int get(){ //出堆
    int ret=hp[1];
    hp[1]=hp[sz--];
    int i=1,j;
    while(i2<=sz){
        int a=hp[i2], b=hp[i3];
        if(i3>sz || dis[a]<dis[b]) j=i2;
        else j=i3;
        a=hp[i], b=hp[j];
        if(dis[b]<dis[a]){
            swap(pos[a], pos[b]);
            swap(hp[i], hp[j]);
            i=j;
        }else break;
    }
    return ret;
}

void ajust(int x){ //调整更新后的点在堆中的位置
    int i=pos[x];
    while(i>>1){
        int a=hp[i], b=hp[i>>1];
        if(dis[a]<dis[b])
            swap(pos[a], pos[b]), swap(hp[i], hp[i>>1]);
    }
}
```

//映射 x 在堆中的位置,因为在更新的时候,堆中的点都有可能更新

```

    else break;
    i>>=1;
}
}
void add(int a,int b,int c){
    Next[tot]=head[a], to[tot]=b, len[tot]=c;
    head[a]=tot++;
}
void heapSPFA(){
    for(int i=1; i<=n; i++) dis[i]=oo, pos[i]=0;
    dis[1]=0;
    put(1);
    int test=0;
    while(sz){
        int x=get(); pos[x]=0;
        if(x==n) break;
        for(int i=head[x]; i!= -1; i=Next[i]){
            int y=to[i], z=len[i];
            if(dis[x]+z<dis[y]){
                dis[y]=dis[x]+z;
                if(! pos[y]) put(y);
                else ajust(y);
            }
        }
    }
}
void solve(){
    tot=sz=0;
    for(int i=1; i<=n; i++) head[i]=-1;
    for(int i=1; i<=m; i++){
        int a,b,c;
        scanf("%d%d%d",&a,&b,&c);
        add(a,b,c);
        add(b,a,c);
    }
    heapSPFA();
    if(dis[n]==oo) dis[n]=-1;
    printf("%d\n",dis[n]);
}
int main(){
    scanf("%d%d",&n,&m);
    solve();
    return 0;
}

```

第三节 图的连通性问题

1. 刻录光盘(cdrom)

在 FJOI2010 夏令营快要结束的时候,很多营员提出来要把整个夏令营期间的资料刻录成一张光盘给大家,以便大家回去后继续学习。组委会觉得这个主意不错!可是组委会一时没有足够的空光盘,没法保证每个人都能拿到刻录上资料的光盘,怎么办呢?

DYJ 分析了一下所有营员的地域关系,发现有些营员是一个城市的,其实他们只需要一张就可以了,因为一个人拿到光盘后,其他人可以带着 U 盘之类的东西去拷贝啊!

他们愿意某一些人到他那儿拷贝资料,当然也可能不愿意让另外一些人到他那儿拷贝资料,这与我们 FJOI 宣扬的团队合作精神格格不入!

现在假设总共有 $N(2 \leq N \leq 200)$ 个营员,每个营员的编号为 $1 \sim N$ 。DYJ 给每个人发了一张调查表,让每个营员填上自己愿意让哪些人到他那儿拷贝资料。当然,如果 A 愿意把资料拷贝给 B,而 B 又愿意把资料拷贝给 C,则一旦 A 获得了资料,则 B,C 都会获得资料。

现在,请你编写一个程序,根据回收上来的调查表,帮助 DYJ 计算出组委会至少要刻录多少张光盘,才能保证所有营员回去后都能得到夏令营资料。

输入:

先是一个数 N ,接下来的 N 行,分别表示各个营员愿意把自己获得的资料拷贝给其他哪些营员。即输入数据的第 $i+1$ 行表示第 i 个营员愿意把资料拷贝给那些营员的编号,以一个 0 结束。如果一个营员不愿意拷贝资料给任何人,则相应的行只有 1 个 0,一行中的若干数之间用一个空格隔开。

输出:

一个正整数,表示最少要刻录的光盘数。

样例输入:

```
5
2 4 3 0
4 5 0
0
0
1 0
```

样例输出:

```
1
```

【参考程序】

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
using namespace std;
int fa[201], tot=0, x, n;           //fa[i]表示 i 在并查集中的父亲
bool f[201][201]; //用邻接矩阵储存这张图,f[i][j]表示是否存在  $i \rightarrow j$  这条边
int main()
{
    cin >> n;
    for (int i=1; i <=n; ++i)
        fa[i]=i;           //初始化,把每个结点的父亲设为自己
    for (int i=1; i <=n; ++i)
```



```

{
    while (cin >> x && x)
        f[i][x]=true;           //邻接矩阵存图,i->x 连一条有向边
}
//f[i][j]的意义表示 i->j 是否有路径可达,则只要 i 得到了资料,j 也能得到资料
for (int k=1; k<=n;++k)
for (int i=1; i<=n;++i)
    for (int j=1; j<=n;++j)
        f[i][j]=f[i][j] || f[i][k] && f[k][j];
                                //利用 Floyed 的变形,求出 f[i][j]
for (int i=1; i<=n;++i)
for (int j=1; j<=n;++j)
if (f[i][j]) //如果 i->j 可行,则 i 有资料,j 也能得到,j 就不用刻录光盘
    fa[j]=fa[i];
//但出现独立的环时,环里面又必须有一个人要刻录光盘,则用 fa[j]=fa[i]就能巧妙地解决这种情况,因为总有一个人的 fa[i]会最后传到它自己,使得 fa[i]=i
for (int i=1; i<=n;++i)
    if (fa[i]==i) tot++; //判断,如果 i 没有其他父亲,那么 i 必须刻录光盘,tot+
+
cout << tot;           //输出最少要刻录的光盘数
return 0;
}

```

2. 珍珠(bead)

有 n 颗形状和大小都一致的珍珠,它们的重量都不相同。 n 为整数,所有的珍珠从 1 到 n 编号。你的任务是发现哪颗珍珠的重量刚好处于正中间,即在所有珍珠的重量中,该珍珠的重量列 $(n+1)/2$ 位。下面给出将一对珍珠进行比较的办法:

给你一架天平用来比较珍珠的重量,我们可以比出两个珍珠哪个更重一些,在作出一系列的比较后,我们可以将某些肯定不具备中间重量的珍珠拿走。

例如,下列给出对 5 颗珍珠进行四次比较的情况:

- (1)珍珠 2 比珍珠 1 重;
- (2)珍珠 4 比珍珠 3 重;
- (3)珍珠 5 比珍珠 1 重;
- (4)珍珠 4 比珍珠 2 重。

根据以上结果,虽然我们不能精确地找出哪个珍珠具有中间重量,但我们可以肯定珍珠 1 和珍珠 4 不可能具有中间重量,因为珍珠 2,4,5 比珍珠 1 重,而珍珠 1,2,3 比珍珠 4 轻,所以我们可以移走这两颗珍珠。

写一个程序统计出共有多少颗珍珠肯定不会是中间重量。

输入:

第 1 行包含两个用空格隔开的整数 N 和 M ,其中 $1 \leq N \leq 99$,且 N 为奇数, M 表示对珍珠进行的比较次数,接下来的 M 行每行包含两个用空格隔开的整数 x 和 y ,表示珍珠 x 比珍珠 y 重。

输出:

仅一行,包含一个整数,表示不可能是中间重量的珍珠的总数。

样例输入:

```

5 4
2 1
4 3
5 1

```

4 2

样例输出:

2

【参考程序 1】

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
#include <algorithm>
using namespace std;
int tot=0, t1, t2, n, m, x, y;
bool a[101][101], b[101][101];
//a[i][j]表示珍珠 i 是否比珍珠 j 重, b[i][j]则表示珍珠 i 是否比珍珠 j 轻
int main()
{
    scanf("%d%d", &n, &m);
    for (int i=1; i <=m; ++i)
    {
        scanf("%d%d", &x, &y);
        a[x][y]=b[y][x]=1;
    }
    //x 比 y 重, 则根据定义 a[x][y]=1, b[x][y]=1, 可以简单写成上一行这样
    //则将所有珍珠之间的关系转化成了两张有向图
    //a[i][j]表示在第一张图中, 是否存在 i->j 这条边
    //b[i][j]表示在第二张图中, 是否存在 i->j 这条边
    //用 Floyd 算法的变形, 求出分别在两张图中, i 到 j 是否有路径可达
    for (int k=1; k <=n; ++k)
        for (int i=1; i <=n; ++i)
            for (int j=1; j <=n; ++j)
            {
                if (a[i][k] && a[k][j]) a[i][j]=1;
                if (b[i][k] && b[k][j]) b[i][j]=1;
                //则此时 a 数组和 b 数组的意义发生了转变
                //a[i][j]表示在第一张图, i 到 j 是否有路径可达, 即珍珠 i 是否比珍珠 j 重
                //b[i][j]表示在第二张图, i 到 j 是否有路径可达, 即珍珠 i 是否比珍珠 j 轻
            }
    for (int i=1; i <=n; ++i)
    {
        t1=t2=0;
        //t1(t2)记录有多少珍珠比珍珠 i 轻(重)
        for (int j=1; j <=n; ++j)
        {
            if (a[i][j]) t1++;
            if (b[i][j]) t2++;
            //若 a[i][j]为真, 则珍珠 j 比珍珠 i 轻
            //若 b[i][j]为真, 则珍珠 j 比珍珠 i 重
        }
        if (t1 >=(n+1)/2) tot++;
        //若 t1 >=(n+1)/2, 则比珍珠 i 轻的不少于一半的, 那么珍珠 i 就不可能是中间重量
        if (t2 >=(n+1)/2) tot++;
    }
}

```

```

//若  $t2 \geq (n+1)/2$ , 则比珍珠  $i$  重的不少于一半的, 那么珍珠  $i$  就不可能是中间重量
}
printf("%d ", tot); //输出不可能是中间重量的珍珠总数
return 0;
}

```

【参考程序 2】

```

#include <cstring>
#include <iostream>
using namespace std;
int n, m, count = 0;
bool g1[201][201], g2[201][201], b[201];
//图 g1 中有  $x \rightarrow y$  这条边表示珍珠  $x$  比珍珠  $y$  重
//图 g2 中有  $x \rightarrow y$  这条边表示珍珠  $y$  比珍珠  $x$  重
void dfs(int p, bool g[][201]) //遍历统计比珍珠  $p$  重(轻)的珍珠数
{
    int i;
    b[p] = 1;
    for(i = 1; i <= n; i++)
        if(g[p][i] && ! b[i])
            dfs(i, g); //递归遍历
    count++; //遍历到一个, 计数器加 1
}

int main()
{
    int i, x, y, tot = 0;
    cin >> n >> m;
    for(i = 1; i <= m; i++)
    {
        cin >> x >> y;
        g1[x][y] = g2[y][x] = 1; //  $x \rightarrow y$  连边
    }
    for(i = 1; i <= n; i++) //统计多少个珍珠比  $(n+1)/2$  以上个数的珍珠重
    {
        count = 0; //初始化 count
        memset(b, 0, sizeof(b));
        dfs(i, g1);
        if(count > (n+1)/2) tot++; //这个珍珠不可能是中间重量
    }
    for(i = 1; i <= n; i++) //统计多少个珍珠比  $(n+1)/2$  以上个数的珍珠轻
    {
        count = 0;
        memset(b, 0, sizeof(b));
        dfs(i, g2);
        if(count > (n+1)/2) tot++; //这个珍珠不可能是中间重量
    }
    cout << tot;
    return 0;
}

```

第四节 并查集

1. 团伙(group)

时间限制 2s

在某城市里住着 n 个人,任何两个认识的人不是朋友就是敌人,而且满足:

(1)我朋友的朋友是我的朋友;

(2)我敌人的敌人是我的朋友。

所有是朋友的人组成一个团伙。告诉你关于这 n 个人的 m 条信息,即某两个人是朋友,或者某两个人是敌人,请你编写一个程序,计算出这个城市最多可能有多少个团伙?

输入:

第 1 行为 n 和 m , $1 < n < 1000$, $1 \leq m \leq 100,000$;

以下 m 行,每行为 $p \ x \ y$, p 的值为 0 或 1, p 为 0 时,表示 x 和 y 是朋友, p 为 1 时,表示 x 和 y 是敌人。

输出:

一个整数,表示这 n 个人最多可能有几个团伙。

输入:

```
6 4
1 1 4
0 3 5
0 4 6
1 1 2
```

输出:

```
3
```

【参考程序 1】

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
#include <algorithm>
using namespace std;
//根据题意,我们把是朋友的并到一个集合中
//则最后只要统计共有几个集合,即为最多有几个团伙
int father[1001], a[1001];
//father[i]即为 i 在集合中的父亲, a[i]表示 i 的第一个敌人(没有敌人时为 0)
int n, m, flag, x, y, tot=0;
//tot 记录最后答案
int find(int x) //在并查集中查询 x 所在集合的根结点
{
    if (father[x] != x)
        father[x]=find(father[x]);
    return father[x];
}
void unionn(int x, int y) //合并 x,y 所在集合
```

```

{
    int a=find(x);
    int b=find(y);
    father[a]=b;
}
int main()
{
    scanf("%d%d", &n, &m);
    for (int i=1; i <=n; ++i)
        father[i]=i; //初始化每个元素为单独一个集合,以每个元素自己作为根结点
    for (int i=1; i <=m; ++i)
    {
        scanf("%d%d%d", &flag, &x, &y);
        if (flag==0)
            unionn(x, y); //如果两个人是朋友,直接合并
        else
        { //如果两个人是敌人,则将两人的敌人分别和另外一人合并(敌人的敌人是朋友)
            if (! a[x])
                a[x]=y; //如果 x 还没有敌人,先将 y 设为 x 的敌人
            else
                unionn(a[x], y); //因为 x 的敌人的敌人都为朋友,将他们并到一个集合中
                //这里只要取其中一个敌人做代表,把他与 y 所在的集合合并即可
            if (! a[y]) //对于 y 同理
                a[y]=x;
            else
                unionn(a[y], x);
        }
    }
    for (int i=1; i <=n; ++i)
        if (father[i]==i)
            tot++; //如果 father[i]==i,则 i 是这个集合的根
    //由于根只有一个,所以满足 father[i]==i 的总数就是团伙的总个数,记为 tot
    printf("%d", tot);
    return 0;
}

```

【参考程序 2】

```

#include<iostream>
#include<cstdio>
#include<cmath>
#include<cstdlib>
#include<cstring>
#include<algorithm>
using namespace std;
int a[10001],b[1001][1001];
//a[i]表示 i 在集合中的父亲,b[i][j]表示 i 与 j 是否是敌人,1 是,0 不是
int find(int x) //查询 x 所在集合的根节点
{

```

```

    if (a[x] != x) a[x] = find(a[x]);
    return a[x];
}

void give(int fa, int fb)           //合并 fa,fb 的集合
{
    fa = find(fa); fb = find(fb);
    a[fb] = fa;
}

int main()
{
    int i, n, t, j, k, s = 0, m, p, x, y;
    cin >> n >> m;
    for(i = 1; i <= n; i++) a[i] = i; //初始化每一个节点为本身的根节点,各为集合
    for(i = 1; i <= m; i++)
    {
        cin >> p >> x >> y;
        if (p == 0) give(x, y);      //直接合并
        else
        {
            b[x][y] = b[y][x] = 1;    //将 x,y 互设为敌人
            for(j = 1; j <= n; j++)
            {
                if (b[x][j]) give(y, j); //将 x 的敌人与 y 合并(敌人的敌人都是朋友)
                if (b[y][j]) give(x, j); //y 同理
            }
        }
    }
    for(i = 1; i <= n; i++)
        if (a[i] == i) s++; //因为每个集合只有一个根,当根为 i 本身时,则有一个集合(团伙)
    cout << s << endl;
    return 0;
}

```

2. 打击犯罪(black)

某个地区有 n ($n \leq 1000$) 个犯罪团伙,当地警方按照他们的危险程度由高到低给他们编号为 $1 \sim n$,他们有些团伙之间有直接联系,但是任意两个团伙都可以通过直接或间接的方式联系,这样这里就形成了一个庞大的犯罪集团,犯罪集团的危险程度由集团内的犯罪团伙数量唯一确定,而与单个犯罪团伙的危险程度无关(该犯罪集团的危险程度为 n)。现在当地警方希望花尽量少的时间(即打击掉尽量少的团伙),使得庞大的犯罪集团分离成若干个较小的集团,并且他们中最大的一个的危险程度不超过 $n/2$ 。为达到最好的效果,他们将按顺序打击掉编号 1 到 k 的犯罪团伙,请编程求出 k 的最小值。

输入:

第 1 行一个正整数 n 。接下来的 n 行每行有若干个正整数,第一个整数表示该行除第一个外还有多少个整数,若第 i 行存在正整数 k ,表示 i, k 两个团伙可以直接联系。

输出:

一个正整数,为 k 的最小值

样例输入:

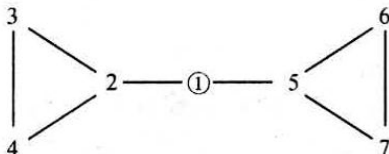
7

2 2 5
 3 1 3 4
 2 2 4
 2 2 3
 3 1 6 7
 2 5 7
 2 5 6

样例输出:

1

【提示】



输出 1(打击掉犯罪团伙)

【参考程序 1】

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
const int N=1005;
bool F;
int n,f[N],a[N][N],s[N];
inline int get()          //读入优化
{
    char ch; bool f=false;
    while ((ch=getchar())<'0' || ch>'9')
    if (ch=='-') f=true;
    int res=ch-48;
    while ((ch=getchar())>='0' && ch<='9')
    res=res*10+ch-48;
    return f? ~res+1 : res;
}
inline int find(int x)    //查找 x 所在集合的根节点
{
    if (f[x] != x) f[x]=find(f[x]);
    return f[x];
}
int main()
{
    n=get();
    for (int i=1; i<=n; ++i)
    {
        a[i][0]=get(); f[i]=i; s[i]=1;
        for (int j=1; j<=a[i][0]; ++j) a[i][j]=get();
    }
    for (int i=n; i>0; --i)
```

```

{
    if (F) break;
    for (int j=1; j<=a[i][0]; ++j)
        if (a[i][j]>i)
            //打击到第 i 个团伙时,1~i-1 的团伙已被打击过了
            //因此利用并查集合并 i~n(未被打击)的团伙
            {
                int tx=find(i),
                    ty=find(a[i][j]);
                if (tx!=ty)
                {
                    f[ty]=tx;
                    s[tx]+=s[ty];    //计算未被打击团伙的总危险程度
                    if (s[tx]>(n+1>>1))/">>"是位运算中的右移,n+1>>1 相当于(n+1)/2
                    { /* 逆向思维:如果未被打击团伙的危险程度大于(n+1)/2 即[n-n/2],
                        则此时打击到当前团伙才能保证危险程度不超过 n/2 */
                        printf("%d\n",i);
                        F=true;
                        break;
                    }
                }
            }
    }
}
return 0;
}

```

【参考程序 2】

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
#include <algorithm>
using namespace std;

```

//分析:如果按题意从 1 到 n 枚举,去掉 1~k 的点,判断每次统计最大集合点数是否不超过 n/2,若是,则为答案,但这么做每次枚举都要重置并查集,考虑倒过来思考:

//从 n 到 1 逆向枚举,每次把点 k 加入图中,也就是删除 1~k-1,剩余 k~n,若最大集合点数不超过 n/2,说明这种方案可行,k 还能更小,一旦不满足,意味着第 k 个点加进图中就不可行了,即点 k 必须删,k 不能更小了,就输出答案。

```
int n, a[1001][1001];
```

//把他们的关系用邻接矩阵储存为一张图:a[i][j]表示是否有 i->j 这条边

```
int c[1001], f[1001];    //c[i],f[i]分别表示 i 所在集合的总个数和根
```

```
int find(int x)          //查找 x 所在集合的根
```

```

{
    if (f[x] != x)
        f[x]=find(f[x]);
}

```



```

    return f[x];
}

int main()
{
    cin >> n;
    for (int i=1; i <=n; ++i)
    {
        c[i]=1;
        f[i]=i; //初始化,把每个点看作一个集合,其根为它本身,个数为 1
    }
    for (int i=1; i <=n; ++i)
    {
        cin >> a[i][0];
        for (int j=1; j <=a[i][0]; ++j)
            cin >> a[i][j]; //将关系转化为边
    }
    for (int k=n; k>=1; --k) //从 n 到 1 逆向枚举(每次把点 k 加入图中)
        for (int i=1; i<=a[k][0]; ++i) //把与其相连的点并到 k 所在的集合里
            if (a[k][i] > k) //如果这个点现在是在图里,即大于 k,才能进行操作
            {
                int x=find(k);
                int y=find(a[k][i]);
                if (x != y)
                {
                    f[y]=x; //将这个点并到 k 所在的集合中
                    c[x]+=c[y]; //累加集合的元素个数
                    if (c[x] > n / 2)
                        { //若集合点数超过 n/2,则点 k 不能加进图中,必须删,k 也不能更小
                            cout << k; //输出答案
                            return 0; //直接退出程序
                        }
                }
            }
    }
}

```

3. 搭配购买(buy)

Joe 觉得云朵很美,决定去山上的商店买一些云朵。商店里有 n 朵云,云朵被编号为 $1, 2, \dots, n$, 并且每朵云都有一个价值。但是商店老板跟他说,一些云朵要搭配来买才好,所以买一朵云则与这朵云有搭配的云都要买。

但是 Joe 的钱有限,所以他希望买的价值越多越好。

输入:

第 1 行三个数: n, m, w , 表示 n 朵云, m 个搭配, Joe 有 w 的钱。

第 2~ $n+1$ 行, 每行两个数: c_i, d_i , 表示 i 朵云的价钱和价值。

第 $n+2 \sim n+1+m$ 行, 每行两个数: u_i, v_i , 表示买 u_i 就必须买 v_i , 同理, 如果买 v_i 就必须买 u_i 。

输出:

一行, 表示可以获得的最大价值。

样例输入:

```
5 3 10
3 10
3 10
3 10
5 100
10 1
1 3
3 2
4 2
```

样例输出:

```
1
```

【数据范围】

30%的数据保证: $n \leq 100$;

50%的数据保证: $n \leq 1,000$; $m \leq 100$; $w \leq 1,000$;

100%的数据保证: $n \leq 10,000$; $0 \leq m \leq 5000$; $w \leq 10,000$ 。

【参考程序 1】

```
#include<cstdio>
#include<algorithm>
#include<string>
#include<cstring>
#include<cstdlib>
#include<cmath>
#include<iostream>
using namespace std;

int n,m,val;

inline int read()                //读入优化
{
    char c;
    bool flag=false;
    while((c=getchar())>'9' || c<'0')
    if(c=='-')flag=true;
    int res=c-'0';
    while((c=getchar())>='0' && c<='9')
    res=(res<<3)+(res<<1)+c-'0';
    return flag? -res : res;
}

const int N=11000;
int tot,father[N];
int w[N],f[N],v[N];
inline int find(int x)           //查找根节点
{
    if(father[x]!=x) father[x]=find(father[x]);
    return father[x];
}
```

```

int main()
{
    n=read();
    m=read();
    val=read();
    for(int i=1;i<=n;++i)
    {
        w[i]=read();
        v[i]=read();
        father[i]=i;          //初始化集合
    }
    int x,y;
    for(int i=1;i<=m;++i)
    {
        x=read();
        y=read();
        int fx=find(x),fy=find(y);
        if(fx!=fy)
            father[fy]=fx,w[fx]+=w[fy],v[fx]+=v[fy];
        //先将每组搭配购买的物品的总花费、总价值算出来
    }
    for(int i=1;i<=n;++i)
    {
        if(father[i]==i)
        {
            w[++tot]=w[i];
            v[tot]=v[i];
        }
        //只需要买一个,就代表买了所有同组应搭配购买的物品
    }
    for(int i=1;i<=tot;++i) //01 背包求解
    {
        for(int j=val;j>=w[i];--j)
            f[j]=max(f[j-w[i]]+v[i],f[j]);
        printf("%d",f[val]);
    }
}

【参考程序 2】
#include<cmath>
#include<cstdio>
#include<cstdlib>
#include<cstring>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn=1e4+1;
const int maxm=5e3+1;
const int maxw=1e4+1;
int c[maxn],d[maxn],f[maxw];
int fat[maxn],pri[maxn],val[maxn];
int Find(int x)
{
    return (fat[x]!=x)?fat[x]=Find(fat[x]):x;
}

```

```

{
    case '#': cin >> s1; break;
    case '+':
        cin >> s2;
        if (s1 != s2) //利用并查集合并最早祖先
            f[father(w(s2))] = father(w(s1));
        break;
    case '?':
        cin >> s2; //查询最早祖先
        cout << s2 << " " << b[father(w(s2))] << endl;
        break;
}
} while (c != '$');
}

```

【参考程序 2】

```

#include <iostream>
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cmath>
#include <string>
#include <algorithm>
using namespace std;
//这一题中,把拥有共同祖先的元素并到一个集合,其根结点就为最早的祖先
int t; //t 记录编号总数
char ch;
string s1, s2, b[50001]; //b[i]表示编号为 i 的字符串
int father[50001]; //father[i]表示 i 在并查集中的父亲
int weiz(string s) //找到字符串 s 在并查集中的序号
{
    for (int i=1; i<=t; ++i)
        if (b[i]==s)
            return i; //如果在之前出现过,直接返回序号
    t++;
    b[t]=s;
    return t; //如果没有,则新增加一个编号
}
int find(int x) //寻找 x 所在的并查集中的根
{
    if (father[x] != x)
        father[x]=find(father[x]);
    return father[x];
}
int main()
{
    for (int i=1; i<=50000; ++i)
        father[i]=i; //开始将每个元素都看作一个独立的集合,其根就为它本身
}

```

```

do
{
    cin >> ch;
    switch (ch)
    {
        case '#':
            cin >> s1;          //s1 记录该组父子关系中的父亲名字
            break;
        case '+':
            cin >> s2;          //这里 s2 记录儿子名字
            if (s1 != s2)        //将 s2 所在的集合并到 s1 所在的集合下面
                father[find(weiz(s2))]=find(weiz(s1));
                //即把 s2 所在的集合的元素的最早祖先都设为 s1 的最早祖先
            break;
        case '?':
            cin >> s2;          //这里 s2 记录要询问的名字
            cout << s2 << " << b[find(weiz(s2))] << endl;
                //输出本人名字及最早祖先的名字
            break;
    }
} while (ch != '$');          //当输入"$"时停止
return 0;
}

```

5. 亲戚

若某个家族人员过于庞大,要判断两个是否是亲戚,确实还很难,现在给出某个亲戚关系图,求任意给出的某个人所在家族的人数。

规定: x 和 y 是亲戚, y 和 z 是亲戚,那么 x 和 z 也是亲戚。如果 x, y 是亲戚,那么 x 的亲戚都是 y 的亲戚, y 的亲戚也都是 x 的亲戚。

输入:

第 1 行:两个整数 n, m ($n \leq 100,000, m \leq 200,000$), 分别表示有 n 个人, m 个信息。

以下 m 行, 信息包含两种形式:

M a b: 表示 a 和 b 具有亲戚关系;

Q a: 要求输出 a 所在家族的人数。

输出:

要求输出 a 所在家族的人数。

样例输入:

```

5 10
M 3 2
Q 4
M 1 2
Q 4
M 3 2
Q 1
M 3 1
Q 5
M 4 2
Q 4

```

样例输出：

1
1
3
1
4

【算法分析】

本题求 a 所在家族的人数,我们可以将家族看成集合,如果两个人是亲戚,就合并他们所在的集合,询问即为求集合内元素的数量,所以我们可以用并查集解决该问题,增加一个数组记录集合内元素数量,用于求答案。

【参考程序——并查集】

```
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
const int M=100005;
int n,m;
int f[M],num[M];           //f[i]表示 i 的根节点,num[i]表示该集合内元素数量
int find(int x){           //查找根节点
    if(x!=f[x]) f[x]=find(f[x]);
    return f[x];
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=n;i++){
        f[i]=i,num[i]=1;    //初始化集合
    }
    for(int i=1;i<=m;i++){
        char op[3];
        scanf("%s",op);     //输入一个字符也看以看成字符串输入
        if(op[0]=='M'){       //合并集合
            int a,b;
            scanf("%d%d",&a,&b);
            int u=find(a),v=find(b);
            if(u==v) continue;
            f[v]=u;
            num[u]+=num[v];
        }else{
            int a;
            scanf("%d",&a);
            int u=find(a);
            printf("%d\n",num[u]);
        }
    }
    return 0;
}
```

6. 食物链【NOI2001】

动物王国中有三类动物 A,B,C,这三类动物的食物链构成了有趣的环形:A 吃 B, B 吃 C,C 吃 A。

现有 N 个动物,以 1~N 编号。每个动物都是 A,B,C 中的一种,但是我们并不知道它到底是哪一种。

有人用两种说法对这 N 个动物所构成的食物链关系进行描述:

第 1 种说法是“1 X Y”,表示 X 和 Y 是同类;

第 2 种说法是“2 X Y”,表示 X 吃 Y。

此人对 N 个动物,用上述两种说法,一句接一句地说出 K 句话,这 K 句话有的是真的,有的是假的。当一句话满足下列三条之一时,这句话就是假话,否则就是真话。

(1)当前的话与前面的某些真的话冲突,就是假话;

(2)当前的话中 X 或 Y 比 N 大,就是假话;

(3)当前的话表示 X 吃 X,就是假话。

你的任务是根据给定的 N($1 \leq N \leq 50,000$)和 K($0 \leq K \leq 100,000$)句话,输出假话的总数。

输入:

第 1 行是两个整数 N 和 K,以一个空格分隔。

以下 K 行每行是三个正整数 D,X,Y,两数之间用一个空格隔开,其中 D 表示说法的种类。

若 $D=1$,则表示 X 和 Y 是同类。

若 $D=2$,则表示 X 吃 Y。

输出:

只有一个整数,表示假话的数目。

样例输入:

```
100 7
1 101 1
2 1 2
2 2 3
2 3 3
1 1 3
2 3 1
1 5 5
```

样例输出:

```
3
```

【样例解释】

100 7	
1 101 1	假话
2 1 2	真话
2 2 3	真话
2 3 3	假话
1 1 3	假话
2 3 1	真话
1 5 5	真话

【算法分析】

本题要将动物分成 3 类,看似要 3 个集合,无法用并查集做。但如果我们在三类关系间引入相对距离(偏移量),判断两个动物相对距离的关系,就能确定 2 个动物的食物链。三类动物的相对距离为 0,1,2,令 0 吃 1,1 吃 2,2 吃 0,从数值上,我们发现如果 a 吃 b,那么 $(a-b+3)\%3=2$ 。由此,我们对于“同类”和“吃”两种操作,还是可以利用并查集合并,具体有:

(1)如果 a, b 同类,我们首先判断 a, b 是属于同一集合(是否有相对关系),如果属于同一集合,就可判断是否是假话;否则,就要合并 a, b, 令 u, v 分别为 a, b 的父亲,不失一般性,我们可以让 b 的父亲 v 合并到 a 上面,现在问题就是 v 到 a 的距离定义为多少。令 $dis[b]$ 表示 b 到父亲 v 的距离,而最终 b 和 a 要是同类,那么 b 到 a 的距离 $(dis[b]+dis[v])\%3=0$, 即 $dis[v]=3-dis[b]$ 。

(2)如果 a 吃 b,我们也要先判断 a, b 是否属于同一集合,如果属于同一集合,就可以判断真假;否则,也需要合并 a, b, 让 a, b 产生相对距离。令 u, v 分别为 a, b 的父亲,同上面的方法, v 合并到 a, b 到 a 的距离 $(0-(dis[b]+dis[v])+3)\%3=2$, 即 $dis[v]=(1-dis[b]+3)\%3$ 。

剩下的问题,就是如何维护 dis 数组。在 x 点找父亲过程中, x 的父亲会不断变化,直到遇到最终的父亲,那么 $dis[x]$ 也就一直需要跟着变化。但其实,我们发现在递归找 x 父亲的时候, x 原父亲 y 离最终父亲的距离能计算出来,那么 x 与最终父亲的距离就为 $(dis[x]+dis[y])\%3$, 至此所有问题可以解决,具体实现参考如下代码。

【参考程序——关系并查集】

```
//0 吃 1, 1 吃 2, 2 吃 0
//(0-1)+3=2, (1-2)+3=2, 2-0=2,
//即  $(dis[x]-dis[y]+3)\%3==2$  表示 x 吃 y
#include<cstdio>
#include<cstring>
#include<iostream>
using namespace std;
const int M=50005;
int f[M], dis[M];
int n, m;
int find(int x){
    if(f[x]!=x){
        int y=f[x]; //先记录 x 之前的父亲
        f[x]=find(f[x]);
        dis[x]=(dis[x]+dis[y])%3;
        //x 到父亲的距离为: x 到 y 的距离+y 到最终父亲的距离
    }
    return f[x];
}

int main(){
    int res=0;
    scanf("%d%d", &n, &m);
    for(int i=1; i<=n; i++) f[i]=i, dis[i]=0;
    for(int i=1; i<=m; i++){
        int a, b, c;
        scanf("%d%d%d", &a, &b, &c);
        if(b>n || c>n){
            res++; continue;
        }
    }
}
```



```

}
if(a==1){
    if(b==c) continue;
    int u=find(b), v=find(c);
    if(u==v){ //在同一集合里
        if(dis[b]! =dis[c]) res++; //不是同类就是假话
    }else{ //不在同一集合里
        dis[v]=3-dis[c]; //先重新计算距离,保证 c 和 b 同类
        f[v]=b; //将 c 的父亲合并到 b
    }
}
} else{
    if(b==c) {
        res++; continue;
    }
    int u=find(b), v=find(c);
    if(u==v){ //在同一集合里
        if( (dis[b]-dis[c]+3)%3 !=2 )
            res++; //不是吃的关系就是假话
    }else{
        dis[v]=(1-dis[c]+3)%3; //先重新计算距离,保证 b 吃 c
        f[v]=b; //将 c 的父亲合并到 b
    }
}
}
}
printf("%d\n",res);
return 0;
}

```

第五节 最小生成树

1. 局域网(net)

某个局域网内有 $n(n \leq 100)$ 台计算机, 由于搭建局域网时工作人员的疏忽, 现在局域网内的连接形成了回路, 我们知道如果局域网形成回路那么数据将不停地在回路内传输, 造成网络卡的现象。因为连接计算机的网线本身不同, 所以有一些连线不是很畅通, 我们用 $f(i, j)$ 表示 i, j 之间连接的畅通程度 ($f(i, j) \leq 1000$), $f(i, j)$ 值越小表示 i, j 之间连接越通畅, $f(i, j)$ 为 0 表示 i, j 之间无网线连接。现在我们需要解决回路问题, 我们将除去一些连线, 使得网络中没有回路, 并且被除去网线的 $\sum f(i, j)$ 最大, 请求出这个最大值。

输入:

第 1 行两个正整数 n, k ;

接下来的 k 行每行三个正整数 i, j, m , 表示 i, j 两台计算机之间有网线联通, 通畅程度为 m 。

输出:

一个正整数, $\sum f(i, j)$ 的最大值。

样例输入:

```
5 5
1 2 8
1 3 1
1 5 3
2 4 5
3 4 2
```

样例输出:

```
8
```

【参考程序 1】

```
#include <iostream> //prim 算法
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <string>
#include <algorithm>
```

```
using namespace std;
```

```
//分析:要使去除的边权最大,那么就相当于剩余的要最小,而问题就转化成了最小生成树
//(以下网线的通畅程度都称边权)
```

```
int n, m, a, b, c, sum=0;
```

```
//n 表示点数, m 表示边数, sum 记录总边权(即总通畅程度)
```

```
int g[101][101], minn[101];
```

```
//用邻接矩阵储存, g[i][j] 表示  $i \rightarrow j$  这条边的边权(不存在时设为一个很大的数)
```

```
//minn[i] 存放还未加入最小生成树的结点 i, 到已经加入最小生成树结点的最小边权
```

```
bool u[101];
```

```
//u[i] 表示结点 i 是否还未加入最小生成树
```

```

int main()
{
    memset(g, 0x7f, sizeof(g));
    memset(minn, 0x7f, sizeof(minn));
    memset(u, true, sizeof(u)); //初始化
    scanf("%d%d", &n, &m);
    for (int i=1; i <= m; ++i)
    {
        scanf("%d%d%d", &a, &b, &c);
        g[a][b]=g[b][a]=c; //在 a,b 之间连一条权为 c 的双向边
        sum+=c; //统计总边权
    }
    minn[1]=0; //直接把一个结点先加入最小生成树
    for (int i=1; i <= n; ++i) //Prim 算法求最小生成树
    {
        int k=0;
        for (int j=1; j <= n; ++j)
            if (u[j] && minn[j] < minn[k])
                k=j; //找到目前还未加入最小生成树且 minn 最小的结点,记为 k
        u[k]=false; //则选择 k 加入最小生成树
        for (int j=1; j <= n; ++j)
            if (u[j] && g[k][j] < minn[j])
                minn[j]=g[k][j]; //利用 k 去更新还未加入最小生成树的结点的 minn 值
    }
    int tot=0;
    for (int i=1; i <= n; ++i)
        tot+=minn[i]; //统计最小生成树的总权值
    printf("%d", sum-tot); //输出要去掉边的总权值
    return 0;
}

```

【参考程序 2】

```

#include<iostream> //kruska 算法
#include<algorithm>
using namespace std;
struct dy
{
    int x;
    int y;
    int z;
}; //定义结构类型,表示边(x-y 通畅度为 z)
dy a[4951]; //存边
int father[101];

int cmp(const dy &a,const dy &b)//快排自定义比较函数
{
    if (a.z<=b.z) return 1;
}

```

```

    return 0;
}

int find(int x)                //寻找 x 所在集合的根节点
{
    if (father[x] != x) father[x] = find(father[x]);
    return father[x];
}

int main()
{
    int n, m, k = 0, ans = 0;
    cin >> n >> m;
    for (int i = 1; i <= m; ++i)
    {
        cin >> a[i].x >> a[i].y >> a[i].z;
        ans += a[i].z;           //初始时,假定所有网络都要除去
    }
    for (int i = 1; i <= n; ++i) father[i] = i;
    sort(a + 1, a + m + 1, cmp); //将网络按通畅程度从小到大排序
    for (int i = 1; i <= m; ++i)
    {
        int fx = find(a[i].x), fy = find(a[i].y);
        if (fx != fy)
        {
            father[fy] = fx;
            k++;
            ans -= a[i].z;       //每保留一条网线,就减去这条网线的通畅程度
        }
        if (k == n - 1) break;
    }
    cout << ans << endl;
    return 0;
}

```

2. 繁忙的都市(city)

城市 C 是一个非常繁忙的大都市,城市中的道路十分的拥挤,于是市长决定对其中的道路进行改造。城市 C 的道路是这样分布的:城市中有 n 个交叉路口,有些交叉路口之间有道路相连,两个交叉路口之间最多有一条道路相连接。这些道路是双向的,且把所有的交叉路口直接或间接地连接起来了。每条道路都有一个分值,分值越小表示这个道路越繁忙,越需要进行改造。但是市政府的资金有限,市长希望进行改造的道路越少越好,于是他提出下面的要求:

- (1) 改造的那些道路能够把所有的交叉路口直接或间接的连通起来;
- (2) 在满足要求(1)的情况下,改造的道路尽量少;
- (3) 在满足要求(1)(2)的情况下,改造的那些道路中分值最大值尽量小。

【编程任务】

作为市规划局的,应当作出最佳的决策,选择哪些道路应当被修建。

输入:

第 1 行有两个整数 $n, m (1 \leq n, m \leq 300)$ 表示城市有 n 个交叉路口, m 条道路。接下来 m 行是对每条道路的描述, $u, v, c (1 \leq c \leq 10,000)$ 表示交叉路口 u 和 v 之间有道路相连, 分值为 c 。

输出:

两个整数 $s, \max$, 表示你选出了几条道路, 分值最大的那条道路的分值是多少。

样例输入:

```
4 5
1 2 3
1 4 5
2 4 7
2 3 6
3 4 8
```

样例输出:

```
3 6
```

【参考程序 1】

```
#include <iostream>                                //Prim 算法
#include <cstdio>
#include <cstring>
#include <cmath>
#include <cstdlib>
#include <string>
#include <algorithm>
using namespace std;

int n, m, a, b, c, maxx=-0x7fffffff;
//n,m 分别表示点数和边数,maxx 记录最小生成树中最大的边权
int g[301][301], minn[301];
//用邻接矩阵储存,g[i][j]表示 i->j 这条边的边权(不存在时设为一个很大的数)
//minn[i]存放还未加入最小生成树的结点 i,到已经加入最小生成树结点的最小边权
bool u[301];u[i]表示结点 i 是否还未加入最小生成树

int main()
{
    memset(g, 0x7f, sizeof(g));
    memset(minn, 0x7f, sizeof(minn));
    memset(u, true, sizeof(u));                //初始化

    scanf("%d%d", &n, &m);
    for (int i=1; i<=m; ++i)
    {
        scanf("%d%d%d", &a, &b, &c);
        g[a][b]=g[b][a]=c;                    //在 a,b 之间连一条边权为 c 的双向边
    }
    minn[1]=0;                                  //直接把一个结点先加入最小生成树
    for (int i=1; i<=n; ++i)
    {
```

```

int k=0;
for (int j=1; j <=n; ++j)
    if (u[j] && minn[j] < minn[k])
        k=j; //找到目前还未加入最小生成树且 minn 最小的结点,记为 k
u[k]=false; //则选择 k 加入最小生成树
maxx=max(maxx, minn[k]); //记录最小生成树中最大的边权

for (int j=1; j <=n; ++j)
    if (u[j] && g[k][j] < minn[j])
        minn[j]=g[k][j]; //利用 k 去更新还未加入最小生成树的结点的 minn 值
}

printf("%d %d ", n-1, maxx); //有 n 个结点的最小生成树边数即为 n-1
return 0;
}

```

【参考程序 2】

```

#include<iostream> //kruska 算法
#include<algorithm>
using namespace std;
struct dy
{
    int x;
    int y;
    int z;
}; //定义结构类型,表示边(x-y 分值为 z)
dy a[44851]; //存边
int father[301];
int cmp(const dy &a,const dy &b) //快排自定义比较函数
{
    if (a.z<=b.z) return 1;
    return 0;
}
int find(int x) //寻找 x 所在集合的根节点
{
    if (father[x] != x) father[x]=find(father[x]);
    return father[x];
}
int main()
{
    int n,m,k=0,maxx=0;
    cin>>n>>m;
    for (int i=1;i<=m;++i) cin>>a[i].x>>a[i].y>>a[i].z;
    for (int i=1;i<=n;++i) father[i]=i; //初始时,每个交叉路口各不相通
    sort(a+1,a+m+1,cmp); //将道路按分值排列,保证改造的道路中分值最小的最小
    for (int i=1;i<=m;++i)
    {
        int fx=find(a[i].x),fy=find(a[i].y);
    }
}

```

```

    if (fx != fy)                //进行连边(即修建)
    {
        father[fy] = fx;
        k++;
        maxx = a[i].z;          //maxx 记录道路的分值,由于已经排序过,不需要比较
    }
    if (k == n - 1) break;        //终止条件为 n 个交叉路口都相通
}
cout << n - 1 << " " << maxx << endl;
return 0;
}

```

3. 联络员(liaison)

Tyvj 已经一岁岁了,网站也由最初的几个用户增加到了上万个用户,随着 Tyvj 网站的逐步壮大,管理员的数目也越来越多,现在你身为 Tyvj 管理层的联络员,希望你找到一些通信渠道,使得管理员两两都可以联络(直接或者是间接都可以)。Tyvj 是一个公益性的网站,没有过多的利润,所以你要尽可能地使费用少才可以。

目前你已经知道,Tyvj 的通信渠道分为两大类,一类是必选通信渠道,无论价格多少,你都需要把所有的都选择上;还有一类是选择性的通信渠道,你可以从中挑选一些作为最终管理员联络的通信渠道。数据保证给出的通行渠道可以让所有的管理员联通。

输入:

第 1 行 n, m 表示 Tyvj 一共有 n 个管理员,有 m 个通信渠道;

第 2 行到 $m+1$ 行,每行四个非负整数 p, u, v, w ,当 $p=1$ 时,表示这个通信渠道为必选通信渠道;当 $p=2$ 时,表示这个通信渠道为选择性通信渠道; u, v, w 表示本条信息描述的是 u, v 管理员之间的通信渠道, u 可以收到 v 的信息, v 也可以收到 u 的信息, w 表示费用。

输出:

最小的通信费用。

样例输入:

```

5 6
1 1 2 1
1 2 3 1
1 3 4 1
1 4 1 1
2 2 5 10
2 2 5 5

```

样例输出:

```
9
```

【样例解释】

1-2-3-4-1 存在四个必选渠道,形成一个环,互相可以到达。需要让所有管理员联通,需要联通 2 号和 5 号管理员,选择费用为 5 的渠道,所以总的费用为 9。

【注意】

u, v 之间可能存在多条通信渠道,你的程序应该累加所有 u, v 之间的必选通行渠道。

【数据范围】

对于 30% 的数据, $n \leq 10, m \leq 100$;

对于 50% 的数据, $n \leq 200, m \leq 1000$;

对于 100% 的数据, $n \leq 2000, m \leq 10,000$ 。

【参考程序 1】

```
#include <cmath>                //Kruskal 算法
```

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;

const int Maxn=2010;
const int Maxm=20010;

struct point
{
    int x, y;           //x,y 表示这条边连接的两点
    int c;              //c 表示这条边的权值
}a[Maxm];              //用结构体存边,方便排序

int n, m, ans, tot;     //ans 记录答案,tot 记录边数
int father[Maxn];      //father[i]记录 i 在并查集中的父亲

int find(int x)         //在并查集中查找 x 所在集合的根
{
    if (father[x] != x) father[x]=find(father[x]);
    return father[x];
}

void unionn(int x, int y) //合并 x,y 所在的集合
{
    int a=find(x);
    int b=find(y);
    if (a != b) father[a]=b;
}

bool cmp(const point &a, const point &b) //sort 的比较函数,按边权排序(从小到大)
{
    return a.c < b.c;
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i=1; i <=n; ++i)
        father[i]=i;           //初始化 father
    for (int i=1, p, u, v, w; i <=m; ++i)
    {
        scanf("%d%d%d%d", &p, &u, &v, &w);
        if (p==1)
        { //如果是必选的,则直接将两点的集合合并(连接两点),累加总花费
```



```

        unionn(u, v);
        ans += w;
    }
    else
    { //如果是可选的,把这条边加到 a 数组里,进行最小生成树算法
        ++tot;
        a[tot].x = u;
        a[tot].y = v;
        a[tot].c = w;
    }
}

sort(a+1, a+tot+1, cmp); //把 a 数组按边权从小到大的顺序排序

for (int i=1; i <= tot; ++i)
{
    if (find(a[i].x) != find(a[i].y)) //如果这条边还可以选(两点还未连通)
    {
        unionn(a[i].x, a[i].y);
        ans += a[i].c; //把这条边给选了(因为边权已排好序,这么做一定是最优的)
    }
}

printf("%d ", ans);
return 0;
}

```

【参考程序 2】

```

#include <iostream> //kruskal 算法
#include <algorithm>
using namespace std;
const int N = 2e3 + 1, M = 1e4 + 1;
struct point
{
    int x, y, w;
} a[M];
int f[N], n, m, k, ans;
inline int get()
{
    char ch;
    while ((ch = getchar()) < '0' || ch > '9');
    int res = ch - '0';
    while ((ch = getchar()) >= '0' && ch <= '9') res = res * 10 + ch - '0';
    return res;
}
inline int find(int x)
{
    if (f[x] != x) f[x] = find(f[x]);
}

```

```

    return f[x];
}
inline void add(const int &x, const int &y, const int &w)
{
    if (x != y)
    {
        f[y] = x;
        k++;
    }
    ans += w;           //加上该通信渠道的费用
    return;
}
inline int cmp(const point &a, const point &b)
{
    if (a.w <= b.w) return 1;
    return 0;
}
int main()
{
    int p, x, y, w, t = 0, fx, fy;
    n = get(), m = get();
    for (int i = 1; i <= n; ++i) f[i] = i;
    for (int i = 1; i <= m; ++i)
    {
        p = get(), x = get(), y = get(), w = get();
        if (p == 1) add(find(x), find(y), w); //提前加上必选通信渠道
        else
        {
            t++;
            a[t].x = x;
            a[t].y = y;
            a[t].w = w;
        }
    }
    sort(a + 1, a + t + 1, cmp);           //将网络按通畅程度从小到大排序
    for (int i = 1; i <= t; ++i)
    {
        fx = find(a[i].x), fy = find(a[i].y);
        if (fx != fy)
        {
            add(fx, fy, a[i].w);           //选择性通信渠道需要判断 x,y 是否还未能通信
            if (k == n-1) break;
        }
    }
    printf("%d\n", ans);
    return 0;
}

```

4. 连接格点(grid)

有一个 M 行 N 列的点阵,相邻两点可以相连。一条纵向的连线花费一个单位,一条横向的连线花费两个单位。某些点之间已经有连线了,试问至少还需要花费多少个单位才能使所有的点全部连通。

输入:

第 1 行输入两个正整数 m 和 n。

以下若干行每行四个正整数 x_1, y_1, x_2, y_2 , 表示第 x_1 行第 y_1 列的点和第 x_2 行第 y_2 列的点已经有连线。输入保证 $|x_1 - x_2| + |y_1 - y_2| = 1$ 。

输出:

输出使得连通所有点还需要的最小花费。

样例输入:

2 2
1 1 2 1

样例输出:

3

【数据规模】

30%数据: $n * m \leq 1000$;

100%数据: $m, n \leq 1000$ 。

【参考程序 1】

```
#include <cstdio> //Kruskal 算法
#include <cstring>
#include <cstdlib>
#include <iostream>
#include <algorithm>
using namespace std;
const int Maxn=1000010;
int n, m, ans, k;
//这里 n 表示行数, m 表示列数, 与题目中的不同; ans 表示答案, k 表示已经加入最小生成树的边数
int father[Maxn]; //father[i] 记录 i 在并查集中的父亲

int num(int x, int y) //将二维的位置 x, y 转化成一维的数字, 方便操作
{
    return m * (x-1) + y;
}

int find(int x) //查找 x 所在集合的根节点
{
    if (father[x] != x) father[x] = find(father[x]);
    return father[x];
}

void unionn(int x, int y) //合并 x, y 所在的集合
{
    int a = find(x);
    int b = find(y);
    if (a != b)
```

```

{
    ++k;                //若合并成功,意味着加入最小生成树的边多了一条
    father[a]=b;
}
}

int main()
{
    scanf("%d%d", &n, &m);
    for (int i=1; i <= n * m; ++i)
        father[i]=i;    //初始化 father

    int x1, y1, x2, y2;
    while (scanf("%d%d%d%d", &x1, &y1, &x2, &y2) != EOF) //有若干条边
        unionn(num(x1, y1), num(x2, y2)); //先连接已有的边

    int t1, t2;
    for (int i=1; i < n; ++i)
        for (int j=1; j <= m; ++j)
        {
            //因为权应是从小到大选,纵向费用小于横向,所以先连纵向,再连横向
            t1=num(i, j);        //t1 表示点(i,j)的一维编号
            t2=num(i+1, j);      //t2 表示下一行与其相邻的点的编号
            if (find(t1) != find(t2)) //如果两个点还未连通
            {
                unionn(t1, t2);
                ++ans;            //那么把两点连接
            }
        }
    if (k==n * m-1) break;
    //n * m 个结点的最小生成树共有 n * m-1 条边,则已经连完后就可以直接 break
}

for (int i=1; i <= n; ++i)
    for (int j=1; j < m; ++j)
    {
        t1=num(i, j);        //t1 表示点(i,j)的一维编号
        t2=num(i, j+1);      //t2 表示下一列与其相邻的点的编号
        if (find(t1) != find(t2)) //如果两个点还未连通
        {
            unionn(t1, t2);
            ans+=2;            //那么把两点连接
        }
        if (k==n * m-1) break;
        //n * m 个结点的最小生成树共有 n * m-1 条边,则已经连完后就可以直接 break
    }
    printf("%d", ans);
    return 0;
}

```

【参考程序 2】

```

#include <iostream>                                //kruskal 算法
#include <algorithm>
using namespace std;
const int N = 1e6 + 1, M = 1998e3 + 1;
int f[N], m, n, k, ans;                            //与题目不同,n 为行数,m 为列数
inline int get()
{
    char ch;
    while ((ch = getchar()) < '0' || ch > '9');
    int res = ch - '0';
    while ((ch = getchar()) >= '0' && ch <= '9') res = res * 10 + ch - '0';
    return res;
}
inline int num(const int &x, const int &y)          //利用坐标给点标号,以便操作
{
    return m * (x - 1) + y;
}
inline int find(int x)
{
    if (f[x] != x) f[x] = find(f[x]);
    return f[x];
}
inline void add(const int &x, const int &y, const int &w)
{
    f[y] = x;
    k++;
    ans += w;                                       //加上该连线的费用
    return;
}
int main()
{
    int x1, y1, x2, y2, fx, fy;
    n = get(), m = get();
    for (int i = 1; i <= n * m; ++i) f[i] = i;
    while (scanf("%d %d %d %d", &x1, &y1, &x2, &y2) != EOF) add(find(num
(x1, y1)), find(num(x2, y2)), 0);                //加上原有的连线,费用为 0
    for (int j = 1; j <= m; ++j)                  //纵向连线费用比横向小,先连纵向
    {
        fy = find(num(1, j));
        for (int i = 2; i <= n; ++i)
        {
            fx = fy;
            fy = find(num(i, j));
            if (fx != fy) add(fx, fy, 1);
            if (k == n * m - 1) //如果此时已经将所有的点全部连通,就直接输出,结束程序
            {

```

```

        printf("%d\n", ans);
        return 0;
    }
}

for (int i = 1; i <= n; ++i)           //再连横向
{
    fy = find(num(i, 1));
    for (int j = 2; j <= m; ++j)
    {
        fx = fy;
        fy = find(num(i, j));
        if (fx != fy) add(fx, fy, 2);
        if (k == n * m - 1) break;
    }
}

printf("%d\n", ans);
return 0;
}

```

第六节 拓扑排序与关键路径

1. 烦人的幻灯片(slides)

李教授将于今天下午作一次非常重要的演讲。不幸的是他不是一个非常爱整洁的人,他把自己演讲要用的幻灯片随便堆在了一起。因此,演讲之前他不得不去整理这些幻灯片。作为一个讲求效率的学者,他希望尽可能简单地完成它。教授这次演讲一共要用 n ($n \leq 26$) 张幻灯片,这 n 张幻灯片按照演讲要使用的顺序已经用数字 $1 \sim n$ 编了号。因为幻灯片是透明的,所以我们不能一下子看清每一个数字所对应的幻灯片。

现在我们用大写字母 A, B, C, ... 再次把幻灯片依次编号。你的任务是编写一个程序,把幻灯片的数字编号和字母编号对应起来,显然这种对应应该是唯一的;若出现多种对应的情况或是某些数字编号和字母编号对应不起来,我们称对应是无法实现的。

输入:

文件的第一行只有一个整数 n , 表示有 n 张幻灯片, 接下来的 n 行每行包括 4 个整数 $xmin, xmax, ymin, ymax$ (整数之间用空格分开) 为幻灯片的坐标, 这 n 张幻灯片按其在文件中出现的顺序从前到后依次编号为 A, B, C, ..., 再接下来的 n 行依次为 n 个数字编号的坐标 x, y , 显然在幻灯片之外是不会有数字的。

输出:

若是对应可以实现, 输出文件应该包括 n 行, 每一行为一个字母和一个数字, 中间以一个空格隔开, 并且每行以字母的升序排列, 注意输出的字母要大写并且定格; 反之, 若是对应无法实现, 在文件的第一行顶格输出 None 即可。首行末无多余的空格。

样例输入:

```
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
```

样例输出:

```
A 4
B 1
C 2
D 3
```

【参考程序】

```
#include<cstring>
#include<cmath>
#include<iostream>
#include<cstdio>
#include<cstdlib>
using namespace std;
long a[5001][5001],ans[5001],f[5001];
long xmin[5001],xmax[5001],ymin[5001],ymax[5001];
bool b[501];
char s[5001];
int main()
{
    long n,m,i,j,k,x,y,l;
    bool bo,boo;
    memset(b,true,sizeof(b));
    memset(a,0,sizeof(a));
    memset(f,0,sizeof(f));           //初始化
    cin>>n;
    for (i=1; i<=n; i++){
        cin>>xmin[i]>>xmax[i]>>ymin[i]>>ymax[i];
    }
    for (i=1; i<=n; i++){
        cin>>x>>y;
        for (j=1; j<=n; j++)
            if((x>=xmin[j])&&(x<=xmax[j])&&(y>=ymin[j])&&(y<=ymax[j]))
            {
                a[i][0]++; a[i][a[i][0]]=j; //将能配对的数字和幻灯片之间连一条边
                f[j]++; }                //统计幻灯片的入度
    }
    boo=true;
    for (k=1; k<=n; k++)
    {
        bo=false;
        for (i=1; i<=n; i++)
            if (b[i]==true){                //当前数字还没被匹配
```



```

for (j=1; j<=a[i][0]; j++)
    if (f[a[i][j]]==1)           //找到能唯一匹配的幻灯片
    {
        for (l=1; l<=a[i][0]; l++)
            f[a[i][l]]--;         //把当前数字能配对的其他幻灯片的入度-1
        bo=true; ans[a[i][j]]=i; b[i]=false; //标记,统计答案
        break;
    }
    if (bo==true) break;          //已经找到一个配对
    if (i==n) boo=false;         //找不到这个数字的配对,标记为无法实现
}
if(boo==false) break;           //无法实现,不用继续找了
}
for (i=1; i<=n; i++) s[i]=i+64;
if (boo==false) cout<<"None "; //为 false 即无法对应
else for(i=1; i<=n; i++) cout<<s[i]<<" "<<ans[i]<<endl;
}

```

2. 病毒(virus)

有一天,小 y 突然发现自己的计算机感染了一种病毒! 还好,小 y 发现这种病毒很弱,只是会把文档中的所有字母替换成其他字母,但并不改变顺序,也不会增加和删除字母。

现在怎么恢复原来的文档呢? 小 y 很聪明,他在其他没有感染病毒的机器上,生成了一个由若干单词构成的字典,字典中的单词是按照字母顺序排列的,他把这个文件拷贝到自己的机器里,故意让它感染上病毒,他想利用这个字典文件原来的有序性,找到病毒替换字母的规律,再用来恢复其他文档。

现在你的任务是:告诉你被病毒感染的字典,要你恢复一个字母串。

输入:

第 1 行为整数 $K(\leq 50,000)$,表示字典中的单词个数。

以下 K 行,是被病毒感染了的字典,每行一个单词。

最后一行是需要你恢复的一串字母。

所有字母均为小写。

输出:

输出仅一行,为恢复后的一串字母。当然也有可能出现字典不完整、甚至字典是错的情况,这时请输出一个 0。

样例输入:

6

cebdbac

cac
ecd
dca
aba
bac
cedab

样例输出:

abcde

【参考程序】

```
#include <iostream>
using namespace std;
#include <cstdio>
#include <cstdlib>
#include <cmath>
int qm[27],sd[27],dy[27];           //sd[i]表示第 i 个字母恢复后的字母
string s[50001];
int next[1000],back[1000],last[1000];
bool yt[27][27];
int tot,cmax;
bool cx[27];
void add_edge(int a,int b)          //连接 a,b
{
    tot++;
    next[tot]=b;
    back[tot]=last[a];
    last[a]=tot;
}
void dfs(int x,int t)
{
    if (t>sd[x]) sd[x]=t;
    int i=last[x];
    while (i!=0)
    {
        int v=next[i];
        if (sd[v]<=t) dfs(v,t+1);
    }
}
```

```

    i=back[i];
}
}
void error()
{
    cout<<" 0 ";
    exit (0);
}
int main()
{
    int k;
    cin>>k;
    for (int i=0; i<=k; i++)
    {
        getline(cin,s[i]);
    }
    for (int i=2; i<=k; i++)           //对相邻两个被感染的单词进行比较
    {
        int l1=s[i-1].size();
        int l2=s[i].size();
        for (int j=0; j<=min(l1,l2)-1; j++)
        {
            int t1=s[i-1][j]-96;
            int t2=s[i][j]-96;
            cmax=max(t1,max(t2,cmax));
            if (t1==t2) continue;
            if (yt[t2][t1])
                error();           //t1 原来在 t2 前,现在又要求满足 t2 在 t1 前,显然字典有错
            qm[t2]++;               //统计入度
            yt[t1][t2]=true;        //因为单词是按照字典序排的,所以当 t1!=
            //t2 时,t1 原来的字母要排在 t2 的原来字母前
            add_edge(t1,t2);        //连边
            break;
        }
    }
}

```

```

for (int i=1; i<=cmax; i++) //利用拓扑排序求出每个字母恢复后的字母
{
    if (qm[i]==0) dfs(i,1);
}
string que,ans="";
getline(cin,que);
for (int i=0; i<=que.size()-1; i++)
{
    if (cx[sd[que[i]-96]]) error(); //一个字母两种恢复,字典出错
    cx[sd[que[i]-96]]=true;
    if (sd[que[i]-96]==0) error(); //一个字母无法恢复,字典不完整
    ans=ans+(char (sd[que[i]-96]+96));
}
cout<<ans;
}

```