信 奥 编 程
PROGRAMMING

信奥编程讲义

（基础算法）

陈 琰 宏 编著

微信：13989476922

邮箱：zufe_graphics@163.com

目　　录

第零讲 函数的应用*

“函数”这个名词的英文原文是“function”，而“function”的原意是“功能”。顾名思义，函数就是用来实现某个功能的，而且通常只实现一个功能(而不会把多个功能糅合到一个函数里)。也就是说，在程序设计语言里引入函数的概念，就是为了进行功能分解。

例如：要输出 100~200 之内的素数，可以用一个二重循环实现。但如果有一个函数 prime，能够实现判断一个整数是否为素数。其调用形式是：prime(m)。该函数调用返回值如果为 1，则 m 为素数，如果为 0，则为合数。因此我们只需要用如下的代码就可以输出 100~200 之内的素数：

```
for( int m=100; m<=200; m++ )
{
    if( prime(m) )
        printf( "%d\n", m );
}
```

把“输出 100~200 之内所有素数”的功能需求进行分解，把“判断一个整数是否为素数”的功能用 prime 函数去实现，这就是函数的功能所在。

- **参数**是函数与函数之间实现通信的数据“接口”。函数调用的过程就是调用者带着实际参数（如果有）执行函数，将实际参数“传递”给形式参数，执行完函数体后再将计算得到的返回值传递给调用者（如果有）。

- 函数参数的**传递方式**，或者说函数参数的调用方式分为三种：**传值（调用），传址（调用）和 引用（调用）**。传值（调用）在上一讲已经讲过，传址（调用）传的是地址，可以通过数组和指针实现。

- **函数的返回值**是通过函数中的 return 语句获得。return 语句将被调用函数中的一个确定值带回主调函数中去。一个函数中可以有一个以上的 return 语句，执行到哪个 return 语句，哪个语句起作用。执行到 return 语句，函数就执行完毕了，之后的语句就不执行了。函数值应属于某个确定的类型。如果函数值的类型和 return 语句中表达式的值不一致，则以函数类型为准，即函数类型决定返回值的类型。

- 在一个函数内部定义的变量被称为**局部变量**，也叫做内部变量。形式参数也是一种局部变量。局部变量只在定义它的函数范围内有效，即在此函数之外无法使用这些变量。**全局变量**又称为外部变量，它是在函数之外定义的变量，其作用范围为从定义的位置开始到本源程序文件的结束。

1 [2020]：哥德巴赫猜想

哥德巴赫猜想：对于任何一个大于或等于 4 的偶数 n，总是可以分解成一个素数对之和。即存在至少一对素数 P1 和 P2，使得 n＝P1+P2。

编程分析：按照模块化编程思想，定义一个 f(int x) 函数实现判断素数的功能，f(int x) 函数的形参为素数则返回 1，否则返回 0。

```
##include<stdio.h>
#include<math.h>
int f(int x)
```

```
{
  int y,i;
  int k=sqrt(x);
  for(i=2;i<=k;i++)
  {
    if(x%i==0) break;
  }
  if(i>k) y=1;
  else y=0;
  return y;
}
int main()
{
  int n;
  while(scanf("%d",&n)!=EOF&&n!=0)
  {
    if(n==4) printf("1\n");
    int j,sum=0;
    for(j=3;j<=n/2;j=j+2)
    {
      if(f(j)==1&&f((n-j))==1) sum++;
    }
    printf("%d\n",sum);
  }
  return 0;
}
```

2 [3340] 哥德巴赫猜想 2

题目描述

1742 年 6 月 7 日哥德巴赫写信给当时的大数学家欧拉，正式提出了以下的猜想：任何一个大于 9 的奇数都可以表示成 3 个质数之和。质数是指除了 1 和本身之外没有其他约数的数，如 2 和 11 都是质数，而 6 不是质数，因为 6 除了约数 1 和 6 之外还有约数 2 和 3。需要特别说明的是 1 不是质数。

这就是哥德巴赫猜想。欧拉在回信中说，他相信这个猜想是正确的，但他不能证明。

从此，这道数学难题引起了几乎所有数学家的注意。哥德巴赫猜想由此成为数学皇冠上一颗可望不可及的“明珠”。

题目描述

现在请你编一个程序验证哥德巴赫猜想。

先给出一个奇数 n，要求输出 3 个质数，这 3 个质数之和等于输入的奇数。

```
#include<bits/stdc++.h>
using namespace std;
int a,i,n1,n2,n3,x;
int zs(int x)//自己定义一个判断质数的函数，用起来会比较方便
{
```

```
  for (i=2;i<=sqrt(x);i++)
    if (x%i==0)
      return 1;
  return 0;
}
int main()
{
  cin>>a;//读入
  for (n1=2;n1<=a-4;n1++)//进入循环，寻找合适的数。
    for (n2=2;n2<=a-4;n2++)//为了节省时间，只设置两层循环。
    {
      n3=a-n1-n2;//用前两个数求出第三个数。
      if (zs(n1)+zs(n2)+zs(n3)!=0)//判断三数是否均为质数，否则直接跳过。
        continue;
      cout<<n1<<' '<<n2<<' '<<n3;//输出
      return 0;
       }
}
```

3 [6859] 楼层编号

[问题描述]
小林在 NOIP 比赛期间住在"新世界"酒店。和其他酒店不一样的是，这个酒店每天都有一个高能的数字 t，这个数字在楼层中是不会出现的，以 t=3 为例，则 3、13、31、33 等楼层是不存在的，楼层编号为 1，2，4，5，…所以实际上的 4 楼才是 3 楼。
已知小林预订了编号为 m 层的房间，并且当天高能数字是 t，现在他想知道房间所在的真实楼层是多少。
[输入格式]
一行两个整数 m 和 t，1≤m≤100000，0≤t≤9，保证 m 对 t 合法。
[输出格式]
一行一个整数，表示真实楼层。
[输入样例]
14 3
[输出样例]
12

[问题分析]
根据题意，只要从 1~m 穷举楼层编号，将所有含高能数字 t 的楼层计数存储在 ans 中，最后的答案就是 m-ans。

```
#include<bits/stdc++.h>
using namespace std;
int check(int x,int t)
{
  while(x)
```

```
  {
    if(x%10==t) return 1;
    x/=10;
  }
  return 0;
}
int main()
{
  int m,t;cin>>m>>t;
  int ans=m;
  for(int i=1;i<=m;i++)
    if(check(i,t)) ans--;
  cout<<ans<<endl;
}
```

4 [1034] 校验和

题目描述

校验和是一种算法，这种算法扫描数据包并返回一个值。这种算法的思想是当数据包被改变时，校验和同样要改变，因此校验和通常用来检查传输错误，用来确认传输内容的正确性。

在本题中，你需要实现一种校验和算法，称为 Quicksum。一个 Quicksum 数据包只允许包含大写字母和空格，并且起始字符和终止字符都是大写字母。除此之外，

空格和大写字母允许以任何的组合方式出现，包括连续的空格。

校验和 Quicksum 是数据包中所有字符在数据包中的位置和它的值的乘积的累加和。空格的值为 0，其他大写字符的值为它在字母表中的位置，即 A=1,B=2，...，

Z=26。

```
# include<stdio.h>
int f(char ch){
  int s;
  if(ch==' ') s=0;
  else s=ch-64;
  return s;
}
int main(){
  char str[256];
  while(gets(str) !=NULL){
    int i=0,sum=0,s;
    while(str[i]!='\0'){
      s=f(str[i]);
      i++;
      sum=sum+s*i;
    }
    if(str[0]==35) break;
```

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

5 [6857] 抽奖

公司举办年会，为了活跃气氛，设置了摇奖环节。参加聚会的每位员工都有一张带有号码的抽奖券。现在，主持人依次公布 n 个不同的获奖号码，小谢看着自己抽奖券上的号码 num，无比紧张。请编写一个程序，如果小谢获奖了，请输出他中的是第几个号码；如果没有中奖，请输出 0。

[输入格式]

第一行一个正整数 n，表示有 n 个获奖号码，2<n≤100。

第二行包含 n 个正整数，之间用一个空格隔开，表示依次公布的 n 个获奖号码。

第三行一个正整数 num，表示小谢抽奖券上的号码。

1≤获奖号码，num<10000。

[输出格式]

一行一个整数，如果小谢中奖了，表示中奖的是第几个号码；如果没有中奖，则为 0。

输入

输出

样例输入

```
7
17 2 3 4 9555 6 1
3
```

样例输出

```
3
```

```
#include<iostream>
using namespace std;
int win,g[101];
int binsearch(int left,int right){
    if(left <= right){
        int mid = (left + right) / 2;
        if(g[mid] == win) return mid;//找到
        if(win < g[mid]) return binsearch(left,mid - 1);//在左半部分
        if(win > g[mid]) return binsearch(mid + 1,right);//在右半部分
    }
    else return 0;//没找到
}
int main(){
    int n,i,f,left,right,mid;
    scanf("%d",&n);
    for(i = 1; i <= n; i++) scanf("%d",&g[i]);
    cin >> win;
```

```
                    f = binsearch(1,n);
                    cout << f <<endl;
    return 0;
}
```

课堂作业列表

2020 哥德巴赫猜想
3340 哥德巴赫猜想 2
6859 楼层编号
1034 校验和
6857 抽奖

函数的应用习题

1. 以下程序运行后的输出结果是_____。

```
int main()
{   int a=3,y=0;
   do
   (      a*=2; y=y+a;
          if(y>50)   break;
   } while(1);
printf("a=%d\n",a);
}
```

2. 以下程序运行结果是_____。

```
int main(   )
{   int a[10]={1,1,2,8,4,2,3,3,8,2};
    int i,j,k,n=0;
    for(i=0; i<10-n; i++)
    {   for(j=i+1; j<10-n; j++)
          if(a[j]==a[i] )
          {   for(k=j; k<10-n; k++)a[k]=a[k+1];
              n++;
          }
    }
    for(i=0;i<10-n;i++)printf("%d ",a[i]);
    printf("\n");
}
```

3. 下面程序的功能是将两个以按从小到大排序的数组合并成一个有序数组。
请填空。

```
main(   )
{      int   a[5]={ 2,7,13,28,76 },i,j,k;
       int   b[5]={ 3,8,11,35,49 },c[10];
       i=0;   j=0;   k=0;
       while( i<5 && j<5 )
           if(____1_____) {   c[k]=a[i];   i++;   k++; }
           else ( c[k]=b[j];   j++;   k++; )
       while( i<5 ) {  _2_____; i++;   k++;   }
       while( j<5 ) {__3_____;   j++;   k++   }
       for( i=0;i<10;i++   )   printf("%4d",c[i]);
```

4. 以下程序判断输入的正整数的各位数字之和是否为质数并打印相应的结果"。请在空格里填写缺失的部分代码。提示：1、填写的代码要确保程序可以正确编译、运行得出结果；2、空格之后的括号里注明了分数，该括号不属于代码部分。

```
#include <stdio.h>
#include <math.h>
int isPrime(int num){   //判断 num 是否为质数
      for(int i=2; ___1_____); i++)
           if( _____2_____) return 0;
      return 1;
}
int sumDigits(int num){   //返回 num 各位数字之和
      int sum;
      for(_____3_____ )
           sum += _____4_____;
      return sum;
}
int main(){
      int num;
      scanf("%d",&num); //输入一个整数，并假设输入的数大于 1
      if(__5_____ == 1 ) printf("各位数字之和是素数！ \n");
      else printf("各位数字之和不是素数！ \n");
      return 0;
}
```

5. 输出 80 到 120 之间的满足给定条件的所有整数。条件为构成该整数的每位数字都相同，要求定义和调用函数 is(n)判断整数 n 的每位数字是否都相同，若相同则返回 1，否则返回 0。请将函数补充完整。 运行示例： 88 99 111

```
#include <stdio.h>
int main(void)
{
```

```
    int i;
    int is(int n);
    for(i = 80; i <= 120; i++)
        if(   is(i)   )
            printf("%d ", i);
    printf("\n");
    return 0;
}
int is(int n)
{
    int old, digit;
    old = n % 10;
    do{
        digit = n % 10;
        if( ____1_________)     return 0;
      _____2________;

        n = n / 10;
    }while(n != 0);
    ________3_____;
}
```

程序设计

[问题描述]
小林在 NOIP 比赛期间住在"新世界"酒店。和其他酒店不一样的是，这个酒店每天都有一个高能的数字 t，这个数字在楼层中是不会出现的，以 t=3 为例，则 3、13、31、33 等楼层是不存在的，楼层编号为 1，2，4，5，…所以实际上的 4 楼才是 3 楼。
已知小林预订了编号为 m 层的房间，并且当天高能数字是 t，现在他想知道房间所在的真实楼层是多少。

第一讲 结构体

重点

1. 理解结构体定义
2. 掌握结构体的使用方式
3. 掌握结构体排序

在存储和处理大批量数据时，一般会使用数组来实现，但是每一个数据的类型及含义必须一样。如果需要把不同类型、不同含义的数据当作一个整体来处理，如 1000 个学生的姓名、性别、年龄、体重、成绩等，怎么处理呢？C++ 提供了结构体（struct）来解决这类问题。

1.1 结构体定义

将不同种类型的数据有序地组合在一起，构造出一个新的数据类型，这种形式称为结构体。结构体是多种类型组合的数据类型。结构体类型只是一种数据类型，不占内存空间，只有定义结构体类型变量时才开辟内存空间。

使用结构体，必须要先声明一个结构体类型，再定义和使用结构体变量。结构体类型的声明格式如下：

```
struct 类型名{
  数据类型 1    成员名 1;
  数据类型 2    成员名 2;
  …
};
```

如：

```
  struct    student
 {   int   num;
     char   name[20];    不同数据类型组成的成员
     char   sex;
     char   addr[30];
 };
```

定义结构体后，再定义变量名：

```
struct    student
{     int   num;
      char   name[20];
      char   sex;
      int   age;
      float   score;
      char   addr[30];
};
struct   student     student1,   student2;
```

也可以在定义类型的同时定义变量

```
struct    student
```

```
{      int   num;
       char   name[20];
       char   sex;
       int   age;
       float   score;
       char   addr[30];
} student1, student2;
```

1.2 变量的引用

对于结构体类型变量的引用要注意以下几点：

1、 不能对结构体变量整体赋值或输出，只能分别对各个成员引用。

```
cin>>student1.num;        student1.num=100;
```

2、同类型的结构体变量之间可以直接赋值。这种赋值等同于各个成员的依次赋值。

3、结构体变量不能直接进行输入输出，它的每一个成员能否直接进行输入输出，取决于其成员的类型，若是基本类型或是字符数组，则可以直接输入输出。

4、结构体变量可以作为函数的参数，函数也可以返回结构体的值。当函数的形参与实参为结构体类型的变量时，这种结合方式属于值调用方式，即属于值传递。（举例说明）

5、结构体类型的变量在内存依照其成员的顺序顺序排列，所占内存空间的大小是其全体成员所占空间的总和。

6、在编译时，仅对变量分配空间，不对类型分配空间。

7、对结构体中各个成员可以单独引用、赋值，其作用与变量等同。

1.3 结构体数组

结构体数组中的每个元素都是一个结构体类型的变量，其中包括该类型的各个成员。数组各元素在内存中连续存放。

一、结构体数组的定义

```
struct    student
   {      int   num;
          char   name[20];
          char   sex;
          int   age;
          float   score;
          char   addr[30];
};
struct   student   stu[30];
```

或者定义结构体时直接定义数组：

```
struct    student
   {      int   num;
          char   name[20];
          char   sex;
          int   age;
```

```
        float   score;
        char    addr[30];
} stu[30];
```

二、结构体数组的初始化

```
struct    student
    {     int   num;
          char   name[20];
          char   sex;
} stu[3]={ {1011, "Li Lin",'M'}, {1012,"Wang Lan",'F'},
                    {1013,"Liu Fang",'F'};
```

例 1 [1472] 现有有 N 个学生的数据记录，每个记录包括学号、姓名、三科成绩。 编写一个函数 input,用来输入一个学生的数据记录。 编写一个函数 print,打印一个学生的数据记录。 在主函数调用这两个函数，读取 N 条记录输入，再按要求输出。 N<100

输入:学生数量 N 占一行 每个学生的学号、姓名、三科成绩占一行，空格分开。

输出:每个学生的学号、姓名、三科成绩占一行，逗号分开。

样例输入

```
2
a100 zhblue 70 80 90
b200 newsclan 90 85 75
```

样例输出

```
a100,zhblue,70,80,90
b200,newsclan,90,85,75
```

```
#include<iostream>
using namespace std;
struct stu{
    char xh[20];//定义学号
    char name[20];//定义姓名
    int yw,sx,yy;//三科成绩
};
int main(){
    struct stu b[200];//定义结构体数组
    int n,i;
    while(scanf("%d",&n)!=EOF)//输出多组数据
    {
        for(i=0;i<n;i++)scanf("%s%s%d%d%d",b[i].name,b[i].xh,&b[i].yw,&b[i].sx,&b[i].yy);
        for(i=0;i<n;i++)printf("%s,%s,%d,%d,%d\n",b[i].name,b[i].xh,b[i].yw,b[i].sx,b[i].yy);
    }
    return 0;
}
```

1.4 结构体排序

结构体排序是指对结构体中的某一个成员的大小关系排序。例如对上述的 student 数组 stu 中的元素学号成员的大小关系从大到小的顺序(升序)排序。compare 函数可编写成：

```
Student Stu[100];
bool cmp2(Student a,Student b)
{
    return a.id>b.id;//按照学号降序排列
    //return a.id<b.id;//按照学号升序排列
}
sort(Stu,Stu+100,cmp2);
```

1. [1473] 各门课的成绩

有 N 个学生，每个学生的数据包括学号、姓名、3 门课的成绩，从键盘输入 N 个学生的数据，要求打印出 3 门课的总平均成绩，以及最高分的学生的数据（包括学号、姓名、3 门课成绩）。
输入:学生数量 N 占一行每个学生的学号、姓名、三科成绩占一行，空格分开。
输出:各门课的平均成绩和最高分的学生的数据（包括学号、姓名、3 门课成绩）
样例输入
2
1 blue 90 80 70
b clan 80 70 60
样例输出
85 75 65
1 blue 90 80 70

```
#include<bits/stdc++.h>
using namespace std;
struct student {
  char xh[20];
  char xm[20];
  int m1,m2,m3;
}stu[100];
int cmp( student s1,student s2){//定义比较函数
  return s1.m1+s1.m2+s1.m3>s2.m1+s2.m2+s2.m3;
}
int main(){
  int n,i,j,s1=0,s2=0,s3=0;
  cin>>n;
  for(i=1;i<=n;i++)
    cin>>stu[i].xh>>stu[i].xm>>stu[i].m1>>stu[i].m2>>stu[i].m3;
```

```
  for(i=1;i<=n;i++)
  {
    s1+=stu[i].m1;
    s2+=stu[i].m2;
    s3+=stu[i].m3;

  }
  s1=s1/n;s2=s2/n;s3=s3/n;    //求平均分
  sort(stu+1,stu+n+1,cmp);//排序函数
  cout<<s1<<" "<<s2<<" "<<s3<<endl;
  cout<<stu[1].xh<<" "<<stu[1].xm<<" "<< stu[1].m1<<" "<<stu[1].m2<<" "<<stu[1].m3<<endl;
  return 0;
}
```

2 [2707] 最高分数的学生姓名

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:第一行输入一个正整数 N（N ≤ 100），表示学生人数。接着输入 N 行，每行格式如下：分数 姓名分数是一个非负整数，且小于等于 100；姓名为一个连续的字符串，中间没有空格，长度不超过 20。数据保证最高分只有一位同学。

输出：获得最高分数同学的姓名。

样例输入

```
5
87 lilei
99 hanmeimei
97 lily
96 lucy
77 jim
```

样例输出

```
Hanmeimei
```

```
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
struct node
{
  int x;
  string name;
}s[111];
bool cmp(node x, node y)
{
  return x.x > y.x;
```

```
}
int main()
{
  int n;
  while (cin>>n)
  {
    for (int i = 0; i < n; i++)
      cin >> s[i].x >> s[i].name;
    sort(s, s + n, cmp);
    cout << s[0].name << endl;
  }
  return 0;
}
```

1.5 案例讲解

1 [1471] 结构体

题目描述
定义一个结构体变量（包括年、月、日）。计算该日在本年中是第几天，注意闰年问题。
输入
年月日
输出
当年第几天
样例输入
2000 12 31
样例输出
366

```
#include<bits/stdc++.h>
#include<stdlib.h>
using namespace std;
 struct date{
  int year,month,day;
};
int lun_nian(int year){
  if(year%4==0&&year%100!=0||year%400==0)
    return 1;
  else
    return 0;
}
int main()
```

```
{
  struct date d1;//2000 12 31
  int sum=0,i;
  int mon[12]={31,28,31,30,31,30,31,31,30,31,30,31};
  cin>>d1.year>>d1.month>>d1.day;
  for(i=0;i<d1.month-1;i++)
    sum=sum+mon[i];
  sum=sum+d1.day;
  if(lun_nian(d1.year))
    sum=sum+1;
  cout<<sum<<endl;
  return 0;
}
```

2 [1952] NO.1

题目描述

所谓 NO.1,就是所有成绩都排在第一的同学，我们假设每个人只有理科，文科，体育这三门课。我们现在假设某门成绩并列第一，并列的人都是这门功课第一名，并且保证数据不会出现 2 个 NO.1 现给定 n 个人的信息，输出第一名的名字。

输入

多组数据，输入文件第一行为一个整数 T，代表测试数据数。 (T<50)

接下来 T 个测试数据。

每个测试数据的的第一行为一个整数 n(n<=100)，接下来有 n 行，每行的格式如下：

名字 理科成绩 文科成绩 体育成绩 (数值越高代表成绩越好).

名字长度不超过 20，3 个成绩的为正整型.

输出

对于每个测试数据，输出 NO.1 的名字，如果不存在第一名，就输出"NO NO.1".

样例输入

```
3
2
lvhao 2 2 2
xiaoshua 1 1 1
2
lvhao 4 4 4
xiaoshua 4 4 3
3
lvhao 3 4 5
xiaoshua 1 3 1
pan 4 1 5
```

样例输出

```
lvhao
lvhao
```

NO NO.1

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

struct Stu {
  char name[20];
  int l,w,t;
}s[100];


int main(){
  int max1,max2,max3;
  int i,j;
  int t,n;
  max1=max2=max3=0;
  cin>>t;
  for(i=1;i<=t;i++){//t 组数据
    max1=max2=max3=0;
    cin>>n;//每组数据的学生人数
    for(j=0;j<n;j++)//输入每个学生的分数和姓名
      cin>>s[j].name>>s[j].l>>s[j].w>>s[j].t;
    for(j=0;j<n;j++){//求解这组数据中的每门课的最大值
      if(s[j].l>max1)max1=s[j].l;
      if(s[j].w>max2)max2=s[j].w;
      if(s[j].t>max3)max3=s[j].t;
    }
    int flag=1;//记录是否存在 NO1 的状态变量
    for(j=0;j<n;j++){
      if(s[j].l==max1&&s[j].w==max2&&s[j].t==max3){
        cout<<s[j].name<<endl;
        flag=0;
        break;
      }
    }
    if(flag==1)cout<<"NO NO.1"<<endl;
  }

  return 0;
}
```

3 [2742]分数线划定

世博会志愿者的选拔工作正在 A 市如火如荼的进行。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。

输入第一行，两个整数 n，m（5 ≤ n ≤ 5000，3 ≤ m ≤ 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。

第二行到第 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 ≤ k ≤ 9999）和该选手的笔试成绩 s（1 ≤ s ≤ 100）。数据保证选手的报名号各不相同。

输出第一行，有两个整数，用一个空格隔开，第一个整数表示面试分数线；第二个整数为进入面试的选手的实际人数。从第二行开始，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

方法 1:

```
#include<iostream>
using namespace std;
struct mc{
  int grade;
  int score;
}a[5500];
int main(){
  int m,n,num,i,j;
  int line;
  cin>>n>>m;
  num=int(m*1.5);
  for(i=1;i<=n;i++){
    cin>>a[i].grade>>a[i].score;
```

```
  }
  for(i=1;i<n-1;i++){
    for(j=1;j<=n-i;j++){
      if(a[j].score<a[j+1].score)
      swap(a[j],a[j+1]);
      else if(a[j].score==a[j+1].score )
      if(a[j].grade>a[j+1].grade)
      swap(a[j],a[j+1]);
    }
  }
  line=a[num].score ;
  num=0;
  for(i=1;i<=n;i++){
    if(a[i].score>=line)
    num++;
  }
  cout<<line<<" "<<num<<endl;
  for(i=1;i<=num;i++){
     cout<<a[i].grade<<" "<<a[i].score<<endl;
  }
return 0;
}
```

方法 2：

```
#include <iostream>
#include <string.h>
#include <stdio.h>
#include <algorithm>
using namespace std;

int n,m;
struct node{
  int num;
  int score;
}a[5050];

bool cmp(node a,node b){
  if(a.score != b.score) return a.score > b.score;
  return a.num < b.num;
}

int main()
{
  while(scanf("%d%d",&n,&m)!=EOF){
```

```
    int men = m * 1.5;
    for(int i=0;i<n;i++) scanf("%d%d", &a[i].num, &a[i].score);
    sort(a, a+n, cmp);
    int score = a[men-1].score;
    for(int i=0;i<n;i++){
      if(a[i].score < score) break;
      men = i+1;
    }
    printf("%d %d\n", a[men-1].score, men);
    for(int i=0;i<n;i++){
      if(a[i].score < score) break;
      printf("%d %d\n", a[i].num, a[i].score);
    }
  }
  return 0;
}
```

课堂作业列表

1472 打印一个学生的数据记录
1473 各门课的成绩
2707 最高分数的学生姓名
1471 结构体
1952 NO.1
2742 分数线划定
3868 阿里巴巴与四十大盗

第二讲 枚举

重点

1. 熟练应用枚举算法解决一些实际问题。
2. 掌握枚举算法的常用优化方法。

枚举又称为穷举，是一种很朴素的解题思想。计算机的特点之一就是运算速度快，善于重复做一件事。"穷举"正是基于这一特点的最古老的算法。它一般是在一时找不出解决问题的更好途径，即从数学上找不到求解的公式或规则时，根据问题中的"约束条件"，将解的所有可能情况一一列举出来，然后再逐个验证是否符合整个问题的求解要求，从而求得问题的可行解或者最优解。

例如，在判断 m=199 是否为素数时，从素数的定义出发，试图找出 2~198 范围内能整除 m 的自然数，如果不能找到，则 m 是素数。根据分析，我们将范围缩小到 2~14。依次判断 2 能否整除 m，判断 3 能否整除 m，…，一直判断到 14 都还没有找到能整除 m 的自然数，因此得出结论：m=199 是素数。这个过程实际上就是一个枚举的过程，枚举 2, 3, 4, 5, … ,14 能否整除 m。

2.1 枚举的基本思路

例 1[3309] 求 $x^2 + y^2 = 2000$ 的正整数解。

分析：x 和 y 都是正整数，因此 x 和 y 的取值范围只能是：1、2、…、44，其中 44 是小于等于 sqrt(2000)的最大正整数。对于在这个范围内的所有(x, y)组合，都去判断一下。也就是枚举所有的(x, y)组合，判断是否满足 x2+y2=2000，如果满足，则是一组解。即当 x 取 1 时，考虑 y 取 1、2、…、44；然后当 x 取 2 时，又考虑 y 取 1、2、…、44；…；最后当 x 取 44 时，又考虑 y 取 1、2、…、44。整个过程如图 4.1 所示。在实现时要用到 2 重循环，从算法思想的角度看，这个过程就是枚举，即枚举所有的(x, y)组合。

```
 #include <stdio.h>
#include <math.h>
int main( )
{
    int x, y;
    int m = sqrt(2000);      //循环变量 x 和 y 的终值
    for( x=1; x<=m; x++ )    //x 从 1 枚举到 m
    {
        for( y=1; y<=m; y++ )   //y 也从 1 枚举到 m
        {
            if( x*x + y*y == 2000 )  //判断相等必须用"=="
                printf( "2000=%d*%d+%d*%d\n", x, x, y, y );
        }
    }
    return 0;
}
```

思考：从运行结果可以看出，(8, 44)这一组解和(44, 8)这一组解实际上只是交换了 x 和 y。如果认为这是同一组解，那么方程就只有两组解：(4, 44)和(20, 40)，该怎样修改程序呢？

```
 #include <stdio.h>
#include <math.h>
```

```
int main( )
{
    int x, y;
    int m = sqrt(2000);      //循环变量 x 和 y 的终值
    for( x=1; x<=m; x++ )    //x 从 1 枚举到 m
    {
        for( y=x; y<=m; y++ )    //y 也从 1 枚举到 m
        {
            if( x*x + y*y == 2000 )  //判断相等必须用"=="
                printf( "2000=%d*%d+%d*%d\n", x, x, y, y );
        }
    }
    return 0;
}
```

2.2 案例讲解

1 [2019] 开关灯

一条线上有 N 盏灯，灯有开有关，1 表示开，0 表示关（ON/OFF），您的任务是确定至少需要切换多少次灯（从 ON 到 OFF，或者从 OFF 到 ON），以使灯交替打开和关闭。

如 1110111 -》 1010101 操作两次

1110111 -》 0101010 操作五次

输入

每个测试数据一行。第一个数为整数 n 表示灯的数量（1＜n＜＝10000），接着是表示灯的状态的 n 个整数（ON 为 1），OFF 为“0”。

输出

对于每个测试数据，输出至少需要操作多少次开关。

样例输入

3 1 1 1

3 1 0 1

样例输出

1

0

有 N 盏灯，排成一排。给定每盏灯的初始状态(开或关)，你的任务是计算至少要切换多少盏灯的状态(将开着的灯关掉，或将关掉的灯开起来)，才能使得这 N 盏灯开和关交替出现。

输入描述：

输入文件中包含多个测试数据，每个测试数据占一行。首先是一个整数 N，1≤N≤10000，然后是 N 个整数，表示这 N 盏灯的初始状态，1 表示开着的，0 表示关着的。测试数据一直到文件尾。

分析：本题可以采取不同的枚举思路求解。

第一种枚举思路：N 盏灯，每盏灯都有两种状态：1 和 0，N 盏灯共有 2N 种状态，从 0 0 0 … 0 到 1 1

1 … 1。可以枚举这 2N 种状态，每种状态都判断一下是否是开和关交替出现，如果是则还要记录从初始状态转换到该状态需要切换的灯的数目。但这种枚举策略势必要花费很多时间，因为 N 最大可以取到 10000，而 210000 的数量级是 103010。

第二种枚举思路：要使得 N 盏灯开和关交替出现，实际上只有两种可能：奇数位置上的灯是开着的，和偶数位置上的灯上是开着的，如图 4.2 所示。只要分别计算从 N 盏灯的初始状态出发，切换到这两种状态所需要切换灯的数目，取较小者即可。

第三种枚举思路：注意到上述分析中 3+6=9，9 就是灯的数目 N。稍加分析就可以得到结论：如果将 N 盏灯调整成奇数位置上的灯是开着的，需要调整灯的数目为 numo，则将这 N 盏灯调整为偶数位置上的灯是开着的，需要调整灯的数目 nume = N - numo。因此只要判断 numo 是否小于 N/2，如果是则取 numo，否则取 N - numo。这种思路只要枚举一种状态即可。

```
#include <stdio.h>
int main( )
{
    int N, i;          //N 表示灯的数目
    int numo;          //调整成奇数位置上的灯是开着的需要调整灯的数目
    int lights[10001];
    while( scanf( "%d", &N )!=EOF )
    {
        numo = 0;
        for( i=1; i<=N; i++ )    scanf( "%d", &lights[i] ); //读入初始状态
         //将 N 盏灯的转换调整为奇数位置上为 1，偶数位置上为 0
        for( i=1; i<=N; i++ )
        {
            if( i%2==1 && lights[i]==0) //奇数位置为 0，需要调整
                numo++;
            if( i%2==0 && lights[i]==1) //偶数位置为 1，需要调整
                numo++;
        }
        printf( "%d\n", numo<=N/2 ? numo : N-numo );//"="不能去掉
    }
    return 0;
}
```

2 [1311] 歌德巴赫猜想

1742 年，德国数学家哥德巴赫(Goldbach)提出了著名的哥德巴赫猜想(Goldbach Conjecture)：任何一个不小于 4 的偶数可以表示为两个素数之和。这个猜想至今都没有完全被证明是正确的。但是，对于一个大于或等于 5 的奇数，有的可以表示成两个素数之和，有的则不能。给定一个大于或等于 5 的奇数，判断是否能分解成两个素数之和。

输入输入文件包含多个测试数据，每个测试数据占一行，为一个正整数 m，m 为奇数，且不小于 5，不大于 32767。测试数据一直到文件尾。

输出对每个测试数据，如果 m 能分解成两个素数之和，输出 yes，否则输出 no。

样例输入

21

75
99
113
样例输出
yes
yes
yes
no

分析：简单题，因为是奇数，而最小的素数是 2，但在本题中，只要判断 m-2 是否是素数就可以。

```
#include<iostream>
using namespace std;
int main()
{
	int m,n;
	while(cin>>m)
	{
		n=0;
		m=m-2;
		for(int i=2;i<m;i++)
		{
			if(m%i==0)
			{
				n++;
			}
		}
		if(n==0)
		{
			cout<<"yes"<<endl;
		}
		else
		cout<<"no"<<endl;
	}
	return 0;
}
```

3 [6860 2975]火柴棒等式

[问题描述]
给出 n 根火柴棒，可以拼出多少个形如"A+B=C"的等式？
等式中的 A、B、C 是用火柴棒拼出的整数（若该数非零，则最高位不能是 0）。用火柴棒拼数字 0~9 的

拼法如图 9.7-1 所示。

需要注意以下几点：

（1） 加号与等号各自需要两根火柴棒。

（2） 如果 A ≠ B，则 A+B=C 与 B+A=C 视为不同的等式（A、B、C 均大于或等于 0）。

（3） n 根火柴棒必须全部用上（n≤24）。

[输入样例]14

[输出样例]2

[样例说明]

两个等式分别为：0+1=1 和 1+0=1。

[问题分析]

首先，预处理每个数字（0~9）需要用几根火柴棒，存储在数组 f 中。然后，穷举 a 和 b，算出它们的和 c，再判断下列约束条件是否成立：f（a）+ f（b）+ f（c）= n-4。现在的问题是：a 和 b 的范围有多大？可以发现尽量用数字 1 拼成的数比较大，分析可知最多不会超过 1111。程序实现时，分别用三个循环语句预处理好所有两位数、三位数、四位数构成所需要的火柴棒数量。

```
#include <iostream>
using namespace std;
int fun(int x)      //用来计算一个数所需要的火柴棍总数
{
    int num=0;   //用来计数变量
    int f[10]={6,2,5,5,4,5,6,3,7,6};    //用一个数组记录 0~9 数字所需的火柴棍数
    while(x/10!=0)       // x 除以 10 不等于 0 的话，说明该数至少有两位
    {
        num+=f[x%10];    //加上该位火柴棍数
        x=x/10;
    }
    num+=f[x];       //加上最高位的火柴棍数
    return num;
}
int main()
{
     int a,b,c,m,sum=0;
     cin>>m;            //火柴棍总个数
     for(a=0;a<=1111;a++)   //开始枚举
     {
         for(b=0;b<=1111;b++)
         {
             c=a+b;
             if(fun(a)+fun(b)+fun(c)==m-4)   //去掉+和=
                 sum++;
```

```
            }
        }
        cout<<sum<<endl;
    return 0;
}
```

4 [2660 6864]金币

国王将金币作为工资，发放给忠诚的骑士。第一天，骑士收到一枚金币；之后两天（第二天和第三天）里，每天收到两枚金币；之后三天（第四、五、六天）里，每天收到三枚金币；之后四天（第七、八、九、十天）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 为正整数）。

请编程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。

[输入格式]

输入包含至少一行，但不多于 1000 。

除最后一行外，输入的每行是一组输入数据，包含一个正整数 n，表示天数。

输入的最后一行为 0，表示输入结束。

[输出格式]

对每个数据输出一行一个整数，表示该数据对应总金币数。

[输入样例]

```
10
6
7
11
15
16
100
10000
1000
21
22
0
```

[输出样例]

```
30
14
18
35
55
61
945
942820
29820
91
```

98

[数据规模]
对于 60% 的数据满足：n≤103 ；
对于 80% 的数据满足：n≤106 ；
对于 100% 的数据满足：n≤1012 。
[问题分析]
每次穷举每天获得的金币数，或者将答案保存在数组中直接输出，可以获得部分分。下面利用数学推导进行优化。

因为：1 2 +2 2 +3 2 +…+k 2 = k ×（k+1）×（2k+1）/ 6，假设前 n 个数为 1，2，2，3，3，3，…，k，k，k，k，如何求 k 呢？根据 1+2+3+…+k = n，即 k 2 +k-2n = 0，把 k 看成未知数，解方程可求出 k =(-1 + sqrt (1+8n))/ 2 = sqrt (2n+0.25)-0.5，另一个负数解不合法舍去。

```
#include<bits/stdc++.h>
using namespace std;
long long int x,n,ans;
int main(){
    scanf("%lld",&x);
    while(x){
        n=(long long )(sqrt(x*2+0.25)-0.5);
        ans=(x-n*(n+1)/2)*(n+1)+n*(n+1)*(2*n+1)/6;
        printf("%lld\n",ans);
        scanf("%lld",&x);
    }

    return 0;
}
```

方法二：找规律，第一个为第一组，后两个为第二组，在后两个为第三组……以此类推。思路就是找到 n 在第几组里，如果 n 在这一组的末尾，则直接求平方和；若在这一组的中间，则还需求出它在这一组的第几个，将数据分为从第一组到前一组和这一组来算。判断它在第几组的方法是：发现每一组的最后一个数是所有组数的和，因此将组数升序加起来，一旦当天数 n 刚不大于组数，就说明 n 在该组里。

代码：

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n,a=0,b,sum=0,result=0;
    scanf("%d",&n);//输入天数
    for(int i=1;;i++){ //分为两类讨论
        sum+=i;//循环，直到从第一组开始的每组数据的个数加起来不小于天数
        if(sum>n){//讨论数据个数大于天数的情况，此时 n 在第 i 组里
            for(int j=1;j<=i-1;j++)
```

```
            a+=j;//前（i-1）组求和得最后一天的天数
        b=n-a;//b 为超出（i-1）组的，在第 i 组里的天数
        for(int k=1;k<=i-1;k++)
            result+=pow(k,2);
        result+=b*i;//两部分加起来得结果
        break;
    }
    else if(sum==n){//讨论数据个数小于天数的情况
        for(int k=1;k<=i;k++){
            result+=pow(k,2);
        }//天数正好在第 i 组的末尾，直接求
        break;
    }

  }
  printf("%d",result);//输出结果
  return 0;
}
```

方法三：

```
#include<iostream>
using namespace std;
int main()
{
  int n;//1 2 2 3 3 3 4 4 4 4
  cin >> n; //1+2*2+3*3+4*4+....
  int res = 0;
  int cmp = 1;
  while (n >= cmp)
  {
    n -= cmp;
    res += cmp*cmp;
    cmp++;
  }
  res += n*cmp;
  cout << res << endl;
  return 0;
}
```

方法四：

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

```
{
  int K,N,coin=0;
  scanf("%d",&K);
  for(N=1;K-N>=0;K-=N++)
    coin+=N*N;
  printf("%d\n",coin+K*N);
  return 0;
}
```

5 [3344]比例简化

[问题描述]

在社交媒体上，经常会看到针对某一个观点同意与否的民意调查以及结果。例如，对某观点表示支持的有 1498 人，反对的有 902 人，那么其比例可以简单地记为 1498：902。

因该比例的数值太大，难以一眼看出它们的关系。若把比例记为 5：3，虽然与真实结果有一定的误差，但依然能够较为准确地反映调查结果，同时也显得比较直观。

现给出支持人数 A 和反对人数 B，以及一个上限 L，请将 A 比 B 化简为 A′ 比 B′，要求在 A′ 和 B′ 均不大于 L，且 A′ 和 B′ 互质（两个整数的最大公约数为 1）的前提下，A′/B′ ≥ A/B 且 A′/B′-A/B 的值尽可能小。

[输入格式]

一行三个整数 A，B，L，每两个整数之间用一个空格隔开，分别表示支持人数、反对人数以及上限。

[输出格式]

一行两个整数 A′ 和 B′，中间用一个空格隔开，表示化简后的比例。

[输入样例]1498 902 10

[输出样例]5 3

[数据规模]

对于 100% 的数据满足：1≤A≤10 6 ，1≤B≤10 6 ，1≤L≤100，A/B≤L。

[问题分析]

首先，答案为一个分数，而且分子，分母都小于或等于 L。所以，可以直接穷举分子 i （对应题目中的 A′）和分母 j （对应题目中的 B′）。结合题目的具体要求分析：

（1） i，j≤L，这个可以作为穷举的范围。

（2） i/j≥A/B，且 i/j-A/B 的值尽量小。前者只要转换成 i*B≥j*A，后者可以"打擂台"实现。

假设用 K1/K2 表示最后的答案，初值设置为 1000000/1，min=abs （k1*B-k2*A）。如果 abs （i/j-A/B）< abs（K1/K2-A/B），即如果 abs（i*B-j*A）< abs（k1*B-k2*A），则更新 K1、K2 和 min。

（3） 答案的分子、分母要互质，这个只要从小到大穷举 i、从大到小穷举 j，第一个符合条件的答案肯定就是最简分数。假设 L=10，如果先穷举到 1/5 得到一个答案，后面的 2/10 是不会更新答案的。

```
#include<bits/stdc++.h>
using namespace std;
int gcd(int x,int y)
{
  if(y==0) return x;
  return gcd(y,x%y);
}
```

```
int main ()
{
  int i,j,a,b,ansa,ansb,l;
  cin>>a>>b>>l;
  ansa=l;ansb=1;
  for(i=1;i<=l;i++)
    for(j=1;j<=l;j++)
      if(gcd(i,j)==1&&i*b>=j*a&&i*ansb<j*ansa)
      {
        ansa=i;
        ansb=j;
      }
  printf("%d %d\n",ansa,ansb);
  return 0;
}
```

6 [6861] 奶牛碑文

[问题描述]

小伟暑假期间到大草原旅游，在一块石头上发现了一些有趣的碑文。碑文似乎是一个神秘古老的语言，只包括三个大写字母 C、O 和 W。尽管小伟看不懂，但是令他高兴的是，C、O、W 的顺序形式构成了一句他最喜欢的奶牛单词"COW"。现在，他想知道有多少次 COW 出现在文本中。

如果 COW 内穿插了其他字符，只要 COW 字符出现在正确的顺序，小伟也不介意。甚至，他也不介意出现不同的 COW 共享一些字母。例如，CWOW 出现了 1 次 COW，CCOW 算出现了 2 次 COW，CCOOWW 算出现了 8 次 COW。

[输入格式]

第 1 行为 1 个整数 N。

第 2 行为 N 个字符的字符串，每个字符是一个 C、O 或 W。

[输出格式]

输出 COW 作为输入字符串的字串出现的次数（不一定是连续的）。

提示：答案会很大，建议用 64 位整数（long long）。

[输入样例]

6

COOWWW

[输出样例]

6

[数据规模]

对于 50% 的数据满足：N≤60。

对于 100% 的数据满足：N≤105 。

[问题分析]

因为只有 3 个字母，所以可以穷举字符串中的每一个"O"，假设位置 i，然后分别计算其左边"C" 的个数 l[i] 和右边"W" 的个数 r[i]，再利用乘法原理进行计数 l[i]*r[i]，每次把答案累加到 ans 中。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=100000;
char t[maxn];
long long dp[4];
int main()
{
  int n;
  scanf("%d%s",&n,t+1);
  for(int i=1;i<=n;i++)
  {
    if(t[i]=='C') dp[0]++;
    if(t[i]=='O') dp[1]+=dp[0];
    if(t[i]=='W') dp[2]+=dp[1];
  }
  cout<<dp[2]<<endl;
}
```

2.3 枚举算法的优化

枚举算法的特点是算法设计、实现都相对简单，但时间复杂度和空间复杂度往往较大。因此，用穷举算法解决问题时，往往需要尽量优化算法，从而减少穷举的次数，提高穷举的效率。穷举算法优化的思路主要有结合约束条件，通过数学推导，减少穷举的范围和数量；通过预处理（如部分和、是否素数等），以空间换时间，避免在穷举过程中重复计算或判断等。

1 [6862] 三角形个数

输入一根木棒的长度 n，1≤n≤10000，将该木棒分成三段，每段的长度为正整数，输出由该三段小木棒组成的不一样的三角形个数。

[输入样例]10

[输出样例]2

[样例说明]

两个能组成的三角形边长分别为 2、4、4 和 3、3、4。

[问题分析]

穷举三角形三条边长（假设为 a、b、c）的可能值，判断能否构成一个三角形，若能则计数，最后输出计数器的值。为了保证组成的三角形不重复，只要在穷举时设定 1≤a≤b≤c≤n-2。优化思想很简单但很重要，"能算不举"，穷举两条边，根据木棒长度直接计算出第三条边长。

写法 1：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int n;
    long long total=0;
    cin>>n;
    for(int a=1;a<=n-2;a++)
    {
        for(int b=a;b<=n-2;b++)
        {
            int c=n-a-n;
            if((c>=b)&&(a+b>c))total++;
        }
    }
    cout<<total<<endl;
}
```

写法 2：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
  int n;long long ans=0;
  cin>>n;
  for(int i=1;i<=n/3;i++)
  {
   for(int j=i;j<=n/2;j++)
    {
        int k=n-i-j;
   if(k<j) break;
   if(i+j>k) ans++;
   }
  }
 cout<<ans<<endl;
}
```

2 [6863] 阿姆斯特朗数

编程找出所有的三位数到七位数中的阿姆斯特朗数。阿姆斯特朗数也叫水仙花数，它的定义如下：若一个 n 位自然数的各位数字的 n 次方之和等于它本身，则称这个自然数为阿姆斯特朗数。例如，153（153=1×1×1+3×3×3 +5×5×5）是一个三位的阿姆斯特朗数，8208 则是一个四位的阿姆斯特朗数。

[问题分析]

由于阿姆斯特朗数是没有规律的，只能采用穷举法一一验证 100~9999999 内的所有数是否是阿姆斯特朗数，若是，则打印之。但是，如果对任意一个数 K，都去求它的各位的若干次方，再求和判断是否等于 K，效率比较差。注意到，每个位只可能是 0~9，而且只会算到 3~7 次方。所以，为了使得程序尽快运行出正确结果，采用"以空间换时间"的策略，使用一个数组 p 预处理出所有数字的各次幂之值，p[i,j]表示 i 的 j 次方。另外，为了避免每次都对一个数进行逐位分解操作，直接用数组 a[8]存储一个数的每一位，穷举 100~9999999。

```
#include<bits/stdc++.h>
using namespace std;
int a[9], p[9][7];
int main(){
    for (int i=0;i<=8;i++)a[i]=0;
    for(int i=0;i<=9;i++)p[i][1]=i;
    for(int j=2;j<=7;j++)
        for(int i=0;i<=9;i++)
            p[i][j]=p[i][j-1]*i;

        int len=3,count=0;
        a[3]=1;
        while(a[8]==0){
            int total=0;
            for(int i=1;i<=len;i++)
                total=total+p[a[i]][len];
            int num=0;
            for(int i=len;i>=1;i--)num=num*10+a[i];
            if(total==num){
                count++;
                printf("Ans.%d: %d\n",count,num);
            }
            int i=1;
            while(a[i]==9)i++;
            a[i]++;
            for(int j=1;j<=i-1;j++)a[j]=0;
            if(i==len+1)len++;
        }
    return 0;
}
```

课堂作业列表

2019 切换状态(Switch)
1311 歌德巴赫猜想
6860 2975 火柴棒等式

2660 6864 金币
3344 比例简化
6861 奶牛碑文

枚举习题

1.输入： 5 98 90 95 88 85
输出_____

```
#include<stdio.h>
int main()
{
    int x,i,n,minn=100,maxn=0,sum;
    scanf("%d",&n);
    sum=0;
    for(i=1;i<=n;i++){
        scanf("%d",&x);
        sum=sum+x;
        if(x>maxn)
        maxn=x;
        if(x<minn)
        minn=x;
    }
printf("%.2lf",1.0*(sum-minn-maxn)/(n-2);
    return 0;
}
```

2. 输出_____

```
#include<stdio.h>
int main()
{
    int k=2,n=0;
    double Sn=0;
    while(Sn<=k){
        n=n+1;
        Sn=Sn+1.0/n;
    }
    printf("%d",n);
    return 0;
}
```

3. 题目描述

假设有 N 盏灯(N 为不大于 5000 的正整数)，从 1 到 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 到 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。

10 10

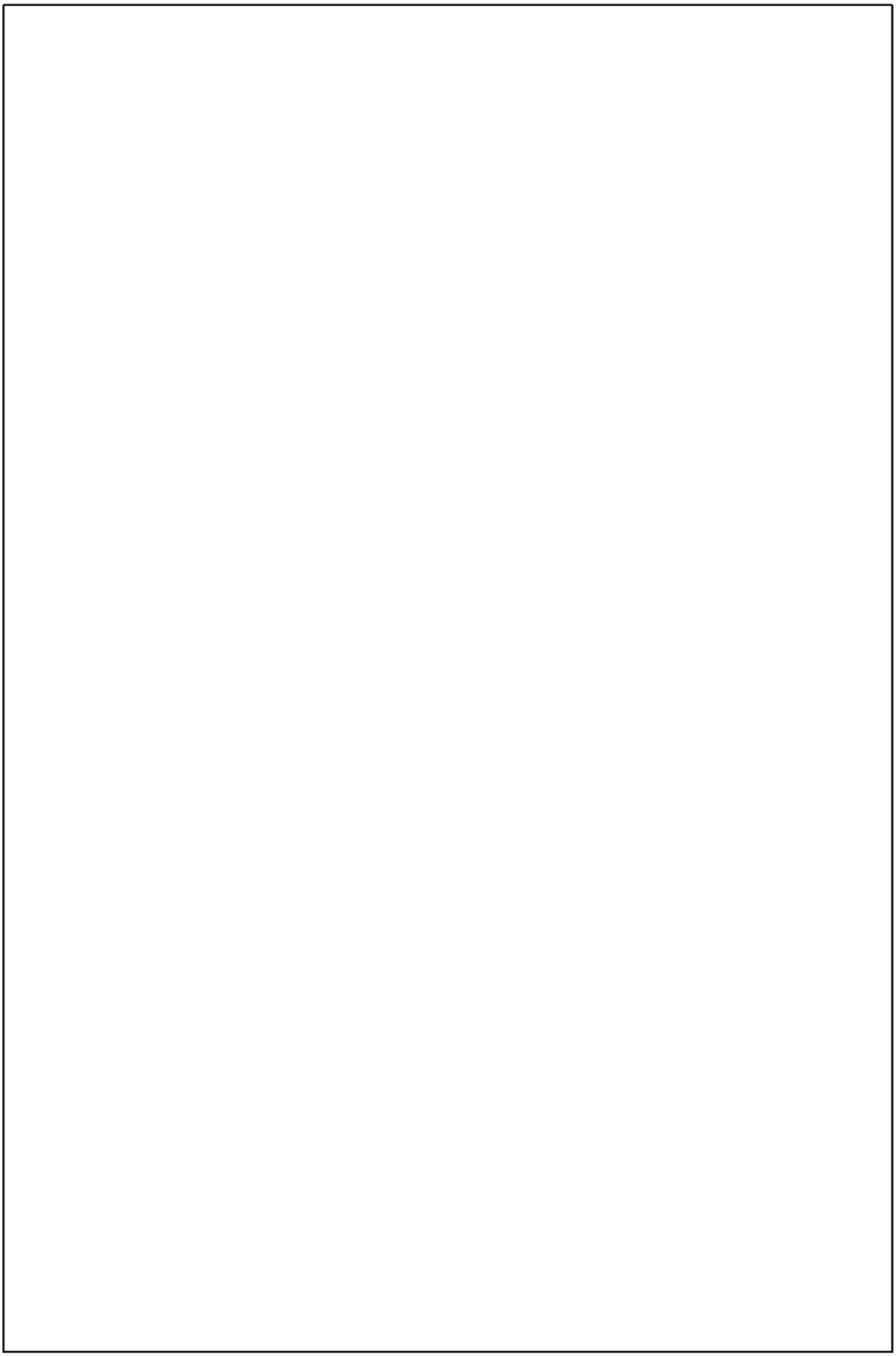
样例输出

1,4,9

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
  bool a[5001]=___1_____;
  int n, m;
  cin>>n>>m;
  for(int i=1;i<=m;i++)
    for(int j=_2_____)
      a[j]=!a[j];
  for(int i=1,j=0;i<=n;i++)
    if(a[i]==true)
    {
      j++;
      If_3_____cout<<i;
      else cout<<','<<i;
    }
  return 0;
}
```

程序设计

有 N 盏灯，排成一排。给定每盏灯的初始状态(开或关)，你的任务是计算至少要切换多少盏灯的状态(将开着的灯关掉，或将关掉的灯开起来)，才能使得这 N 盏灯开和关交替出现。



第三讲 模拟

重点

1. 熟练应用模拟法解决一些实际问题。
2. 体验模拟法的审题分析和细节测试。

现实中有些问题难以找到公式或规律来求解，只能按照一定的步骤不停的"模拟"下去，最后才能得到答案。这样的问题，用计算机来求解十分合适，只要能让计算机模拟人在解决此问题时的行为即可。这种求解问题的思想，可以称为"模拟"。

3.1 模拟的基本思路

例 1 **[2948]醉酒的狱卒**(The Drunk Jailer)

某个监狱有一排、共 n 间牢房，一间挨一间。每间牢房关着一名囚犯，每间牢房的门刚开始时都是关着的。有一天晚上，狱卒厌烦了看守工作，决定玩一个游戏。游戏的第 1 轮，他喝了一杯酒，然后沿着监狱，把所有牢房的门挨个挨个打开；游戏的第 2 轮，他又喝了一杯酒，然后沿着监狱，把编号为偶数的牢房的门关着；游戏的第 3 轮，他又喝了一杯酒，然后沿着监狱，对编号为 3 的倍数的牢房，如果牢房的门开着，则关上，否则打开；...，狱卒重复游戏 n 轮。游戏结束后，他喝下最后一杯酒，然后醉倒了。
这时，囚犯才意识到他们牢房的门可能是开着的，而且狱卒醉倒了，所以他们越狱了。
给定牢房的数目，求越狱囚犯的人数。

分析：

在本题中，n 轮游戏过后，哪些牢房的门是开着的，并无规律可循。但这个游戏的规则和过程都很简单：游戏有 n 轮，第 j 轮游戏是将编号为 j 的倍数的牢房状态变反，j = 1, 2, 3, …, n。如图 5.1 所示。这些规则和过程用程序能较容易地实现，所以适合采用"模拟"的思路求解。
具体实现时可以定义一个一维数组，表示每个牢房的状态，初始为 1，表示 Locked。如图 5.1 所示。模拟 n 轮游戏过程：第 j 轮时，改变牢房号为 j 的倍数的牢房的状态，为 0 改为 1，为 1 改为 0。最后统计状态为 0 的牢房数。

```
#include <stdio.h>
int main( )
{
    int kase, n;     //测试数据的个数，及牢房的个数
    int prision[101];   //n 个牢房(编号从 1~n)的状态,1 表示锁着的
    int i, j;           //循环变量
    scanf( "%d", &kase );
    for( i=0; i<kase; i++ )
    {
        scanf( "%d", &n );
        for( j=1; j<=n; j++ )    prision[j] = 1;//初始时各牢房都是锁着的
        for( j=1; j<=n; j++ )      //游戏进行 n 轮
```

```
        {
            int tmp = j;
            while( tmp<=n )
            {
                prision[tmp] = (prision[tmp]==1) ? 0 : 1;
                tmp += j;
            }
        }
        int num = 0;   //游戏结束后,牢房门开着的牢房的数目
        for( j=1; j<=n; j++ )
            if( !prision[j] )    num++;
        printf( "%d\n", num );
    }
    return 0;
}
```

3.2 案例讲解

1 [3204]模拟约瑟夫环

出列游戏：n 个人围成一圈，从第 1 个人开始报数，报数报到 m 的人出列；然后又从下一个人开始从 1 开始报数；重复 n-1 轮游戏，每轮游戏淘汰 1 个人，最后剩下的人就是胜利者。模拟该游戏，输出依次出列的位置及最后的胜利者。

开始报数位置：s=1
游戏人数：n=8
间隔：m=4
游戏规则：从起始位置开始，第 m 个位置上的数依次出列，循环直至只剩下一个数。
有 8 个人参加报数，每间隔 4 个位置淘汰 1 个出列。
有 7 轮，每轮淘汰一个位置出列，在算法中用 r 来代表第几轮；
用一维数组来存储 8 个位置上的 8 个号码，数组长度为 9，为符合人们的习惯，只使用 a[1]~a[8]。
用变量 i 来指向剩余的位置(要跳过已经出列的位置)
用变量 j 来累加间隔位置，达到 4 后，对应位置要出列，然后又从 1 开始计数。因此要对 4 进行取模运算。本来应该是 j=(j+1)%m，但是(j+1)%m 的范围是 0~3，我们希望 j 取到 1~4，所以改成 j=j%m+1。其中 m=4。

方法 1：

```
 #include <iostream>
using namespace std;
int main( ) {
    int n,m;
    cin>>n>>m;
```

```
    //int a[21]={ 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20 };
    for( i=0; i<=n; i++ )       a[i] = i;
    int r = 1;  //第 r 轮
    int i;//i 指向所有的位置(跳过已经出列的位置)
    int j;//j 累加到 n 然后重新累加
    for( i=1, j=1; r<=n-1; i=i%n+1, j=j%m+1 )
    {
        while( a[i]==-1 ) { i = i%n+1; }//跳过已经出列的
        if( j%m==0 )
        {
            printf( "%d ",a[i] );
            a[i] =-1;      j = 0;      r = r+1;
        }
    }
    for( i=0; i<n; i++ )
    {
        if( a[i+1]!=-1 )
        printf( "%d\n", a[i+1]);
    }
     return 0;
}
```

方法 2：

```
#include<iostream>
using namespace std;
const int ym=25;
int n,m;
int a[ym];
int main(){
    scanf("%d %d",&n,&m);
    int number=n,l=1;
    while(number){
        int p=0;
        for(int i=l;p!=m;i++){
            if(i>n) i%=n;
            if(a[i]==-1)
                continue;
            p++;
            if(p==m){
                a[i]=-1;
                number--;
                l=i+1;
                printf("%d ",i);
                break;
```

```
            }
        }
    }
    return 0;
}
```

2 [6869]网络堵塞

你肯定经历过很多人同时使用网络，网络变得很慢很慢。为了彻底解决这个问题，Ulm 大学决定采取突发事件处理方案：在网络负荷高峰期，将公平地、系统地切断某些城市的网络连接。德国的城市被随机地标上 1~n 的序号。比如金华的序号为 1，杭州的序号为 2，温州的序号为 3 等等，这些序号顺序纯粹是随机的。然后随机地选择一个数 m。首先切断第 1 个城市的网络连接，然后间隔 m 个序号，切断对应的城市，如果超出范围，则取模，并且忽略已经被切断网络连接的城市。例如，如果 n=17，m=5，被切断网络连接的城市依次为：[1,6,11,16,5,12,2,9,17,10,4,15,14,3,8,13,7]。
本题的目的是，希望最后被切断网络连接的城市是杭州。对于给定的 n 值, m 的值必须很仔细地选择，使得 2 号城市(即杭州的序号)是最后被选中切断网络连接的城市。m 值应该如何选？
你的任务是编写程序，读入 n 的值，求 m 的值，使得 Ulm 最后被选择切断网络连接。

分析：
这道题也是模拟约瑟夫环问题，只不过不是求最后的胜利者，而是给定 n，要使得最后的胜利者为 2，求 m。与上题不同的是，这里的约瑟夫环问题首先淘汰的是编号为 1 的城市，然后是编号为 m+1 的城市。本题的思路是借用上例的方法，定义函数 Joseph 实现 m 和 n 取任意值的约瑟夫环问题。在主函数中读入城市个数，从 1 开始枚举 m，直到某个 m 能使得该约瑟夫环问题的最后胜利者为 2 号城市，这个 m 值就是题目要求的结果。

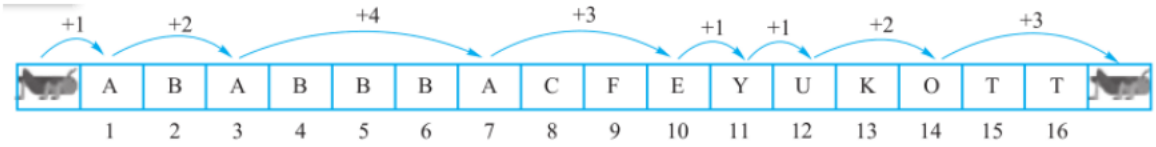
```
#include <stdio.h>
#include <string.h>
int cities;     //读入的城市个数
int circle[160];    //每个城市的序号，初始时
int temp[160];    //临时
bool Joseph( int n, int m ) {
    int r = 1;
    int i, j;
    circle[1] = 0; //第 1 个城市首先被淘汰
    for( i=2, j=1; r<=n-2; i=i%n+1, j=j%m+1 ) //剩余 n-2 轮游戏
    {
        while( circle[i]==0 ) { i = i%n+1; }//跳过已经被切断的
        if( j%m==0 )
        {
          if( i==2 ) //如果将被切断的城市是 2 号城市，提前结束
          return false;
          circle[i] = 0;
          j = 0;
          r = r+1;
```

```
            }
        }
    if( circle[2]!=0 ) return true;//n 轮游戏后，2 号城市还没被切断
    else return false;
}
int main( )
{
    int i;
    scanf("%d", &cities);
        for( i=0; i<=cities; i++ )
            circle[i] = i;
        memcpy( temp, circle, sizeof(circle) );
        int m;
        for( m=1;  ; m++ ) //不停地选择 m，直到某个 m 满足题目要求
        {
            memcpy( circle, temp, sizeof(circle) );
            if( Joseph(cities, m) ) break;
        }
        printf( "%d\n", m );
    return 0;
}
```

3 [6865]蚱蜢

有一天，一只蚱蜢像往常一样在草地上愉快地跳跃，它发现了一条写满了英文字母的纸带。蚱蜢只能在元音字母(A、E、I、O、U、Y)间跳跃，一次跳跃所需的能力是两个位置的差。纸带所需的能力值为蚱蜢从纸带开头的前一个位置根据规则跳到纸带结尾的后一个位置的过程中能力的最大值。

蚱蜢想知道跳跃纸带所需的能力值(最小)是多少。如图 9.3-1 所示的纸带所需的能力值(最小)是 4。



[输入格式]

一行一个字符串，字符串长不超过 100。

[输出格式]

一行一个整数，代表(最小)能力值。

[输入样例]

KMLPTGFHNBVCDRFGHNMBVXWSQFDCVBNHTJKLPMNFVCKMLPTGFHNBVCDRFGHNMBVXWSQFDCVBNHTJKLPMNFVC

[输出样例]

85

[问题分析]

从头到尾枚举纸带的每一个字母，按照规则模拟蚱蜢在元音字母之间跳跃的过程，打擂台记录能力值。

```
#include<bits/stdc++.h>
using namespace std;
char t[101];
int main()
{
    scanf("%s",t+1);
    int n=strlen(t+1);
    int ans=0,x=0;
    for(int i=1;i<=n;i++)
    {
        if(t[i]=='A'||t[i]=='E'||t[i]=='O'||t[i]=='U'||t[i]=='I'||t[i]=='Y')
        {
            ans=max(ans,i-x);
            x=i;
        }
    }
    ans=max(n+1-x,ans);
    cout<<ans<<endl;
}
```

4 [6866]遭遇战

小林和小华在一个 n×n 的矩形方格里玩游戏，矩形左上角为(0，0)，右下角为(n-1，n-1)。两人同时进入地图的随机位置，并以相同速度进行走位。为了隐蔽性，两人都不会再走自己走过的格子。如果两人向某一方向前进，那么他们会跑到不能跑为止，当不能跑的时候，小林会向右转，小华则会向左转，如果不能跑，则不再动。现在已知两人进入地图的初始位置和方向，请算出两人遭遇的位置。

[输入格式]

第 1 行 1 个正整数 t，表示测试数据组数，1≤t≤10。

接下来的 t 组数据，每组数据的第 1 行包含 1 个整数 n，1≤n≤1000。

第 2 行包含 3 个整数 x、y 和 d，表示小林的初始位置和一开始跑的方向。其中，d=0 表示东； d=1 表示南；d=2 表示西；d=3 表示北。

第 3 行与第 2 行格式相同，但描述的是小华。

[输出格式]

输出 t 行，若会遭遇，则包含两个整数，表示他们第一次相遇格子的坐标，否则输出"-1"。

[输入样例]

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

[输出样例]

-1

1 3

[问题分析]

设置两个布尔型数组，分别记录模拟每个人走过的格子。如果两人没有相遇并且还可以跑，就让他们按照规则一直跑下去。

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

int n,x1,y1,d1,x2,y2,d2;
int movex[4]={0,1,0,-1},movey[4]={1,0,-1,0};
bool book1[1001][1001],book2[1001][1001];//记录
bool stop1,stop2;
void rr1(bool xyds)//小林
{
    book1[x1][y1]=1;
    int xx=x1+movex[d1],yy=y1+movey[d1];
    if(xx<0||xx>=n||yy<0||yy>=n||book1[xx][yy]){//判断越界、重复情况
        if(xyds){
            stop1=1;//走到地图边缘死胡同就停下
            return;
        }
        else d1=(d1+1)%4,rr1(1);//右转
    }
    else x1=xx,y1=yy;
}
void rr2(bool xyds)//小华
{
    book2[x2][y2]=1;
    int xx=x2+movex[d2],yy=y2+movey[d2];
    if(xx<0||xx>=n||yy<0||yy>=n||book2[xx][yy]){
        if(xyds){
            stop2=1;
            return;
        }
        else d2=(d2+3)%4,rr2(1);//左转
    }
    else x2=xx,y2=yy;
}
int main()
{
    int t;
```

```
    scanf("%d",&t);
    for(int i=1;i<=t;i++){
        memset(book1,0,sizeof(book1));
        memset(book2,0,sizeof(book2));
        stop1=0,stop2=0;//初始化
        scanf("%d%d%d%d%d%d%d",&n,&x1,&y1,&d1,&x2,&y2,&d2);
        if(x1==x2||y1==y2) printf("%d %d\n",x1,y1);//特判，当两人开始时就在同一点，则直接输出
        else{
            for(;;){//一直跑下去！！！
                if(!stop1) rr1(0);
                if(!stop2) rr2(0);
                if(stop1&&stop2){
                    printf("-1\n");//都凉了
                    break;
                }
                if(x1==x2&&y1==y2){//碰上了
                    printf("%d %d\n",x1,y1);
                    break;
                }
            }
        }
    }
    return 0;
}
```

5 [3231] 图书管理员

题目描述：
图书馆中每本书都有一个图书编码，可以用于快速检索图书，这个图书编码是一个 正整数。每位借书的读者手中有一个需求码，这个需求码也是一个正整数。如果一本书的图 书编码恰好以读者的需求码结尾，那么这本书就是这位读者所需要的。 小 D 刚刚当上图书馆的管理员，她知道图书馆里所有书的图书编码，她请你帮她写 一个程序，对于每一位读者，求出他所需要的书中图书编码最小的那本书，如果没有他 需要的书，请输出-1。

说明
【数据规模与约定】
对于 20%的数据，1 ≤ n ≤ 2。
另有 20%的数据，q = 1。
另有 20%的数据，所有读者的需求码的长度均为 1。
另有 20%的数据，所有的图书编码按从小到大的顺序给出。
对于 100%的数据，1 ≤ n ≤ 1,000，1 ≤ q ≤ 1,000，所有的图书编码和需求码均 不超过 10,000,000。

输入
输入格式：
输入文件的第一行，包含两个正整数 n 和 q，以一个空格分开，分别代表图书馆里 书的数量和读者的数量。

接下来的 n 行，每行包含一个正整数，代表图书馆里某本书的图书编码。

接下来的 q 行，每行包含两个正整数，以一个空格分开，第一个正整数代表图书馆 里读者的需求码的长度，第二个正整数代表读者的需求码。

输出
输出格式：
输出文件有 q 行，每行包含一个整数，如果存在第 i 个读者所需要的书，则在第 i 行输出第 i 个读者所需要的书中图书编码最小的那本书的图书编码，否则输出-1。

样例输入
5 5
2123
1123
23
24
24
2 23
3 123
3 124
2 12
2 12
样例输出
23
1123
-1
-1
-1
思路：比对合法的方法考虑的是取余求末几位，使得它与需求编码的长度相同
查最小值的时候，可以先排序，节省时间。
代码：
```
#include<bits/stdc++.h>
using namespace std;
int book[1005];
int main()
{
        int n, q;
        cin >> n >> q;
        for (int i = 1; i <= n; i++)
```

```
        {
                cin >> book[i];
        }
        sort(book + 1, book + n + 1);//排序节约时间
        int len, num;
        int flag = 1;
        for (int i = 1; i <= q; i++)
        {
                cin >> len >> num;
                flag = 1;
                for (int j = 1; j <= n; j++)
                {
                        int xx = pow(10, len);
                        if (book[j] % xx == num)//取余判断
                        {
                                flag = 0;
                                cout << book[j] << endl;
                                break;
                        }
                }
                if (flag == 1)
                {
                        cout << "-1" << endl;
                }
        }
        int z;
        cin >> z;

}
```

6 [6867]保龄球

打保龄球是用一个滚球去打击十个站立的柱，将柱击倒。一局分十轮，每轮可滚球一次或多次，以击倒的柱数为依据计分。一局得分为十轮得分之和，而每轮的得分不仅与本轮滚球情况有关，还可能与后续一两轮的滚球情况有关。即某轮某次滚球击倒的柱数不仅要计入本轮得分，还可能会计入前一两轮得分。具体的滚球击柱规则和计分方法如下：

(1) 若某一轮的第一次滚球就击倒全部十个柱，则本轮不再滚球(若是第十轮则还需另加两次滚球，不妨称其为第十一轮和第十二轮，并不是所有的情况都需要滚第十一轮和第十二轮球)。该轮得分为本次击倒柱数 10 与以后两次滚球所击倒柱数之和。

(2) 若某一轮的第一次滚球未击倒十个柱，则可对剩下未倒的柱再滚球一次。如果这两次滚球击倒全部十个柱，则本轮不再滚球(若是第十轮则还需另加一次滚球)，该轮得分为这两次共击倒柱数 10 与以后一次滚球所击倒柱数之和。

(3) 若某一轮两次滚球未击倒全部十个柱，则本轮不再继续滚球，该轮得分为这两次滚球击倒的柱数之和。总之，若某一轮中一次滚球或两次滚球击倒十个柱，则本轮得分是本轮首次滚球开始的连续三次滚球击倒柱数之和(其中有一次或两次不是本轮滚球)。若一轮内二次滚球击倒柱数不足十个，则本轮得分即为这两次击倒柱数之和。下面以实例说明如下：

轮	1	2	3	4	5	6	7	8	9	10	11	12
击球情况	/	/	/	72	9/	81	8/	/	9/	/	8/	
各轮得分	30	27	19	9	18	9	20	20	20	20		
累计总分	30	57	76	85	103	112	132	152	172	192		

现在请编写一个保龄球计分程序，用来计算并输出最后的总得分。

[输入格式]

输入一行，为前若干轮滚球的情况，每轮滚球用一到两个字符表示，每一个字符表示一次击球，字符"/"表示击倒当前球道上的全部的柱，否则用一个数字字符表示本次滚球击倒的当前球道上的柱的数目，两轮滚球之间用一个空格隔开。

[输出格式]

输出一行一个整数，代表最后的得分。

[输入样例 1]

/ / / 72 9/ 81 8/ / 9/ / 8/

[输出样例 1]

192

[输入样例 2]

90 90 / 9/ 81 / / / 72 / /0

[输出样例 2]

170

[输入样例 3]

/ / / 72 9/ 81 8/ / 9/ 15

[输出样例 3]

169

[问题分析]

本题的难点在于每一次、每一轮滚球的分数不能立刻算出，可能依赖于后面 1~2 轮，所以需要多次积分。

```
 #include <iostream>
#include <string>
using namespace std;
#define SIZE 210
string s[SIZE];
int hit[SIZE][2], score[SIZE];

int main(void)
{
    int n = 1, i, res = 0;
    while (cin >> s[n])n++;
    n--;
    for (i = 1; i <= n; ++i)
```

```
    {
        if (s[i][0] == '/')
            hit[i][0] = 10;
        else if (s[i][1] == '/')
        {
            hit[i][0] = s[i][0] - '0';
            hit[i][1] = 10 - hit[i][0];
        }
        else
        {
            hit[i][0] = s[i][0] - '0'; //  每轮的击中数
            hit[i][1] = s[i][1] - '0';
        }
    }
    for (i = 10; i; --i)
    {
        if (s[i][0] == '/') //  第一次就全部击中
            score[i] = hit[i+1][0] + ((s[i+1].size() < 2) ? hit[i+2][0] : hit[i+1][1]) + 10;
        else if (s[i][1] == '/') //  第二次全部击中
            score[i] = hit[i+1][0] + 10;
        else //  没有全部击中
            score[i] = hit[i][0] + hit[i][1];
        res += score[i];
    }
    printf("%d", res);
    return 0;
}
```

课堂作业列表

[2948]醉酒的狱卒　　[3204]模拟约瑟夫环
[6869]网络堵塞　　[6865]蚱蜢
[6866]遭遇战　　[3070]乒乓球
[6867]保龄球枚举习题

模拟习题

1．[2004]

```
#include <stdio.h>
int main(){
    int i, j;
    char str1[] = "pig-is-stupid";
    char str2[] = "clever";
```

```
        str1[0] = 'd'; str1[1] = 'o';
        for (i = 7, j = 0; j < 6; i++, j++)
            str1[i] = str2[j];
        printf("%s\n", str1);
        return 0;
}
```
输出：_____

2．[2004]
```
#include <stdio.h>
int main(){
        int u[4], a, b, c, x, y, z;
        scanf("%d %d %d %d",&(u[0]), &(u[1]), &(u[2]), &(u[3]));
        a = u[0] + u[1] + u[2] + u[3] - 5;
        b = u[0] * (u[1] - u[2] / u[3] + 8);
        c = u[0] * u[1] / u[2] * u[3];
        x = (a + b + 2) * 3 - u[(c + 3) % 4];
        y = (c * 100 - 13) / a / (u[b % 3] * 5);
        if ((x + y) % 2 == 0) z = (a + b + c + x + y) / 2;
        else z = (a + b + c -x - y) * 2;
        printf("%d\n", x + y - z);
        return 0;
}
```
输入：2 5 7 4

输出：_____

3．[2005]
```
#include <stdio.h>
int main()
{
char str[20] = "Today-is-terrible!";
int i;
for (i = 6; i <= 10; i++)
if (str[i] == '-') str[i - 1] = 'x';
for (i = 12; i >= 0; i--)
if (str[i] == 't') str[i + 1] = 'e';
printf("%s\n", str);
return 0;
}
```
输出：_____

1．出列游戏：n 个人围成一圈，从第 1 个人开始报数，报数报到 m 的人出列；然后又从下一个人开始从 1 开始报数；重复 n-1 轮游戏，每轮游戏淘汰 1 个人，最后剩下的人就是胜利者。模拟该游戏，输出依次出列的位置及最后的胜利者

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

```
int main( ) {
    int n,m;
    cin>>n>>m;
    for( i=0; i<=n; i++ ) a[i] = i;
    int r = 1;  //第 r 轮
    int i,j;
    for( i=1, j=1; r<=n-1; i=__1_____  , j=__2_____    )
    {
        while( a[i]==-1 ) { i = i%n+1; }
        if(___3_____    )
        {
            printf( "%d ",a[i] );
            a[i] =-1; j = 0;
            __4_____;
        }
    }
    for( i=0; i<n; i++ )
    {
        if( a[i+1]!=-1 )
        printf( "%d\n", a[i+1]);
    }
     return 0;
 }
```

第四讲 贪心基础

4.1 理解贪心

所谓贪心算法是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。

[例 1]在 N 行 M 列的正整数矩阵中，要求从每行中选出 1 个数，使得选出的总共 N 个数的和最大。

算法分析]

要使总和最大，则每个数要尽可能大，自然应该选每行中最大的那个数。因此，我们设计出如下算法:

读入 N, M,矩阵数据;

```
total = 0;
for (int i = 1;   i <= N; i++ )
{                                          //对 N 行进行选择
    选择第 I 行最大的数,记为 K;
    Total += K;
}
```

$$\begin{bmatrix} 5 & 6 & 19 & 23 & 56 \\ 32 & 123 & 5 & 1 & 23 \\ 66 & 1 & 23 & 567 & 2 \\ 100 & 23 & 4 & 231 & 3 \end{bmatrix}$$

贪心算法没有固定的算法框架，**算法设计的关键是贪心策略的选择，贪心策略使用的前提是局部最优能导致全局最优**。必须注意的是，贪心算法不是对所有问题都能得到整体最优解，选择的贪心策略必须具备无后效性，即某个状态以后的过程不会影响以前的状态，只与当前状态有关。所以对所采用的贪心策略一定要仔细分析其是否满足无后效性。

利用贪心算法求解的问题往往具有两个重要的特性：贪心选择性质和最优子结构性质。如果满足这两个性质就可以使用贪心算法了。

（1）贪心选择

贪心选择是指原问题的**整体最优解可以通过一系列局部最优的选择得到**。应用同一规则，将原问题变为一个相似的但规模更小的子问题，而后的每一步都是当前最佳的选择。这种选择依赖于已做出的选择，但不依赖于未做出的选择。运用贪心策略解决的问题在程序的运行过程中无回溯过程。

例如，挑选苹果，如果你认为个大的是最好的，那你每次都从苹果堆中拿一个最大的，作为局部最优解，贪心策略就是选择当前最大的苹果；如果你认为最红的苹果是最好的，那你每次都从苹果堆中拿一个最红的，贪心策略就是选择当前最红的苹果。因此根据求解目标不同，贪心策略也会不同。

（2）最优子结构

当**一个问题的最优解包含其子问题的最优解**时，称此问题具有最优子结构性质。问题的最优子结构性质是该问题是否可用贪心算法求解的关键。例如原问题

S={a1，a2，… ，ai，… ，an}，通过贪心选择选出一个当前最优解{ai}之后，转化为求解子问题 S−{ai}，如果原问题的最优解包含子问题的最优解，则说明该问题满足最优子结构性质。

4.2 案例分析

1 [3867]最优装载问题

在北美洲东南部，有一片神秘的海域，那里碧海蓝天、阳光明媚，这正是传说中海盗最活跃的加勒比海（Caribbean Sea）。17 世纪时，这里更是欧洲大陆的商旅舰队到达美洲的必经之地，所以当时的海盗活动非常猖獗，海盗不仅攻击过往商人，甚至攻击英国皇家舰……

有一天，海盗们截获了一艘装满各种各样古董的货船，每一件古董都价值连城，一旦打碎就失去了它的价值。虽然海盗船足够大，但载重量为 C，每件古董的重量为 wi，海盗们该如何把尽可能多数量的宝贝装上海盗船呢？

输入第一行是一个整型数 m(m<100)表示共有 m 组测试数据。每组测试数据的第一行是两个整数 c,n(1<c,n<10000)表示该测试数据载重量 c 及古董的个数 n。

输出对于每一组输入，输出能装入的古董最大数量。每组的输出占一行

样例输入

2

30 8

4 10 7 11 3 5 14 2

45 10

5 12 7 3 20 9 15 11 8 32

样例输出

5

6

a. 问题分析

根据问题描述可知这是一个可以用贪心算法求解的最优装载问题，要求装载的物品的数量尽可能多，而船的容量是固定的，那么**优先把重量小的物品放进去**，在容量固定的情况下，装的物品最多。**采用重量最轻者先装的贪心选择策略，从局部最优达到全局最优**，从而产生最优装载问题的最优解。

（1）当载重量为定值 c 时，w_i 越小时，可装载的古董数量 n 越大。只要依次选择最小重量古董，直到不能再装为止。

（2）把 n 个古董的重量从小到大（非递减）排序，然后根据贪心策略尽可能多地选出前 i 个古董，直到不能继续装为止，此时达到最优。

b. 图解说明

表 1 古董重量清单

重量	w[i]	4	10	7	11	3	5	14	2

每个古董的重量如表 1 所示，海盗船的载重量 c 为 30，那么在不能打碎古董又不超过载重的情况下，怎么装入最多的古董？

（1）因为贪心策略是每次选择重量最小的古董装入海盗船，因此可以按照古董重量非递减排序，排序后如表 2 所示。

表 2 按重量排序后古董清单

重量	w[i]	2	3	4	5	7	10	11	14

（2）按照贪心策略，每次选择重量最小的古董放入（tmp 代表古董的重量，ans 代表已

装裁的古董个数）。
i=0，选择排序后的第 1 个，装入重量 tmp=2，不超过载重量 30，ans =1。
i=1，选择排序后的第 2 个，装入重量 tmp=2+3=5，不超过载重量 30，ans =2。
i=2，选择排序后的第 3 个，装入重量 tmp=5+4=9，不超过载重量 30，ans =3。
i=3，选择排序后的第 4 个，装入重量 tmp=9+5=14，不超过载重量 30，ans =4。
i=4，选择排序后的第 5 个，装入重量 tmp=14+7=21，不超过载重量 30，ans =5。
i=5，选择排序后的第 6 个，装入重量 tmp=21+10=31，超过载重量 30，算法结束。
即放入古董的个数为 ans=5 个

c. 代码详解

（1）数据结构定义

根据算法设计描述，我们用一维数组存储古董的重量：

```
double w[N]; //一维数组存储古董的重量
```

（2）按重量排序

可以利用 C++中的排序函数 sort，对古董的重量进行从小到大（非递减）排序。要使用此函数需引入头文件：

```
#include <algorithm>
```

（3）按照贪心策略找最优解

首先用变量 ans 记录已经装载的古董个数，tmp 代表装载到船上的古董的重量，两个变量都初始化为 0。然后按照重量从小到大排序，依次检查每个古董，tmp 加上该古董的重量，如果小于等于载重量 c，则令 ans ++；否则，退出。

```
int tmp = 0,ans = 0; //tmp 代表装载到船上的古董的重量，ans 记录已经装载的古董个数
for(int i=0;i<n;i++) {
   tmp += w[i];
   if(tmp<=c) ans ++;
   else
   break;
}
```

完整代码为：

```
#include <iostream>
#include <algorithm>
const int N=1000005;
using namespace std;
double w[N]; //古董的重量数组
int main()
{
    double c;
    int n;
    cin>>c>>n;//请输入载重量 c 及古董个数 n
    for(int i=0;i<n;i++)
      cin>>w[i]; //输入输入每个古董的重量
```

```
    1                                  //按古董重量升序排序
    double tmp=0.0;
    int ans=0; // tmp 为已装载到船上的古董重量，ans 为已装载的古董个数
    for(int i=0;i<n;i++)
    {
      tmp+=w[i];
      if(  2                    )
        ans++;
      else
        3
    }
    cout<<ans<<endl;//能装入的古董最大数量 Ans
    return 0;
}
```

2 [3868]阿里巴巴与四十大盗

有一天，阿里巴巴赶着一头毛驴上山砍柴。砍好柴准备下山时，远处突然出现一股烟尘，弥漫着直向上空飞扬，朝他这儿卷过来，而且越来越近。靠近以后，他才看清原来是一支马队，他们共有四十人，一个个年轻力壮、行动敏捷。一个首领模样的人背负沉重的鞍袋，从丛林中一直来到那个大石头跟前，喃喃地说道："芝麻，开门吧！"随着那个头目的喊声，大石头前突然出现一道宽阔的门路，于是强盗们鱼贯而入。阿里巴巴待在树上观察他们，直到他们走得无影无踪之后，才从树上下来。他大声喊道："芝麻，开门吧！"他的喊声刚落，洞门立刻打开了。他小心翼翼地走了进去，一下子惊呆了，洞中堆满了财物，还有多得无法计数的金银珠宝，有的散堆在地上，有的盛在皮袋中。突然看见这么多的金银财富，阿里巴巴深信这肯定是一个强盗们数代经营、掠夺所积累起来的宝窟。为了让乡亲们开开眼界，见识一下这些宝物，他想一种宝物只拿一个，如果太重就用锤子凿开，但毛驴的运载能力是有限的，怎么才能用驴子运走最大价值的财宝分给穷人呢？

阿里巴巴陷入沉思中……

输入每组测试数据的第一行是两个整数 n,c (1<n,c<10000)表示该测试数据宝物数量及驴子的承载重量。随后的 n 行，每行有两个正整数 wi, vi 分别表示第 i 个宝物的重量和价值(1<wi,vi<100)。

输出对于每一组输入，输出毛驴运走宝物的最大价值。每组的输出占一行,结果保留一位小数

样例输入

6 19
2 8
6 1
7 9
4 3
10 2
3 4

样例输出

24.6

a. 问题分析

假设山洞中有 n 种宝物，每种宝物有一定重量 w 和相应的价值 v，毛驴运载能力有限，只能运走 m 重

量的宝物，一种宝物只能拿一样，宝物可以分割。那么怎么才能使毛驴运走宝物的价值最大呢？

我们可以尝试贪心策略：
（1）每次挑选价值最大的宝物装入背包，得到的结果是否最优？
（2）每次挑选重量最小的宝物装入，能否得到最优解？
（3）每次选取单位重量价值最大的宝物，能否使价值最高？

思考一下，如果选价值最大的宝物，但重量非常大，也是不行的，因为运载能力是有限的，所以第 1 种策略舍弃；如果选重量最小的物品装入，那么其价值不一定高，所以不能在总重限制的情况下保证价值最大，第 2 种策略舍弃；而**第 3 种是每次选取单位重量价值最大的宝物，也就是说每次选择性价比（价值/重量）最高的宝物，如果可以达到运载重量 m 那么一定能得到价值最大。**因此采用第 3 种贪心策略，每次从剩下的宝物中选择性价比最高的宝物。

（1）数据结构及初始化。将 n 种宝物的重量和价值存储在结构体 three（包含重量、价值、性价比 3 个成员）中，同时求出每种宝物的性价比也存储在对应的结构体 three 中，将其按照性价比从高到低排序。采用 sum 来存储毛驴能够运走的最大价值，初始化为 0。

（2）根据贪心策略，按照性价比从大到小选取宝物，直到达到毛驴的运载能力。每次选择性价比高的物品，判断是否小于 m（毛驴运载能力），如果小于 m，则放入，sum（已放入物品的价值）加上当前宝物的价值，m 减去放入宝物的重量；如果不小于 m，则取该宝物的一部分 m * p[i]，m=0，程序结束。m 减少到 0，则 sum 得到最大值。

b. 图解说明

假设现在有一批宝物，价值和重量如表 3 所示，毛驴运载能力 m=30，那么怎么装入最大价值的物品？

表 3 宝物清单

宝物	i	1	2	3	4	5	6	7	8	9	10
重量	w[i]	4	2	9	5	5	8	5	4	5	5
价值	v[i]	3	8	18	6	8	20	5	6	7	15

（1）因为贪心策略是每次选择性价比（价值/重量）高的宝物，可以按照性价比降序排序，排序后如表 4 所示。

表 4 排序后宝物清单

宝物	i	2	10	6	3	5	8	9	4	7	1
重量	w[i]	2	5	8	9	5	4	5	5	5	4
价值	v[i]	8	15	20	18	8	6	7	6	5	3
性价比	p[i]	4	3	2.5	2	1.6	1.5	1.4	1.2	1	0.75

（2）按照贪心策略，每次选择性价比高的宝物放入：
第 1 次选择宝物 2，剩余容量 30-2=28，目前装入最大价值为 8。
第 2 次选择宝物 10，剩余容量 28-5=23，目前装入最大价值为 8+15=23。
第 3 次选择宝物 6，剩余容量 23-8=15，目前装入最大价值为 23+20=43。
第 4 次选择宝物 3，剩余容量 15-9=6，目前装入最大价值为 43+18=61
第 5 次选择宝物 5，剩余容量 6-5=1，目前装入最大价值为 61+8=69。
第 6 次选择宝物 8，发现上次处理完时剩余容量为 1，而 8 号宝物重量为 4，无法全部

放入，那么可以采用部分装入的形式，装入 1 个重量单位，因为 8 号宝物的单位重量价值为 1.5，因此放入价值 1×1.5=1.5，你也可以认为装入了 8 号宝物的 1/4，目前装入最大价值为 69+1.5=70.5，剩余容量为 0。

c. 代码详解

（1）数据结构定义

根据算法设计中的数据结构，我们首先定义一个结构体 Baowu：

```
struct Baowu{
    double w; //每种宝物的重量
    double v; //每种宝物的价值
    double p; //每种宝物的性价比（价值/重量）
};
```

（2）性价比排序

我们可以利用 C++中的排序函数 sort，对宝物的性价比从大到小（非递增）排序。

```
bool cmp(Baowua,Baowub)//比较函数按照宝物性价比降序排列 {
    return a.p > b.p; //指明按照宝物性价比降序排列
 }
sort(s, s+n, cmp);
```

（3）贪心算法求解

在性价比排序的基础上，进行贪心算法运算。如果剩余容量比当前宝物的重量大，则可 以放入，剩余容量减去当前宝物的重量，已放入物品的价值加上当前宝物的价值。如果剩余容量比当前宝物的重量小，表示不可以全部放入，可以切割下来一部分（正好是剩余容量），然后令剩余容量乘以当前物品的单位重量价值，已放入物品的价值加上该价值，即为能放入宝物的最大价值。

```
for(int i = 0;i < n;i++)//按照排好的顺序，执行贪心策略
{
    if( m > s[i].w )//如果宝物的重量小于毛驴剩下的运载能力，即剩余容量
    {
        m -= s[i].w;
        sum += s[i].v;
    }
    else //如果宝物的重量大于毛驴剩下的承载能力
    {
        sum += m*s[i].p; //进行宝物切割，切割一部分(m 重量)，正好达到驴子承重
        break;
    }
}
```

完整代码：

```
#include<bits/stdc++.h>
using namespace std;
struct bao_wu{
```

```
        int w;//每个宝物的重量
        int v;//每个宝物的价值
        double p;//性价比
}bw[10001];
int cmp(__1__){
        __2__//根据性价比从大到小排序
}
int main(){
        int n,c;
        double sum=0;
        cin>>n>>c;
        for(int i=0;i<n;i++){
                cin>>bw[i].w>>bw[i].v;//输入
                __3__//计算性价比
        } //输入数据，初始化
        __4__ //对决策参数（性价比）作排序操作
        for(int i=0;i<n;i++){
                if(bw[i].w<=c){
                        __5__ //获得价值
                        __6__ //装入背包，重量减小
                }
                else{//没法把当前物品整个装入，进行分割
                        __7__
                        break;
                }
        }
        printf("%.11f",sum);
        return 0;
}
```

4 进一步分析

想一想，**如果宝物不可分割，贪心算法是否能得到最优解？**下面我们看一个简单的例子。假定物品的重量和价值已知，如表 5 所示，最大运载能力为 10。采用贪心算法会得到怎样的结果？

表 5 物品清单

物品	i	1	2	3	4	5
重量	w[i]	3	4	6	10	7
价值	v[i]	15	16	18	25	14
性价比	p[i]	5	4	3	2.5	2

如果我们采用贪心算法，先装性价比高的物品，且物品不能分割，剩余容量如果无法再装入剩余的物品，不管还有没有运载能力，算法都会结束。那么我们选择的物品为 1 和 2，总价值为 31，而实际上还有 3 个剩余容量，但不足以装下剩余其他物品，因此得到的最大值为 31。但实际上我们如果选择物品 2 和 3，正好达到运载能力，得到的最大价值为 34。也就是说，在物品不可分割、没法装满的情况下，贪心算法并不能得到最优解，仅仅是最优解的近似解。

想一想，为什么会这样呢？

物品可分割的装载问题我们称为背包问题，物品不可分割的装载问题我们称之为 0-1 背包问题。在物品不可分割的情况下，即 0-1 背包问题，已经不具有贪心选择性质，原问题的整体最优解无法通过一系列局部最优的选择得到，因此这类问题得到的是近似解。如果一个问题不要求得到最优解，而只需要一个最优解的近似解，则不管该问题有没有贪心选择性质都可以使用贪心算法。

3 [3407] 看电视-会议安排

暑假到了，小明终于可以开心的看电视了。但是小明喜欢的节目太多了，他希望尽量多的看到完整的节目。现在他把他喜欢的电视节目的转播时间表给你，你能帮他合理安排吗？
输入包含多组测试数据。每组输入的第一行是一个整数 n（n<=100），表示小明喜欢的节目的总数。
接下来 n 行，每行输入两个整数 si 和 ei（1<=i<=n），表示第 i 个节目的开始和结束时间，为了简化问题，每个时间都用一个正整数表示。
当 n=0 时，输入结束。
输出
对于每组输入，输出能完整看到的电视节目的个数。
样例输入
12
1 3
3 4
0 7
3 8
15 19
15 20
10 15
8 18
6 12
5 10
4 14
2 9
0
样例输出
5

a. 问题分析

这是一个典型的会议安排问题，**会议安排的目的是能在有限的时间内召开更多的会议（任何两个会议不能同时进行）**。在会议安排中，每个会议 i 都有起始时间 bi 和结束时间 ei，且 bi<ei，即一个会议进行的时间为半开区间[bi，ei)。如果[bi，ei)与[bj，ej)均在"有限的时间内"，且不相交，则称会议 i 与会议 j 相容的。也就是说，当 bi≥ej 或 bj≥ei 时，会议 i 与会议 j 相容。会议安排问题要求在所给的会议集合中选出最大的相容活动子集，即尽可能在有限的时间内召开更多的会议。

在这个问题中，"有限的时间内（这段时间应该是连续的）"是其中的一个限制条件，也应该是有一个起始时间和一个结束时间（简单化，起始时间可以是会议最早开始的时间，结束时间可以是会议最晚结束的时间），任务就是实现召开更多的满足在这个"有限的时间内"等待安排的会议，会议时间表如表 6 所示。

表 6 会议时间表

会议	i	1	2	3	4	5	6	7	8	9	10
开始时间	bi	8	9	10	11	13	14	15	17	18	16
结束时间	ei	10	11	15	14	16	17	17	18	20	19

会议安排的时间段如图所示。

从表 6 中可以看出，{会议 1，会议 4，会议 6，会议 8，会议 9}，{会议 2，会议 4，会议 7，会议 8，会议 9}都是能安排最多的会议集合。

要让会议数最多，我们需要选择最多的不相交时间段。我们可以**尝试贪心策略**：

（1）每次从剩下未安排的会议中选择会议具有最早开始时间且与已安排的会议相容的会议安排，以增大时间资源的利用率。

（2）每次从剩下未安排的会议中选择持续时间最短且与已安排的会议相容的会议安排，这样可以安排更多一些的会议。

（3）每次从剩下未安排的会议中选择具有最早结束时间且与已安排的会议相容的会议安排，这样可以尽快安排下一个会议。

思考一下，如果选择最早开始时间，则如果会议持续时间很长，例如 8 点开始，却要持续 12 个小时，这样一天就只能安排一个会议；如果选择持续时间最短，则可能开始时间很晚，例如 19 点开始，20 点结束，这样也只能安排一个会议，所以我们最好选择那些开始时间要早，而且持续时间短的会议，即最早开始时间+持续时间最短，就是最早结束时间。

因此采用第（3）种贪心策略，每次从剩下的会议中选择具有最早结束时间且与已安排的会议相容的会议安排。

b. 算法设计

（1）初始化：将 n 个会议的开始时间、结束时间存放在结构体数组中，如果需要知道选中了哪些会议，还需要在结构体中增加会议编号，然后按结束时间从小到大排序（非递减），结束时间相等时，按开始时间从大到小排序（非递增）；

（2）根据贪心策略就是选择第一个具有最早结束时间的会议，用 last 记录刚选中会议的结束时间；

（3）选择第一个会议之后，依次从剩下未安排的会议中选择，如果会议 i 开始时间大于等于最后一个选中的会议的结束时间 last，那么会议 i 与已选中的会议相容，可以安排，更新 last 为刚选中会议的结束时间；否则，舍弃会议 i，检查下一个会议是否可以安排。

(1)原始的会议时间表（见表 7）：

表 7 原始会议时间表

会议	num	1	2	3	4	5	6	7	8	9	10
开始时间	beg	3	1	5	2	5	3	8	6	8	12
结束时间	end	6	4	7	5	9	8	11	10	12	14

(1) 排序后的会议时间表（见表 8）：

表 8 排序后的会议时间表

会议	num	2	4	1	3	6	5	8	7	9	10
开始时间	beg	1	2	3	5	3	5	6	8	8	12
结束时间	end	4	5	6	7	8	9	10	11	12	14

(2) 贪心选择过程

（1）首先选择排序后的第一个会议即最早结束的会议（编号为 2），用 last 记录最后一个被选中会议的结束时间，last=4。

（2）检查余下的会议，找到第一个开始时间大于等于 last（last=4）的会议，子问题转化为从该会议开始，余下的所有会议。如表 8 所示。

表 9 会议时间表

会议	num	2	4	1	3	6	5	8	7	9	10
开始时间	beg	1	2	3	5	3	5	6	8	8	12
结束时间	end	4	5	6	7	8	9	10	11	12	14

从子问题中，选择第一个会议即最早结束的会议（编号为 3），更新 last 为刚选中会议的结束时间 last=7。

（3）检查余下的会议，找到第一个开始时间大于等于 last（last=7）的会议，子问题转化为从该会议开始，余下的所有会议。如表 9 所示。

表 10 会议时间表

会议	num	2	4	1	3	6	5	8	7	9	10
开始时间	beg	1	2	3	5	3	5	6	8	8	12
结束时间	end	4	5	6	7	8	9	10	11	12	14

从子问题中，选择第一个会议即最早结束的会议（编号为 7），更新 last 为刚选中会议的结束时间 last=10。

（4）检查余下的会议，找到第一个开始时间大于等于 last（last=11）的会议，子问题转化为从该会议开始，余下的所有会议。如表 11 所示。

表 11 会议时间表

会议	num	2	4	1	3	6	5	8	7	9	10
开始时间	beg	1	2	3	5	3	5	6	8	8	12
结束时间	end	4	5	6	7	8	9	10	11	12	14

从子问题中，选择第一个会议即最早结束的会议（编号为 10），更新 last 为刚选中会议的结束时间 last=14；所有会议检查完毕，算法结束。如表 11 所示。

4．构造最优解

从贪心选择的结果，可以看出，被选中的会议编号为{2，3，7，10}，可以安排的会议数量最多为 4，如表 12 所示。

c. 代码详解

（1）数据结构定义

以下 C++程序代码中，结构体 meet 中定义了 beg 表示会议的开始时间，end 表示会议的结束时间，会议 meet 的数据结构：

```
struct Meet {
    int beg; //会议的开始时间
    int end; //会议的结束时间
} meet[1000];
```

（2）对会议按照结束时间非递减排序

我们采用 C++中自带的 sort 函数，自定义比较函数的办法，实现会议排序，按结束时间

从小到大排序（非递减），结束时间相等时，按开始时间从大到小排序（非递增）：

```
bool cmp(Meet x,Meet y) {
  if(x.end==y.end) //结束时间相等时
  return x.beg>y.beg; //按开始时间从大到小排序
return x.end<y.end; //按结束时间从小到大排序
}
sort(meet,meet+n,cmp);
```

（3）会议安排问题的贪心算法求解

在会议按结束时间非递减排序的基础上，首先选中第一个会议，用 last 变量记录刚刚被选中会议的结束时间。下一个会议的开始时间与 last 比较，如果大于等于 last，则 选中。每次选中一个会议，更新 last 为最后一个被选中会议的结束时间，被选中的会议数 ans 加 1；如果会议的开始时间不大于等于 last，继续考查下一个会议，直到所有会议考查完毕。

```
int ans=1; //用来记录可以安排会议的个数，初始时选中了第一个会议
int last = meet[0].end; //last 记录第一个会议的结束时间
for( i = 1;i < n; i++) //依次检查每个会议 {
   if(meet[i].beg > =last)
   {//如果会议 i 开始时间大于等于最后一个选中的会议的结束时间
      ans++;
      last = meet[i].end; //更新 last 为最后一个选中会议的结束时间
   }
}
return ans; //返回可以安排的会议最大数
```

上面介绍的程序中，只是返回了可以安排的会议最大数，而不知道安排了哪些会议，这显然是不满足需要的。我们可以改进一下，在会议结构体 meet 中添加会议编号 num 变量，选中会议时，显示选中了第几个会议。

```
#include<bits/stdc++.h>
using namespace std;
struct meeting{
   int s;
   int e;
}meet[101];

int cmp(meeting a, meeting b){
   1                          return a.e<b.e;
}

int main(){
   int last,n;
   int sum;
   while(cin>>n){
      if(n==0)break;
```

```
    for(int i=0;i<n;i++){
      cin>>meet[i].s>>meet[i].e;
    }//输入数据
    __2_____________________sort(meet,meet+n,cmp);
    sum=1;//选择第一个会议
    __3_____________________last=meet[0].e;//当前会议的结束时间
    for(int j=1;j<n;j++){//扫描除第一个会议外的所有会议
      if(__4_____________meet[j].s>=last){
        sum++;
        __5_____________________ last=meet[j].e;//更新 last
      }
    }
    cout<<sum<<endl;
  }

  return 0;
}
```

课堂作业列表

3867　最优装载问题
3868　阿里巴巴与四十大盗
3407　会议安排
1912　硕鼠
6990　购买贺年卡(card)
2349　贪心的小猫咪

课前练习

```
#include<iostream>
#include<string>
using namespace std;
int main()
{
    string map= "22233344455566677778889999";
    string tel;
    int i;
    cin>>tel;
    for(i=0;i<tel.length();i++)
       if((tel[i]>='0') && (tel[i]<='9') )
           cout<<tel[i];
       else if( (tel[i]>='A') && (tel[i]<='Z'))
           cout<<map[tel[i]-'A'];
```

```
    cout<<endl;
    return 0;
}
```

输入：CCF-NOIP-2011

输出：_____

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string a, b;
    int i;
    a = "AABBCCDKKRRSSXX";
    cin >> b;
    for (i = 0; i < b.length(); ++i)
    {
        if ((b[i] >= '0') && (b[i] <= '9'))
            cout << b[i];
        else
        {
            if ((b[i] >= 'A') && (b[i] <= 'Z'))
                cout << a[b[i] - 'A' - 1];
        }
    }
    return 0;
}
```

输入：NOIP-2012

输出：_____

贪心习题

1. 执行以下程序段后，k 的值是____

```
int a[7],b[4],i,k=0;
for( i=0;i<7;i++ )  a[i]=i*i;
for( i=0;i<4;i++ )
{   b[i]=a[i*(i-1)];
        k=k+b[i];
}
```

2.[2016]

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

```
    int i = 100, x = 0, y = 0;
    while (i > 0) {
        i--;
        x = i % 8;
     if (x == 1) y++;
    }
    cout << y << endl;
    return 0;
}
```

输出：_____

3.[2015]

```
#include <iostream>
using namespace std;
int main() {
    int a,b,c;
    a=1; b=2; c=3;
    if (a>c){
        if(a>c)
        cout << a <<" ";
        else
        cout << b <<" ";
    }
    cout << c << endl;
    return 0;
}
```

输出：_____

4. 阿里巴巴填空

```
#include<iostream>
#include<algorithm>
using namespace std;
struct bao_wu{
        ____1____
}bw[10001];
int cmp(bao_wu a, bao_wu b){
    return a.p>b.p;
}
int main(){
    int n,c;
    double sum=0;
    cin>>n>>c;
    for(int i=0;i<n;i++){
        cin>>bw[i].w>>bw[i].v;
```

```
        2                                   ;
    }//输入数据，初始化
    3                                   ;
        for(int i=0;i<n;i++){
        if(     4                 ){
            sum=sum+bw[i].v; //获得价值
            5                                   //装入背包，重量减小
        }
        else{//没法把当前物品整个装入，进行分割
            6
            7
        }
    }
    printf("%.1lf",sum);



    return 0;
}
```

第五讲 贪心综合

重点：

1. 体会贪心法的基本思想。
2. 熟练应用贪心法求解一些实际问题

5.1 [3780]排队打水

有 n 个人排队到 m 个水龙头去打水，他们装满水桶的时间 t1, t2 , ……, tn 为整数且各不相同，应如何安排他们的打水顺序才能使他们花费的总时间最少?

输入 n m 输出 所有人的花费时间总和

样例输入

4 2

2 6 4 5

样例输出

23

5.1.1 分析

由于排队时，越靠前面的计算的次数越多，显然越小的排在越前面得出的结果越小，所以这道题可以用贪心法解答，基本步骤：

(1)将输入的时间按从小到大排序；

(2)将排序后的时间按顺序依次放入每个水龙头的队列中；

(3)统计，输出答案。

参考程序主要框架如下：

```
cin>>n>>r;
memset(s,0,sizeof(s));                    //初始化
j=0; minx=0;
for (i=1;i<=n;i++)                  //用贪心法求解
 {n 人打水，r 个水龙头
    j++;
    if (j==r+1) j=1;     //前 r 个人为一组，第 r+1 个人回到第一个水龙
    s[j]+=a[i];          //加上等待时间
    minx+=s[j];
 }
cout<<minx;     //输出解答
```

5.1.2 代码

```
#include<iostream>
```

```
#include<algorithm>
using namespace std;
int main()
{
   int t[10002]={0},sum=0,j=0,a[10002]={0},m,n;
   cin>>n>>m;
   for(int i=1;i<=n;i++)
      cin>>t[i];
   sort(t+1,t+n+1);
   for(int i=1;i<=n;i++)
   {
      j++;
      if(j==____1_____   m+1)
            j= ____2_____ j=1;
      a[j] ____3_____          a[j]+=t[i];_____
      sum____3_____ sum+=a[j];
   }
   cout<<sum<<endl;
}
```

```
#include<iostream>
#include<algorithm>
const int N=1000,M=100;
using namespace std;
int n,m,a[N],b[M],sum;
int main()
{
   scanf("%d%d",&n,&m);
   for(int i=0;i<n;i++) scanf("%d",&a[i]);
   sort(a,a+n);
   for(int i=0;i<m;i++) b[i]=a[i];
   for(int i=m;i<n;i++) b[i]=b[i-m]+a[i];
   for(int i=0;i<n;i++) sum+=b[i];
   printf("%d",sum);
   return 0;
}
```

5.2 [3066] 均分纸牌

有 N 堆纸牌，编号分别为 1，2，…，N。每堆上有若干张，但纸牌总数必为 N 的倍数。可以在任一堆上取若干张纸牌，然后移动。

移牌规则为：在编号为 1 的堆上取的纸牌，只能移到编号为 2 的堆上；在编号为 N 的堆上取的纸牌，只能移到编号为 N-1 的堆上；其他堆上取的纸牌，可以移到相邻左边或右边的堆上。

现在要求找出一种移动方法，用最少的移动次数使每堆上纸牌数都一样多。
例如 N=4，4 堆纸牌数分别为：
① 9 ② 8 ③ 17 ④ 6
移动 3 次可达到目的：
从③取 4 张牌放到④（9 8 13 10）->从③取 3 张牌放到②（9 11 10 10）->从②取 1 张牌放到①（10 10 10 10）。

输入
每个测试文件只包含一组测试数据，每组输入的第一行输入一个整数 N（1<=N<=100），表示有 N 堆纸牌。
接下来一行输入 N 个整数 A1 A2…An，表示每堆纸牌初始数，1<=Ai<=10000。
输出
对于每组输入数据，输出所有堆均达到相等时的最少移动次数。
样例输入
4
9 8 17 6
样例输出
3

5.2.1 分析

把每堆牌的张数减去平均张数，题目就变成移动正数，加到负数中，使大家都变成 0，那就意味着成功了一半！平均张数为 10,原张数 9，8，17，6，变为 -1，-2，7，-4，其中没有为 0 的数，我们从左边出发：

1. 要使第 1 堆的牌数-1 变为 0，只须将-1 张牌移到它的右边（第 2 堆）-2 中；
2. 结果是-1 变为 0，-2 变为-3，各堆牌张数变为 0，-3，7，-4。

9， 8， 17， 6
-1， -2， 7， -4
0， -3， 7， -4
0 0 4 -4
0 0 0 0

如果张数中本来就有为 0 的，怎么办呢？如 0，-1，-5，6，还是从左算起（从右算起也完全一样），第 1 堆是 0，无需移牌，余下与上相同；再比如-1，-2，3，10，-4，-6，从左算起，第 1 次移动的结果为 0，-3，3，10，-4，-6；第 2 次移动的结果为 0，0，0，10，-4，-6，现在第 3 堆已经变为 0 了，可节省 1 步，余下继续。

本题有 3 个关键点：

一是善于将每堆牌数减去平均数，简化了问题；
二是要过滤掉 0（不是所有的 0，如-2，3，0，-1 中的 0 是不能过滤的）；
三是负数张牌也可以移动，这是关键中的关键。

5.2.2 代码

```
#include<bits/stdc++.h>
using namespace std;
int main(){
  int n,ave,step,a[101];
  int i,j;
```

```
    cin>>n;
    ave=0,step=0;
    for (i=1;i<=n;++i)
      cin>>a[i],ave+=a[i];//读入各堆牌张数，求总张数 ave
    ave/=n;    //求牌的平均张数 ave
    for (i=1;i<=n;++i) a[i]-=ave; //每堆牌的张数减去平均数
    __1_____                    i=1;j=n;
    __2_____                  while (a[i]==0&&i<n) i++; //过滤左边的 0
    __3_____                  while (a[j]==0&&j>1) j--; //过滤右边的 0
    while (i<j)
    {
      a[i+1]__4_____                +=a[i];  //将第 i 堆牌移到第 i+1 堆中去
      a[i]__5_____                  =0;      //第 i 堆牌移走后变为 0
      step++;       //移牌步数计数
      i++;          //对下一堆牌进行循环操作
      while (a[i]==0&&i<j)
            i++;  //过滤移牌过程中产生的 0
    }
    cout<<step<<endl;
}
```

5.3 [2872] 删数问题

输入一个高精度的正整数 n，去掉其中任意 s 个数字后剩下的数字按原左右次序组成一个新的正整数。编程对给定的 n 和 s，寻找一种方案使得剩下的数字组成的新数最小。输出新的正整数。（n 不超过 240 位）输入数据均不需判错。

输入 n，s

输出 最后剩下的最小数。

样例输入

175438

4

样例输出

13

5.3.1 算法分析

由于正整数 n 的有效数位为 240 位，所以很自然地采用字符串类型存贮 n。

那么如何决定哪 s 位被删除呢？是不是最大的 s 个数字呢？

例如：n=175438

s=4

删数的过程如下：

n=175438　　//删掉 7

15438　　//删掉 5

1438　　//删掉 4
138　　//删掉 8
13　　//解为 13

这样，删数问题就与如何寻找递减区间首字符问题对应起来。不过还要注意一个细节性的问题，就是**可能会出现字符串串首有若干 0 的情况**，甚至整个字符串都是 0 的情况。如 900067800

贪心策略为：每一步总是选择一个使剩下的数最小的数字删去，即按高位到低位的顺序搜索，若各位数字递增，则删除最后一个数字；否则删除第一个递减区间的首字符，这样删一位便形成了一个新数字串。然后回到串首，按上述规则再删下一个数字。重复以上过程 s 次为止，剩下的数字串便是问题的解了。

5.3.2 代码

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

char str[260];
int main(){
  int len,i,j,s;
  cin>>str>>s;
  len=strlen(str);
  while(s--){//删除 s 个数字
    for(i=__1__0;i<=len-2;i++)
      if(__2__str[i]>str[i+1]){//找到满足字符
        for(j=i;j<=len-2;j++)
          __3__str[j]=str[j+1];//删除
        break;
      }
    len--;
  }
  i=0;
  __4__while(i<=len-1&&str[i]=='0')i++;//处理前导 0
  if(i==len)printf("0");
  else
    for(j=i;j<=len-1;j++)
      cout<<str[j];
  return 0;
}
```

5.4 [2961] 拦截导弹问题

某国为了防御敌国的导弹袭击，开发出一种导弹拦截系统，但是这种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。所以一套系统有可能不能拦截所有的导弹。
输入导弹依次飞来的高度（雷达给出的高度不大于 30000 的正整数）。计算要拦截所有导弹最小需要配备多少套这种导弹拦截系统。
输入：n 颗依次飞来的高度（1≤n≤1000）。
输出：要拦截所有导弹最小配备的系统数 k。
样例输入：389 207 155 300 299 170 158 65
样例输出：2

5.4.1 算法分析

按照题意，被一套系统拦截的所有导弹中，最后一枚导弹的高度最低。

若导弹 i 的高度高于所有系统的最低高度，则断定导弹 i 不能被这些系统所拦截，应增设一套系统来拦截导弹 I；若导弹 i 低于某些系统的最低高度，那么导弹 i 均可被这些系统所拦截。

设 k 为当前配备的系统数；

len[k]为被第 k 套系统拦截的最后一枚导弹的高度，简称系统 k 的最低高度（1≤k≤n）。

若导弹 i 的高度高于所有系统的最低高度，则断定导弹 i 不能被这些系统所拦截，应增设一套系统来拦截导弹 len(k←k+1,l[k]←导弹 i 的高度)；

若导弹 i 低于某些系统的最低高度，那么导弹 i 均可被这些系统所拦截。

贪心策略：选择其中最低高度最小（即导弹 i 的高度与系统最低高度最接近）的一套系统 p(len[p]=min{l[j]|l[j]>导弹 i 的高度}；len[p]←导弹 i 的高度)（i≤j≤k）。这样可使得一套系统拦截的导弹数尽可能增多。依次类推，直至分析了 n 枚导弹的高度为止。此时得出的 k 便为应配备的最少系统数。

5.4.2 代码

```
#include<bits/stdc++.h>
using namespace std;
int s[1005];//存储导敌国的导弹高度
int len[1005];//第 k 套系统拦截的最后一枚导弹的高度
int main()
{
  int n;
  n=1;
  memset(s,0,sizeof(s));
  memset(len,0,sizeof(len));
  while(cin>>s[n]){
    n++;//计算导弹数量
  }
  int k=1;//当前配备的系统数
  len[k]=s[1];//把第一个导弹的高度作为第一套系统的初始高度
  int p;//表示某一套系统
  for(int i=2;i<n;i++)
  {//遍历每一颗导弹
    p=0;//记录当前导弹 j 属于哪套系统
    for(int j=1;j<=k;j++)//一共有 k 套系统
```

```
    {
      if(len[j]>=s[i])//从 k 套系统选择
      {
        if(p==0)p=j;//p 赋值为第一个满足条件的第 j 套系统
        else if(len[j]<len[p]) p=j;
        //贪心，选择最小系统中最小的值，
        //然后后面会把这个比最小系统还小的值赋值给
        //这个最小系统的最小值
      }
    }
    if(p==0)
    {//没有找到合适的系统，生成新的导弹系统
      k++;
      len[k]=s[i];//把 s[i]作为第 k 套系统的最小值
    }
    else
      len[p]=s[i];//到合适的系统,把 s[i].....
  }
  cout<<k<<endl;
  return 0;
}
```

5.5 [1206] 母舰

题目描述：

在小 A 的星际大战游戏中，一艘强力的母舰往往决定了一场战争的胜负。一艘母舰的攻击力是普通的 MA(Mobile Armor)无法比较的。 对于一艘母舰而言，它是由若干个攻击系统和若干个防御系统组成的。两艘母舰对决时，一艘母舰会选择用不同的攻击系统去攻击对面母舰的防御系统。当这个攻击系统的攻击力大于防御系统的防御力时，那个防御系统会被破坏掉。当一艘母舰的防御系统全部被破坏掉之后，所有的攻击都会攻击到敌方母舰本身上去造成伤害。 这样说，一艘母舰对对面的伤害在一定程度上是取决于选择的攻击对象的。 在瞬息万变的战场中，选择一个最优的攻击对象是非常重要的。所以需要写出一个战斗系统出来，判断出你的母舰最多能对对手造成多少伤害并加以实现。

输入描述：输入第一行两个整数 M 和 N，表示对方母舰的防御系统数量和你的母舰的攻击系统数量。接着 M 行每行一个整数每一个表示对方防御系统的防御力是多少。 接着 N 行每行一个整数每一个表示己方攻击系统的攻击力是多少。

输出描述：输出仅有一行，表示可以造成的最大伤害。

样例输入：

```
3 5
1000
2000
1200
2100
```

1200
2000
1000
1000
样例输出：
2000

5.5.1 算法分析

两艘母舰对决时，一艘母舰会选择用不同的攻击系统去攻击对面母舰的防御系统。当这个攻击系统的攻击力大于防御系统的防御力时，那个防御系统会被破坏掉。当一艘母舰的防御系统全部被破坏掉之后，所有的攻击都会攻击到敌方母舰本身上去造成伤害。为了目标是让伤害最大化，每个攻击要选最合适的防御系统，刚好能消灭防御系统（攻击=防御+k）k>=1 ,k 越小，优势越明显。

本题可用贪心求解，将敌方防御系统防御值和我方攻击系统攻击值分别 从小到大排序，再依次从小到大枚举敌方防御系统，枚举一个防御系统，就在我方还剩着的攻击系统中找到最小的且可以爆破这个防御系统的攻击系统，攻击； 完爆敌方防御系统后，将还剩下的攻击值累加即得到答案。为何排序和枚举时是从小到大呢？因为大的要留着攻击母舰。

题目中攻击最大有 10000,防御最大有 10000，O(10000*10000) 会超限，所以每次查找时不能都是从头开始找，要从上次查找到位置继续往下找。

5.5.2 代码

```
#include <bits/stdc++.h>
using namespace std;
const int M =100005;
int f[M],g[M];   // >1000 的数组定义全局变量

main(){
   int n,m;
   cin>>m>>n;
   for (int i = 0 ;i < m ;i++)
      cin>>f[i];//防御力
   for (int i = 0;i < n ;i++)
      cin>>g[i];//攻击力
   1                    sort(f,f+m);
   2                    sort(g,g+n);
   int L=0,sum=0;
   int count=0;
   for(int i = 0;i<m;i++){
      for(int j =  3                    L;j<n;j++){
        4                    L=j;//放在外面，节省循环次数
        if(f[i]<g[j]){
           count++;//计算销毁的防御系统的个数
           f[i]=0;g[j]=0;
```

```
            break;
          }
        }
    }
    if(count<m){
      cout<<"0";
      return 0;
    }
    for(int i=0;i<n;i++)
      sum+=g[i];
    cout<<sum<<endl;
    return 0;
}
```

5.6 [3854]小李发奖金

当然打台球只是小李的休闲娱乐活动，对待他的本职工作，他还是非常兢兢业业的。但是小李的老板是个周扒皮，每次都想克扣小李的工资和奖金，甚至制定出非常奇葩的规则。

又到了每年发年终奖的时候了，今年老板的规则是这样的：给你 n 个数，每次你可以对任意一个数加 1，直到所有的数都不相等为止，每加一次都要花费一定数额的费用。为了小李的幸福生活，聪明的你可否帮助小李，让他尽量少扣钱。

输入：第一行 n，表示共有 n 个数；第二行共 n 个用空格隔开的非负整数 ai。

输出：仅一个整数，表示加到让每个数都不相等的最少次数。

样例输入

4

1 1 3 2

样例输出

3

分析题意，符合贪心规则。如 1 2 2 2 2 6 8　比如排完序后是会变成 1 2 3 4 5 6 7，最优。

贪心方法：

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=3e4+20;
int a[maxn];
int main()
{
int n;
long long int ans=0;
    scanf("%d",&n);
    for(int i=1;i<=n;i++) scanf("%d",&a[i]);
    sort(a+1,a+n+1);
    for(int i=2;i<=n;i++)
```

```
    {
        if(a[i]<=a[i-1])
        {//如果当前这个数比前一个数小 就把当前这个数加到前一个数加 1
            ans=ans+a[i-1]-a[i]+1;//累加次数   a[i-1]-a[i]+1
            a[i]=a[i-1]+1;//把当前这个数变成比前一个数大 1
        }
    }
    printf("%lld\n",ans);
    return 0;
}
```

课堂作业列表

[3780] 排队打水问题
[2870] 排队接水
[3066] 均分纸牌
[2872] 删数问题
[2891] 拦截导弹问题
[3854] 小李发奖金
[2777] An Easy Problem
[2783] 电池的寿命

贪心综合习题

1. [2013]

```
#include <iostream>
using namespace std;
int main()
{
   int a, b, u, i, num;
   cin>>a>>b>>u;
   num = 0;
   for (i = a; i <= b; i++)
     if ((i % u) == 0)
            num++;
   cout<<num<<endl; return 0;
}
```

输入： 1 100 15
输出： _____

2. [2014]

```
#include <iostream>
using namespace std;
int main()
{
	int a, b, c, d, ans;
	cin >> a >> b >> c;
	d = a- b;
	a = d + c;
	ans = a * b;
	cout << "Ans = " << ans << endl;
	return 0;
}
```

输入：2 3 4

输出：_____

3. [2014]

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
   string st;
int i, len;
getline(cin, st);
  len = st.size();
  for(i = 0; i < len; i++)
        if(st[i] >= 'a' && st[i] <= 'z')
        st[i] = st[i] - 'a' + 'A';
cout << st << endl;
return 0;
}
```

输入：Hello, my name is Lostmonkey.

输出：_____

4. 有 n 个人排队到 m 个水龙头去打水，他们装满水桶的时间 t1, t2 , ……, tn 为整数且各不相同，应如何安排他们的打水顺序才能使他们花费的总时间最少?

```
#include<iostream>
#include<algorithm>
using namespace std;
int main()
{
   int t[10002]={0},sum=0,j=0,a[10002]={0},m,n;
   cin>>n>>m;
```

```
    for(int i=1;i<=n;i++)
        cin>>t[i];
    ____1_____________________
    for(int i=1;i<=n;i++)
    {
        j++;
        if(_____2_____    )
            j=1;
        a[j]=_____3_____________  ;
        sum=____4_____________;
    }
    cout<<sum<<endl;
}
```

第六讲 前缀和

1 [1058] 前缀和

输入一个长度为 n 的整数序列。接下来再输入 m 个询问，每个询问输入一对 l,r。
对于每个询问，输出原序列中从第 l 个数到第 r 个数的和。
输入
第一行包含两个整数 n 和 m。第二行包含 n 个整数，表示整数数列。
接下来 m 行，每行包含两个整数 ll 和 r，表示一个询问的区间范围。
1≤l≤r≤n,1≤n,m≤100000　　−1000≤数列中元素的值≤1000
输出共 m 行，每行输出一个询问的结果。

具体做法：

首先做一个预处理，定义一个 sum[]数组，sum[i]代表 a 数组中前 i 个数的和。

求前缀和运算：

```
const int N=1e5+10;
int sum[N],a[N]; //sum[i]=a[1]+a[2]+a[3].....a[i];
for(int i=1;i<=n;i++)
{
    sum[i]=sum[i-1]+a[i];
}
```

对于每次查询，只需执行 sum[r]-sum[l-1] ，时间复杂度为 O(1)

```
sum[r] =a[1]+a[2]+a[3]+a[l-1]+a[l]+a[l+1].....a[r];
sum[l-1]=a[1]+a[2]+a[3]+a[l-1];
sum[r]-sum[l-1]=a[l]+a[l+1]+.....+a[r];
```

```
#include<iostream>
//1058  前缀和
using namespace std;
int a[100005],b[100005];
int n,m;
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++){
        scanf("%d",&a[i]);
        b[i]=b[i-1]+a[i];
    }
```

```
    while(m--){
        int le,ri;
        scanf("%d%d",&le,&ri);
        printf("%d\n",b[ri]-b[le-1]);
    }

    return 0;
}
```

2 [8211] 子序列总和为 7 的倍数

农夫约翰的 N 头牛排成一排。每头奶牛都有一个不同的整数编号，因此约翰可以区分它们。
约翰想为一组连续的奶牛拍照，但是，由于童年创伤事件涉及数字 1…6，所以，这一组奶牛的整数编号加起来需要等于 7 的倍数。请帮助约翰确定他可以拍摄的最大奶牛数量。
输入第一行包含整数 N。接下来 N 行，包含 N 头奶牛的编号。
输出满足条件的最大的一组连续奶牛的数量。
如果不存在这样一组连续的奶牛，则输出 0。
数据范围 1≤N≤50000,奶牛编号范围 [0,106]
题意可以简化为：给你 n 个数，分别是 a[1],a[2],...,a[n]。求一个最长的区间[x,y]，使得区间中的数(a[x],a[x+1],a[x+2],...,a[y-1],a[y])的和能被 7 整除。输出区间长度。若没有符合要求的区间，输出 0。

输入：
7
3
5
1
6
2
14
10

输出：
5
如样例所示，5+1+6+2+14=27，能被 7 整除，长度为 5

这个题肯定不能直接模拟，暴力枚举端点，时间复杂度为 $O(n^3)$,即 50000^3，必然超时。

s[i]表示[1,i]的前缀和，[l,r]的和就可以拆成 s[r]-s[l-1]，当 s[r]和 s[l-1]模 7 相同时，区间就能被 7 整除，求出前缀和每个余数对应的最小的 l-1 和最大的 r 从而算出最长区间长度，可以边读入边记录。端点模 7 为 0 到 6 的最长区间长度的最大值就是答案，注意判断区间是否存在 用滚动数组，时间 O(n).

	1	2	3	4	5	6	7
a[i]	3	5	1	6	2	14	10
s[i]	3	8	9	15	17	31	41
mod 7	3	1	2	1	3	3	6

```
#include<cstdio>
#include<algorithm>
using namespace std;
int n,a,ans;
int s;//s 是前缀和,
int l[7];
   //l[i]存对 7 取模余数为 i 的最小 l-1(i=0..6)
int r[7];
   //r[i]存对 7 取模余数为 i 最大 r,
   //-1 代表没有%7 为 i 的前缀和,
//当有任意前缀和 s[x]%7 等于 0 时,最长区间长度就是 x
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%d",&a);
        s=(s+a)%7;
        if(l[s]==0)l[s]=i;//记录最左余数为 s 的位置
        r[s]=i;//记录最左余数为 s 的位置
    }
    for(int i=0;i<7;++i)
            ans=max(ans,r[i]-l[i]);
    printf("%d",ans);
}
```

3 [8398]COW Operations　奶牛操作

Bessie 找到了一个长度不超过 2e5，且仅包含字符 'C'，'O' 和 'W' 的字符串 s。她想知道是否可以使用以下操作将该字符串变为单个字母 'C'（她最喜欢的字母）：

1. 选择两个相邻相等的字母并将其删除。
2. 选择一个字母，将其替换为另外两个字母的任一排列。

求出这个字符串本身的答案对 Bessie 而言并不足够，所以她想要知道 s 的 Q（$1 \le Q \le 2 \times 10^5$）个子串的答案。

输入格式：

输入的第一行包含 s。

第二行包含 Q。
以下 Q 行每行包含两个整数 l 和 r（1≤l≤r≤|s|，其中 |s| 表示 s 的长度）。
输出格式：
输出一个长为 Q 的字符串，如果第 i 个子串可以被转变则第 i 个字符为 'Y'，否则为 'N'。
输入样例：
COW
6
1 1
1 2
1 3
2 2
2 3
3 3
输出样例：
YNNNYN
第一个询问的答案是「是」，因为 ss 的第一个字符已经等于 'C'。
第五个询问的答案是「是」，因为 ss 的第二到第三个字符组成的子串 OW 可以通过两步操作变为 'C'：
OW
-> CWW
-> C
这个样例字符串 COW 的其他子串均不能被转变为 'C'。

分析：

我们将字符串 s 排序，把相邻的删掉，最后每种字符最多有一个。完了。
用前缀和处理一下，对于每个询问，统计区间内的三个字符的个数除以 2 的余数，再根据上面的表格求出答案就行了。

时间复杂度：O(N+Q)

```
#include <bits/stdc++.h>
using namespace std;
const int N=500005;
int n,m; char a[N];int s[N];
inline int read(){
    int F=1,ANS=0;
    char C=getchar();
    while (C<'0'||C>'9')
    {
        if (C=='-') F=-1;
        C=getchar();
    }
    while (C>='0'&&C<='9')
```

```
    {
        ANS=ANS*10+C-'0';
        C=getchar();
    }
    return F*ANS;
}
int main(){
    cin >> (a+1);
    n=strlen(a+1);
    m=read();
    for (int i=1;i<=n;i++)
    {
        if (a[i]=='C') s[i]=(s[i-1]^1);
        else if (a[i]=='O') s[i]=(s[i-1]^2);
        else s[i]=(s[i-1]^3);
    }
    for (int i=1;i<=m;i++)
    {
        int l=read(),r=read(),ans=s[r]^s[l-1];
        if (ans==1) printf("Y");
        else printf("N");
    }
    return 0;
}
```

4 [8313] S 组奶牛 2

共有 N 个信号灯，编号为 1~N，有 B 个信号灯损坏，给你它们的编号。
问，最少修好几个信号灯，可以有 K 个编号连续的信号灯。

输入

第一行三个整数，N,K,B

接下来 N 行，每行一个整数代表坏掉的信号灯

输出

一个整数

样例输入

10 6 5
2
10
1
5
9

样例输出

1

由于区间的长度已经确定了，那么我们可以通过枚举区间的左端点来确定一个区间。
枚举每一个长度为 k 的区间，然后就判断哪一个区间中损坏的灯的数目最少，取 min 之后输出就可以了
时间复杂度 O（N）

```
#include<iostream>
#include<algorithm>
#define maxn 100201
using namespace std;
int n,k,b;
int vis[maxn];
int sum[maxn];
int ans=0x7fffffff;
int main()
{
    cin>>n>>k>>b;
    for(int i=1;i<=b;i++)
    {
        int x;
        scanf("%d",&x);
        vis[x]=1;
    }
    for(int i=1;i<=n;i++) sum[i]+=sum[i-1]+vis[i];
    for(int i=1;i<=n-k+1;i++) ans=min(ans,sum[i+k-1]-sum[i-1]);
    cout<<ans<<endl;
    return 0;
}
```

5 [8459] S 组石头，剪刀，布

你可能玩过“石头，剪刀，布”，这个游戏在奶牛中同样流行，不过它的名字变成了“蹄子，剪刀，布”。
“蹄子，剪刀，布”和“石头，剪刀，布”的规则十分类似，两只奶牛数到三，然后出一个代表蹄子，剪刀或布的手势。蹄子胜过剪刀，剪刀胜过布，布胜过蹄子。特别地，如果两只奶牛的手势相同，则视为平局。
现在 FJ 和 Bassie 要进行 N 轮对抗。Bassie 已经预测了 FJ 每一轮要出的手势。然而 Bassie 很懒，她最多只想变换一次手势。
现在请你帮 Bassie 求出她最多能赢多少轮。

输入格式第一行输入一个整数 N（$1 \le N \le 10^5$）。
接下来 N 行，每行一个字母，代表 FJ 这一轮出的手势。H 代表蹄子（Hoof），S 代表剪刀（Scissors），P 代表布（Paper）。
输出一个整数，代表 Bassie 在最多变换一次手势的前提下最多赢多少轮。

输入样例：
5
P

P
H
P
S
输出样例：
4

分析：

首先，如果在第 k 次变，前面是出 x 手势，后面出 y 手势，那么就是 k[x]1-x + k[y]x-n。其实 k 数组是可以用前缀和完成。

```
#include<bits/stdc++.h>
using namespace std;
int n,s[100001],p[100001],h[100001],ans;
int main()
{
	cin>>n;
	for(int i=1;i<=n;i++)
	{
		s[i]=s[i-1];//s 的前缀和
		p[i]=p[i-1];//p 的前缀和
		h[i]=h[i-1];//h 的前缀和
		char c;
		cin>>c;
		if(c=='S')
			s[i]++;
		if(c=='P')
			p[i]++;
		if(c=='H')
			h[i]++;
	}
	for(int i=1;i<=n;i++)
		ans=max(ans,max(s[i],max(p[i],h[i]));
	cout<<ans;
	return 0;
}
```

第七讲　双指针*

1 [2709] 最输出每个单词

输入一个行简单英文句子，单词之间用空格分隔，没有缩写形式和其它特殊形式，请输出改句子中的每个单词。 输入一个简单英文句子（长度不超过 500），单词之间用空格分隔，没有缩写形式和其它特殊形式。 输出该句子中每个单词

输入样例
I am a student of Peking University
输出样例
I
am
a
student
of
Peking
University

双指针算法的核心思想是把朴素的暴力写法 o(n2)优化到 o(n)。双指针写法看着两个循环，但两个指针总的移动次数不超过 2n

```
for(int i=0;i<len;i++){
    while(j<i && check(i,j))
        j++;

    //每道题的具体逻辑
}
```

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  char str[1000];
  cin.getline(str,1000);
  int len=strlen(str);
  for(int i=0;i<len;i++){
    int j=i;
    while(j<len &&str[j] != ' ')j++;
    for(int k=i;k<j;k++)
      cout<<str[k];
    cout<<"\n";
    i=j;
  }
```

```
    return 0;
}
```

2 [1107] 最长连续不重复子序列

给定一个长度为 n 的整数序列，请找出最长的不包含重复的数的连续区间，输出它的长度。
第一行包含整数 n。1≤n≤105

第二行包含 n 个整数（均在 0~ 105 范围内），表示整数序列。共一行，包含一个整数，表示最长的不包含重复的数的连续区间的长度
输入样例
5
1 2 2 3 5
输出样例
3
用暴力法：对每个 i(终点） 和 j（起点） 都遍历一遍，对每个 i 和 j 都检查一下[i,j]间的数据是否满足给定的条件。这样的时间复杂度是 O(n2)；数据稍微大点就会超时。
代码如下：

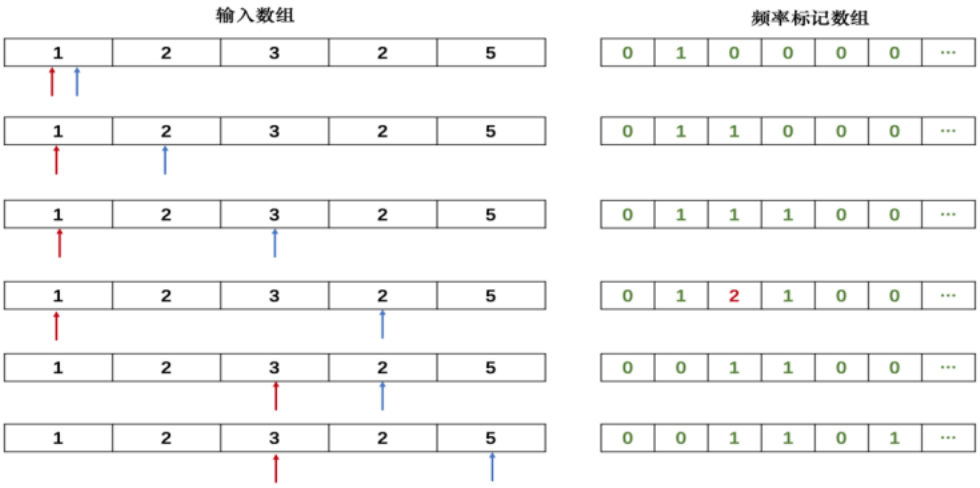
```
for (int i = 0; i < n; i++)
        for (int j = 0; j <= i; j++)
            if (check(v1,j, i) == 0)//检查 i 和 j 之间是否有重复的数字
                res = max(res, i - j + 1);

  //check函数
int check(int  v1[], int l, int r)
{
    for (int i = l+1; i <=r ; i++)
        for (int j = l; j < i; j++)
            if (v1[i] == v1[j])
                return 1;
    return 0;
}
```

仔细考虑暴力法就会发现，暴力法在解题时有很多地方是重复计算了 (i 指针在 j 指针的后面，i 是遍历的整个数组的，j 是遍历 0 到 i 的)：
比如 j = 1，i = 5，此时发现 i，j 是满足题解条件的；那么后面的 j = 2 到 5，就不用计算了，肯定是满足条件的。所以引出了双指针法：当发现 j = 1，i = 5 满足题解条件，那就不用计算 j = 2 到 5，直接计算 j = 1，i = 6，如果不满足条件，那就计算 j = 2，i = 6，然后接着计算。这样就是 i 和 j 指针都是从前移到后，也就是计算 2n 次。时间复杂度是 O(2n)。
双指针方法 1：

```
for (int i = 0, j = 0; i < n; i++)
    {//
        while (j <= i)
        {
            if (check(v1, j, i) == 0)
            {//如果没有重复，i往前走
                res = max(res, i - j + 1);
                break;
            }
            else//有重复j++
                j++;
        }
    }
```

```
int check(int  v1[], int l, int r)
{//找得到重复的数返回1
    for (int i = l + 1; i <= r; i++)
        for (int j = l; j < i; j++)
        {
            if (v1[i] == v1[j])
                return 1;
        }
    return 0;
}
```

上面代码还是超时，因为 check 函数写的不好，循环太多，直接是暴力计算找重复数字的，显然不好。所以引出一个新的 check 方法：寻找是否有重复数字，可以用 hash 表。



1. 遍历数组 a 中的每一个元素 a[i]，对于每一个 i，找到 j 使得双指针[j, i]维护的是以 a[i]结尾的最长连续不重复子序列，长度为 i - j + 1。
2. 对于每一个 i，如何确定 j 的位置：由于[j, i - 1]是前一步得到的最长连续不重复子序列，所以如果[j, i]中有重复元素，一定是 a[i]，因此右移 j 直到 a[i]不重复为止。（由于[j, i - 1]已经是前一步的最优解，此时 j 只可能右移以剔除重复元素 a[i]，不可能左移增加元素，因此，j 具有“单调性”、本题可用双指针降低复杂度）。
3. 用数组 s 记录子序列 a[j~i]中各元素出现次数，遍历过程中对于每一个 i 有四步操作：cin 元素 a[i] -> 将 a[i]出现次数 s[a[i]]加 1 -> 若 a[i]重复则右移 j（s[a[j]]要减 1） -> 确定 j 及更新当前长度 i - j + 1 给 r。

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

```
const int N = 100010;
int a[N], count[N];
int n, ans;
int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]);
    }
    for (int i = 0, j = 0; i < n; i++) {
        count[a[i]] ++;
        while (count[a[i]] > 1) {
        //当 a[i]重复时，先把 a[j]次数减 1，再右移 j。
        //只需考虑 a[j]~a[i]间的元素出现的次数即可
            count[a[j]] --;
            j++;//右移 j 直到 a[i]不重复为止
        }
        ans = max(ans, i - j + 1);
    }
    printf("%d", ans);
    return 0;
}
```

3 [5166] 数组元素的目标和

给定两个升序排序的有序数组 A 和 B，以及一个目标值 x。
数组下标从 0 开始。 请你求出满足 A[i]+B[j]=x 的数对(i,j)。 数据保证有唯一解。
输入第一行包含三个整数 n,m,x，分别表示 A 的长度，B 的长度以及目标值 x。第二行包含 n 个整数，表示数组 A。
第三行包含 m 个整数，表示数组 B。 数组长度不超过 105。
同一数组内元素各不相同。1≤数组元素≤109
输出共一行，包含两个整数 i 和 j。

输入样例
4 5 6
1 2 4 7
3 4 6 8 9

输出样例 1 1

```
4   int a[100001],b[100001];
5   int main()
6   {
7       cin>>n>>m>>s;
8       for(int i=0;i<m;i++)
9           for(int j=0;j<n;j++)
10              if(a[i]+a[j]==s){
11                  cout<<i<<" "j<<endl;
12                  break;
13              }
```

为了减少循环，采用了两个指针来保证能遍历符合要求的所有数；
i 指针从 A 数组的头开始，此时 a[i]最小；
j 指针从 B 数组的尾开始，此时 b[j]最大；

在 i = 0 时，判断 a[i] + b[j] 的值，如果大于 x，j- -；
也就是说 b[j]的值一直在减小；

那么当 a[i] + b[j] 的值第一次小于 x 时，j 指针的左边的数与 a[i]相加一定都小于 x 了；那么我们只能增大 a[i]的值；于是 i ++;

反复循环,如果 a[i] + b[j] 大于 x，就减小 b[j]（相当于将 j 指针左移）反之，就增大 a[i]（相当于 i 指针右移）这样，一定保证了每一个符合要求的两个数都能加一遍，判断是不是和等于 x。

```
#include <iostream>
using namespace std;
const int N = 100010;
int a[N], b[N];
int main()
{
    int n ,m ,x;
    scanf("%d%d%d", &n, &m, &x);
    for (int i = 0; i < n; i ++ ) scanf("%d", &a[i]);
    for (int j = 0; j < m; j ++ ) scanf("%d", &b[j]);
    for (int i = 0, j = m - 1; a[i] < x; i ++ )
    {
        while(a[i] + b[j] > x) j --;
           // 重要的地方在这！
        if (a[i] + b[j] == x)
        {
            printf("%d %d", i, j);
            break;
        }
    }
```

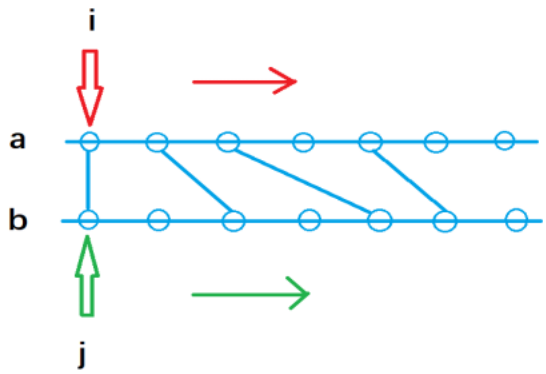
```
    return 0;
}
```

4 [5167] 判断子序列

给定一个长度为 n 的整数序列 a1,a2,…,an 以及一个长度为 m 的整数序列 b1,b2,…,bm。
请你判断 a 序列是否为 b 序列的子序列。
子序列指序列的一部分项按原有次序排列而得的序列，例如序列 {a1,a3,a5} 是序列 {a1,a2,a3,a4,a5} 的一个子序列。
输入样例
4 5 6
1 2 4 7
3 4 6 8 9

输出样例 1 1

1.j 指针用来扫描整个 b 数组，i 指针用来扫描 a 数组。若发现 a[i]==b[j]，则让 i 指针后移一位。
2.整个过程中，j 指针不断后移，而 i 指针只有当匹配成功时才后移一位，若最后若 i==n，则说明匹配成功。
整个过程中 j 指针不断扫描 b 数组并且向后移动，相当于不断给 i 指针所指向的 a 数组创建匹配的机会，只有匹配成功时 i 指针才会向后移动一位，当 i==n 时，说明全部匹配成功。



j指针用来扫描整个b数组，i指针用来扫描a数组。若发现a[i]==b[j]，则让i指针后移一位。
整个过程中，j指针不断后移，而i指针只有当匹配成功时才后移一位，若最后i==n，则说明匹配成功。

```
int i = 0;
for(int j = 0;j < m;j++)
{
  if(i < n && a[i] == b[j]) i++;
}
if(i == n) puts("Yes");
else puts("NO");
```

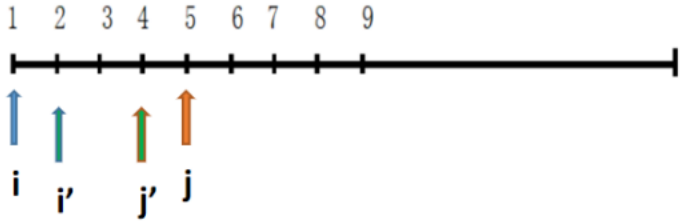
```
#include<iostream>
#include<cstdio>
using namespace std;
const int N=1e5+10;
int a[N],b[N];
int main()
{
    int n,m;
    scanf("%d%d",&n,&m);
    for(int i = 0;i < n; i++) scanf("%d",&a[i]);
    for(int j = 0;j < m; j++) scanf("%d",&b[j]);
```

```
    int i = 0;
    for(int j = 0;j < m; j++)
    {
        if(i < n&&a[i] == b[j])  i++;
    }
    if(i == n) puts("Yes");
    else puts("No");
    return 0;
}
```

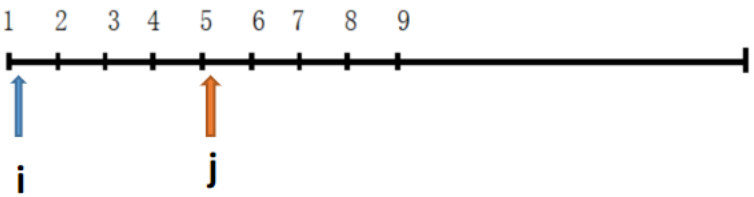
5 [6686] 和为 S 的连续正数序列

题目描述
输入一个非负整数 S，打印出所有和为 S 的连续正数序列（至少含有两个数）。
例如输入 15，由于 1+2+3+4+5=4+5+6=7+8=15，所以结果打印出 3 个连续序列 1~ 5、4~ 6 和 7~ 8。
输入格式
输入一个非负整数 S,0≤S≤100000
输出格式
打印出所有和为 S 的连续正数序列（至少含有两个数）
输入样例 复制
15
输出样例 复制
1 2 3 4 5
4 5 6
7 8

题目要求一个连续区间的值[i,j]，使得的[i,i+1,..,j]的和等于 x



当 i=1 时，可以求出 sum=1+2+3+4+5=15。此时 i++, 去求下一个符合要求的 i 值。这时的 j 呢？
j 可能项左走吗？显然不能！因为如果往左走[i,j]将包含[i’, j’]，因此 j 必然是往右走，符合单调性。



1. 设置两个指针 i 和 j，分别指向连续正数序列的起始和终止

2. 用 s 表示当前连续正数序列的和，即 s=i+(i+1)+…+j
3. 以 i 递增的方式遍历整个序列(1 到 n)，代表查找以 i 开头的时候结尾 j 应该是多少。当 s<sum 说明 j 应该往后移动，当 s=sum 说明满足题意，当 s>sum 说明向后走即可。
4. 注意上述遍历过程中，s=sum 的情况下不需要把 j 往前移动，原因是当进入下一个循环前 s-=i，即(i+1)到 j 的和肯定小于 sum。

```
#include <iostream>
using namespace std;
typedef long long LL;
LL b[500001]={0},l[50001]={0},r[50001]={0};
int main(){
    int n;
    scanf("%d", &n);
    int end=n;
    int idx=1;

    for(int i=1;i<=end;i++)
        b[i]=b[i-1]+i;
    for(int i=1,j=1;i<=end;i++){
       // b[i]=b[i-1]+i;
        while(b[i]-b[j-1]>=n and j<i){
            if(b[i]-b[j-1]==n){
                for(int kk=j;kk<=i;kk++)
                    printf("%d ",kk);
                printf("\n");
                break;
            }
            else j++;
        }
    }


    return 0;
}
```

第八讲 二进制运算*

二进制操作又叫做位运算，二进制有很多用处，常用来提高程序的效率。

在 A 竞赛中，常用二进制进行数据压缩，就是用二进制来表示复杂的状态，然后通过位运算来完成问题的求解，这样能大大提高算法的效率，这样的状态表示经常用来优化 DP 或者是搜索。。。

左移

在很多语言里左移的符号位<<,　　假设有个变量 x=1；

那么 x=x<<1 表示的意思如下：

x:　　0000000000000001（二进制表示）

x<<1:0000000000000010 (也就是把 x 的每一位向左移动 1 位的结果，右边不够的用 0 添上)

此时 x=x<<1 ==2，其实可以看做乘 2

x<<i 表示的就是左移 i 位,可以看做乘 2i

右移

在很多语言里右移的符号位>>　　假设有个变量 x=2；

那么 x=x>>1 表示的意思如下：

x:　　0000000000000010（二进制表示）

x>>1:0000000000000001 (也就是把 x 的每一位向右移动 1 位的结果，左边不够的用 0 添上)

此时 x=x>>1 ==1，其实可以看做除 2

x>>i 表示的就是右移 i 位,可以看做除 i 次 2

或运算

在很多语言里或运算的符号位 |，这个与逻辑运算符 || 是有区别的，而一些语言和伪代码也用 or 表示

加上 x=101 y=010(都是二进制表示)

那么 x|y 表示如下

x:　　101

y:　　010

x|y:　111

其实也就是将 x 和 y 表示成二进制，然后按位做或运算

此时 x|y==7(10 进制表示)

与运算

在很多语言里或运算的符号位 &，这个与逻辑运算符 && 是有区别的，而一些语言和伪代码也用 and 表示

加上 x=101 y=010(都是二进制表示)

那么 x&y 表示如下

x:　　101

y:　　010

x&y:　000

其实也就是将 x 和 y 表示成二进制，然后按位做与运算

此时 x&y==0(10 进制表示)

非运算

在很多语言里或运算的符号位 ~

加上 x=101(都是二进制表示)

那么~x 表示如下

x: 101

~x: 010

其实也就是将 x 表示成二进制，然后按位做取反运算

此时~x==2(10 进制表示)

异或运算

在很多语言里或运算的符号位 ^

加上 x=1010 y=1100 (都是二进制表示)

那么 x^y 表示如下

x: 1010

y: 1100

x^y: 0110

其实也就是将 x 和 y 表示成二进制，然后按位做异或运算，也就是相同为 0，不同为 1

此时 x^y==6(10 进制表示)

获取一个或多个固定位的值

假设 x=1010(10 进制的 10)

我们要获取从左边数第 2 为的值，那么我们可以这样来取

x&(1<<1)也就是

x: 1010

1<<1: 0010

x&(1<<1) 0010

这样我们就可以通过判断 x&(1<<2)是否等于 0 来知道这一位是 0 还是 1 了

当然我们可以用 x&(3<<2)来取得第 3 位和第 4 位

把一个或多个固定位的值置零

假设 x=1010(10 进制的 10)

我们要把从左边数第 2 为的值置零，那么我们可以这样来做

x&(~(1<<1))也就是

x: 1010

~(1<<1): 1101

x&(~(1<<1)) 1000

当然我们可以用 x&(~(3<<2))来把第 3 位和第 4 位置零

把一个或多个固定位的值取反

假设 x=1010(10 进制的 10)

我们要把从左边数第 2 为的值取反，那么我们可以这样来做

x^(~(1<<1))也就是　　如果 x=1000

x:　　1010　　1000

1<<1:　　0010　　0010

x^(1<<1) :　1000　　1010

当然我们可以用 x^(3<<2)来把第 3 位和第 4 位取反

lowbit 运算

lowbit(n)定义为非负整数 n 在二进制表示下“最低位的 1 及其后边所有的 0”构成的数值。

例如　n = 10 的二进制表示为(1010)2,则

lowbit(n) = 2 = (10)2

下 面我们来推导 lowbit(n)的公式

负数的二进制运算为补码运算

-x=~x+1

x&-x=x&(~x+1)

假设 x=10101...100000

~x=01010...011111

+1

100000

x&-x=

假设 x=10101...100000

~x=01010...100000

00000...100000

1 [1213]二进制中 1 的个数

给定一个长度为 n 的数列，请你求出数列中每个数的二进制表示中 1 的个数。

输入:第一行包含整数 n。

第二行包含 n 个整数，表示整个数列。

1≤n≤100000　0≤数列中元素的值≤109

输出共一行，包含 n 个整数，其中的第 i 个数表示数列中的第 i 个数的二进制表示中 1 的个数。

样例输入

5

1 2 3 4 5

样例输出

1 1 2 1 2

O(nlogn)

使用 lowbit 操作，每次 lowbit 操作截取一个数字最后一个 1 后面的所有位，每次减去 lowbit 得到的数

字，直到数字减到 0，就得到了最终 1 的个数.

```
#include<bits/stdc++.h>
using namespace std;
int n,a[100001];
int lowbit(int m){
    return -m&m;
}
int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
    }
    for(int i=1;i<=n;i++){
        int x=a[i],ans=0;
        while(x){
            x-=lowbit(x);
            ans++;
        }
        cout<<ans<<" ";
    }
    return 0;
}
```

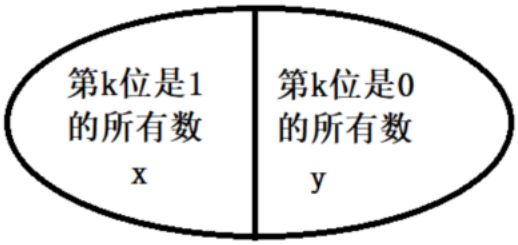
2 [6687] 数组中只出现一次的两个数字

一个整型数组里除了两个数字之外，其他的数字都出现了两次。
请写程序找出这两个只出现一次的数字。
你可以假设这两个数字一定存在。
数组长度 1<=n<=1e5
1<=a[i]<=1e9

输入样例 5
1 2 2 3 2
输出样例
1 3

哈希表
时间复杂度 O(n)，空间复杂度 O(n2)

异或满足交换律,如 x1^x2^x3=x1^x3^x2
1. 第一步异或，相同的数其实都抵消了，剩下两个不同的数。这两个数异或结果肯定有某一位为 1，不然都是 0 的话就是相同数。




```cpp
#include <bits/stdc++.h>
using namespace std;

int main(){
    int n,x;
    vector<int> ans;
    map<int, int> hash;
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>x;
        hash[x]++;
    }
    for (auto & it : hash)
        if (it.second == 1)
            ans.push_back(it.first);
    for(int i=0;i<ans.size();i++)
        cout<<ans[i]<<" ";
    return 0;
}
```

3 [2819] 背包问题

内存限制：128 MB 时间限制：1 S 标准输入输出
题目类型：传统评测方式：文本比较上传者：quanxing
提交：1157 通过：649
提交提交记录统计讨论版
编辑题目测试数据
题目描述
一个旅行者有一个最多能装 M 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...,Wn 它们的价值分别为 C1,C2,...,CnC1,C2,...,Cn，求旅行者能获得最大总价值。

视频讲解：https://v.youku.com/v_show/id_XNDgyMTY1NDQ3Ng==.html
输入格式
第一行：两个整数，M(背包容量，M≤200)和 N(物品数量，N≤30)；
第 2..N+1 行：每行二个整数 Wi，Ci，表示每个物品的重量和价值。
输出格式
仅一行，一个数，表示最大总价值。

输入样例 复制

10 4
2 1
3 3
4 5
7 9

输出样例 复制

12

```
#include<bits/stdc++.h>
using namespace std;
int w[105],v[105],n,W;

int main(){

    scanf("%d%d", &W, &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d", &w[i], &v[i]);
    int ans = 0;
    for (int i = 0; i < (1 << n); i++)
    {
        int a = 0, b = 0;
        for (int j = 0; j < n; j++)
        {
            if (i&(1 << j)) a += v[j], b += w[j];
        }
        if (b <= W) ans = max(ans, a);

    }
    printf("%d\n", ans);


    return 0;
}
```

第九讲 字符与大数加减

重点：

1. 掌握字符数组和数值数组的转换。
2. 理解大数加减法。
3. 了解大数乘除法。

在 32 位机器里，有符号整数(int)的取值范围是-2147483648～+2147483647，无符号整数(unsigned int) 的取值范围是 0～4294967295。超过这个范围的数据可以用浮点型(double)来表示，如 50!。但用浮点数来表示整数通常不便于整数的运算，比如整数除法跟浮点数除法含义不一样，浮点数也无法实现取余运算等等。

另外，超过浮点数取值范围的数据，比如一个 1000 位的整数，无法用常规方法来处理。这些精度很高的数据通常称为高精度数，或称为大数。高精度数的运算只能用本章介绍的高精度数计算方法来处理。

高精度计算中需要处理好以下几个问题：

(1)数据的接收方法和存贮方法

数据的接收和存贮：当输入的数很长时，可采用字符串方式输入，这样可输入数字很长的数，利用字符串函数和操作运算，将每一位数取出，存入数组中。

```
int init(int a[])          //传入一个数组
{
    char s[1000];//string s;
    int len;
    cin>>s;
    len=strlen(s);//s.length(); //用 len 计算字符串 s 的位数
    for(int i=1;i<=len;i++)
    a[i]=s[len -i]-'0'; //将数串 s 转换为数组 a，并倒序存储
}
```

(2) 高精度数位数的确定

位数的确定：接收时往往是用字符串的，所以它的位数就等于字符串的长度。

(3) 进位,借位处理

加法进位：c[i]=a[i]+b[i];

if (c[i]>=10) { c[i]%=10; c[i+1]++; }

减法借位：if (a[i]<b[i]) { a[i+1]--; a[i]+=10; }

c[i]=a[i]-b[i];

(4) 进位,借位处理

乘法进位：c[i+j-1]= a[i]*b[j] + x + c[i+j-1];

x = c[i+j-1]/10;

c[i+j-1] %= 10;

(5) 商和余数的求法

商和余数处理：视被除数和除数的位数情况进行处理.

例：3290: 输出大数的字符

题目描述:输出一个大数如，9385398593859285023589325829435359O, 输出第 n 位的数（从右往左数）。

输入

938539859385928502358932582943535 90
10
输出
8

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

int main(){
    int n,a[200],len,i;
    char str[200];
    cin>>str>>n;
    len=strlen(str);
    for(i=1;i<=n;i++)
       a[i]=str[__________len-i]-'0';
    cout<<a[n];
    return 0;
}
```

10.1 [2727] 高精度加法

输入两个正整数，求它们的和。

在 C++语言中任何数据类型都有一定的表示范围。而当两个被加数很大时，上述算法显然不能求出精确解，因此我们需要寻求另外一种方法。在读小学时，我们做加法都采用竖式方法，如图 1。 这样，我们方便写出两个整数相加的算法。

因此，算法描述如下：

```
int c[100];
void add(int a[],int b[])      //a,b,c 都为数组，分别存储被加数、加数、结果
{
     int   i=1,x=0;                                    //x 是进位
     while ((i<=a 数组长度)||(i<=b 数组的长度))
  {
        c[i]=a[i]+b[i]+x;        //第 i 位相加并加上次的进位
        x=c[i]/10;                    //向高位进位
        c[i]%=10;                          //存储第 i 位的值
        i++;                                    //位置下标变量
  }
}
```

通常，读入的两个整数用可用字符串来存储，程序设计如下：

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

```
using namespace std;
int main()
{
   char a1[100],b1[100];
        int a[100],b[100],c[100],lena,lenb,lenc,i,x;
        memset(a,0,sizeof(a));
        memset(b,0,sizeof(b));
        memset(c,0,sizeof(c));

   gets(a1);
   gets(b1)                  //输入加数与被加数
        lena=strlen(a1);
        lenb=strlen(b1);
        for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48;
                  //加数放入 a 数组
        for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48;
                    //加数放入 b 数组
        lenc =1;
        x=0;
   while (lenc <=lena||lenc <=lenb)
   {
              c[lenc]=a[lenc]+b[lenc]+x;        //两数相加
              x=c[lenc]/10;
              c[lenc]%=10;
              lenc++;
   }
   c[lenc]=x;
   if (c[lenc]==0)
        lenc--;                                          //处理最高进位
   for (i=lenc;i>=1;i--)
   cout<<c[i];                                           //输出结果
   cout<<endl;
   return 0;
}
```

10.2 [2728]高精度减法

输入两个正整数，求它们的差。

类似加法，可以用竖式求减法。在做减法运算时，需要注意的是：被减数必须比减数大，同时需要处理借位。高精度减法的参考程序：

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

```
using namespace std;
int main()
{
   int a[256],b[256],c[256],lena,lenb,lenc,i;
   char n[256],n1[256],n2[256];
   memset(a,0,sizeof(a));
   memset(b,0,sizeof(b));
   memset(c,0,sizeof(c));

   printf("Input minuend:");       gets(n1);     //输入被减数
   printf("Input subtrahend:"); gets(n2);     //输入减数
   if (strlen(n1)<strlen(n2)||(strlen(n1)==strlen(n2)&&strcmp(n1,n2)<0))
      //strcmp()为字符串比较函数，当 n1==n2，返回 0;n1>n2 时，返回正整数；n1<n2 时，返回负整数
   {    //处理被减数和减数，交换被减数和减数
         strcpy(n,n1);        //将 n1 数组的值完全赋值给 n 数组
         strcpy(n1,n2);
         strcpy(n2,n);
         cout<<"-";       //交换了减数和被减数，结果为负数
   }
   lena=strlen(n1); lenb=strlen(n2);
   for (i=0;i<=lena-1;i++) a[lena-i]=int(n1[i]-'0');   //被减数放入 a 数组
   for (i=0;i<=lenb-1;i++) b[lenb-i]=int(n2[i]-'0');   //减数放入 b 数组
   i=1;
   while (i<=lena||i<=lenb)
   {
        if (a[i]<b[i])
        {
             a[i]+=10;                    //不够减，那么向高位借 1 当 10
             a[i+1]--;
        }
        c[i]=a[i]-b[i];                              //对应位相减
        i++;
   }
   lenc=i;
   while ((c[lenc]==0)&&(lenc>1)) lenc--;     //最高位的 0 不输出
   for (i=lenc;i>=1;i--) cout<<c[i];                  //输出结果
   cout<<endl;
   return 0;
}
```

10.3 [7093] ISBN 号码

每一本正式出版的图书都有一个 ISBN 号码与之对应，ISBN 码包括 9 位数字、1 位识别码和 3 位分隔符，其规定格式如“x-xxx-xxxxx-x”，其中符号“-”是分隔符（键盘上的减号），最后一位是识别码，例如 0-670-82162-4

就是一个标准的 ISBN 码。ISBN 码的首位数字表示书籍的出版语言，例如 0 代表英语；第一个分隔符“-”之后的三位数字代表出版社，例如 670 代表维京出版社；第二个分隔之后的五位数字代表该书在出版社的编号；最后一位为识别码。

识别码的计算方法如下：

首位数字乘以 1 加上次位数字乘以 2…以此类推，用所得的结果 mod 11，所得的余数即为识别码，如果余数为 10，则识别码为大写字母 X。例如 ISBN 号码 0-670-82162-4 中的识别码 4 是这样得到的：对 067082162 这 9 个数字，从左至右，分别乘以 1，2，…，9，再求和，即 0×1+6×2+…+2×9=158，然后取 158 mod 11 的结果 4 作为识别码。

你的任务是编写程序判断输入的 ISBN 号码中识别码是否正确，如果正确，则仅输出“Right”；如果错误，则输出你认为是正确的 ISBN 号码。

输入：

每组输入数据只有一行，是一个字符序列，表示一本书的 ISBN 号码（保证输入符合 ISBN 号码的格式要求）。

输出：

每组输出共一行，假如输入的 ISBN 号码的识别码正确，那么输出“Right”，否则，按照规定的格式，输出正确的 ISBN 号码（包括分隔符“-”）。

样例输入：0-670-82162-4

样例输出：Right

这道题只要找到相应字符的位置就 ok，注意当取膜后为 10，那么识别码是 X

```
#include <iostream>
#include <cstring>
using namespace std;
char s[100];
int main()
{
    while(cin>>s)//多组读入
    {
        //获得前 9 位相应权值乘积和
        int sum = 0;
        sum = (s[0] - '0') * 1;
        for(int i = 2; i <= 4; i ++)
            sum += (s[i] - '0') * i;
        for(int i = 6; i <= 10; i ++)
            sum += (s[i] - '0') * (i - 1);
        sum %= 11;
        //对于识别码判断
        if(sum == s[12] - '0' || (sum == 10 && s[12] == 'X'))
            printf("Right\n");
        else
        {
            //如不符合则前 12 位照写，识别码另外处理
            for(int i = 0; i <= 11; i ++)
                cout << s[i];
            if(sum == 10)
```

```
                cout << "X" << endl;
            else
                cout << sum << endl;
        }
    }
    return 0;
}
```

10.4 [2962] 统计单词数

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：

数据的第 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。（1≤单词长度≤10，1≤文章长度≤1,000,000）

输出：

每组输出只有一行，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：

```
To
to be or not to be is a question
```

样例输出：

```
2 0
```

首先预处理，把所有字母统一改为小写字母。然后暴力从文本中找出每一个单词。从 i=0 开始扫描，遇到第一个字母后往后找，直到找到连续的最后一个字母。这一段即是一个单词。然后把这个单词与给定的单词暴力匹配一下即可。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e6+10;
char txt[maxn],t[15];//文本串和待匹配的单词
bool check(int l,int r,int len)//暴力匹配
{
    if(r-l+1!=len)//剪枝，若长度不同则不匹配
        return 0;
    for(int i=l,j=0;j<len;j++,i++)//判断每个字符是否匹配
        if(txt[i]!=t[j])
```

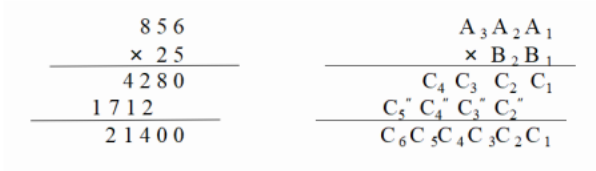
```
                return 0;
        return 1;
}
void Init(char a[],int len)//所有字母统一改为小写
{
        for(int i=0;i<len;i++)
                if(a[i]>='A'&&a[i]<='Z')
                        a[i]='a'+a[i]-'A';
}
int main()
{
        scanf("%s",t);
        getchar();
        gets(txt);//读入整行
        int len=strlen(txt);
        int Len=strlen(t);
        Init(txt,len);//改为小写
        Init(t,Len);//同上
        int num=0,ans=-1;
        for(int i=0;i<len;i++)
        {
                if(txt[i]>='a'&&txt[i]<='z')//获取一个单词
                {
                        int r=i;
                        while(r+1<len&&txt[r+1]>='a'&&txt[r+1]<='z')
                                r++;
                        if(check(i,r,Len))//判断是否匹配
                        {
                            num++;//匹配数++
                            if(ans==-1)//若是第一次匹配，记录一下位置
                                    ans=i;
                        }
                        i=r;
                }
        }
        if(ans!=-1)
                printf("%d %d\n",num,ans);
        else
                printf("%d\n",ans);
        return 0;
}
```

第十讲 大数乘除

1 [2733]高精度乘法

类似加法，可以用竖式求乘法。在做乘法运算时，同样也有进位，同时对每一位进行乘法运算时，必须进行错位相加，如图所示。

分析 c 数组下标的变化规律，可以写出如下关系式：$c_i = c_i' + c_i'' + \cdots$ 由此可见，c_i 跟 a[i]*b[j]乘积有关，跟上次的进位有关，还跟原 c_i 的值有关，分析下标规律，有 c[i+j-1]= a[i]*b[j]+ x + c[i+j-1]; x=c[i+j-1]/10 ; c[i+j-1]%=10;

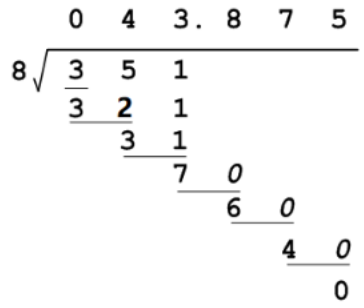


高精度乘法的参考程序：

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    char a1[100],b1[100];
    int a[100],b[100],c[100],lena,lenb,lenc,i,j,x;
    memset(a,0,sizeof(a));
    memset(b,0,sizeof(b));
    memset(c,0,sizeof(c));
    gets(a1);gets(b1);
    lena=strlen(a1);lenb=strlen(b1);
    for (i=0;i<=lena-1;i++) a[lena-i]=a1[i]-48;
    for (i=0;i<=lenb-1;i++) b[lenb-i]=b1[i]-48;
    for (i=1;i<=lena;i++)
    {
        x=0;                                                    //用于存放进位
        for (j=1;j<=lenb;j++)                          //对乘数的每一位进行处理
        {
          c[i+j-1]=a[i]*b[j]+x+c[i+j-1]; //当前乘积+上次乘积进位+原数
          x=c[i+j-1]/10;
          c[i+j-1] %= 10;
        }
        c[i+lenb]=x;                                          //进位
    }
    lenc=lena+lenb;
    while (c[lenc]==0&&lenc>1)          //删除前导 0
```

```
        lenc--;
    for (i=lenc;i>=1;i--)
        cout<<c[i];
    cout<<endl;
    return 0;
}
```

2 [2734]高精度除低精度



除法运算过程(除数只有1位数)

大数除法，是四则运算里面最难的一种。不同于一般的模拟，除法操作步数模仿手工除法，而是利用减法操作实现的。如图所示，为普通除法处理过程。

1) 先将被除数"351"最高位除以 8，得到的商为 0，余数为 3。

2) 把余数和被除数"351"的次高位组合，为"35"，除以 8，得到的商为 4，余数为 3。

3) 把余数和被除数"351"的最后一位组合，为"31"，除以 8，得到的商为 3，余数为 7，不为 0，所以还没有除尽，要补 0，图中所有补 0 均用斜体标明。补 0 前的商为整数部分，补 0 后的商为小数部分。

4) 补 0 后为"70"，除以 8，得到的商为 8，余数为 6，再补 0。

5) 补 0 后为"60"，除以 8，得到的商为 7，余数为 4，再补 0。

6) 补 0 后为"40"，除以 8，得到的商为 5，余数为 0。

至此，整个除法运算完毕，得到的商为 43.875

做除法时，每一次上商的值都在 0 ～ 9 ，每次求得的余数连接以后的若干位得到新的被除数，继续做除法。因此，在做高精度除法时，要涉及到乘法运算和减法运算，还有移位处理。当然，为了程序简洁，可以避免高精度除法，用 0～9 次循环减法取代得到商的值。这里，我们讨论一下高精度数除以单精度数的结果，采取的方法是按位相除法。

```
#include<iostream>
#include<cstring>
#include<cstdio>
using namespace std;
int main()
{
    char a1[100],c1[100];
    int a[100],c[100],lena,i,x=0,lenc,b;
    memset(a,0,sizeof(a));
    memset(c,0,sizeof(c));
    gets(a1);//a 为大数，被除数
    cin>>b;//b 为除数，非大数
    lena=strlen(a1);
    for (i=0;i<=lena-1;i++)
        a[i+1]=a1[i]-48;
    for (i=1;i<=lena;i++)   //按位相除
    {
```

```
        c[i]=(x*10+a[i])/b;//把余数和被除数"351"的次高位组合
        x=(x*10+a[i])%b;//x 表示当前余数 如 x=35%8=3
    }
    lenc=1;
    while (c[lenc]==0&&lenc<lena)
        lenc++;     //删除前导 0
    for (i=lenc;i<=lena;i++)
        cout<<c[i];
    cout<<endl;
    return 0;}
```

3 [3032]: 回文数

题目描述

若一个数（首位不为零）从左向右读与从右向左读都一样，我们就将其称之为回文数。

例如：给定一个 10 进制数 56，将 56 加 56（即把 56 从右向左读），得到 121 是一个回文数。

又如：对于 10 进制数 87：

STEP1：87+78 = 165　　STEP2：165+561 = 726

STEP3：726+627 = 1353　　STEP4：1353+3531 = 4884

在这里的一步是指进行了一次 N 进制的加法，上例最少用了 4 步得到回文数 4884。

输入每个测试文件只包含一组测试数据，每组输入一个 N（2<=N<=10，N=16）进制数 M，每组的第一行输入 N，第二行输入 M。

输出对于每组输入数据，输出最少经过几步可以得到回文数。如果在 30 步以内（包含 30 步）不可能得到回文数，则输出"Impossible!"。

样例输入

9

87

样例输出

STEP=6

[算法分析]

N 进制运算

1、当前位规范由%10 改为% n

2、进位处理由/10 改为/n

3、其他运算规则不变

```
#include<iostream>
#include<cstring>
using namespace std;
int n,a[101],b[101],i;
int ans,len; //ans 表示步数，len 表示长度
void init(int a[]) //将数串 s 转化为整数数组 a
```

```
{
    string s;
    cin>>n>>s; //读入字符串 s
    memset(a,0,sizeof(a)); //数组 a 清 0
    len=s.length();     //用 len 计算字符串 s 的位数
    for(i=1;i<=len;i++)
        if(s[len-i]>='0'&&s[len-i]<='9')   a[i]=s[len-i]-'0';
        else a[i]=s[len-i]-'A'+10;//如果十六进制就有 A~F
}
bool check(int a[])   //判别整数数组 a 是否为回文数
{
    for(i=1;i<=len;i++)
        if(a[i]!=a[len-i+1])return false;
    return true;
}
void   jia(int a[])   //整数数组 a 与其反序数 b 进行 n 进制加法运算
{
    for(int i=1;i<=len;i++)b[i]=a[len-i+1]; //反序数 b
    for(int i=1;i<=len;i++) a[i]+=b[i];   //逐位相加
    for(int i=1;i<=len;i++)        //处理进位
    {    a[i+1]+=a[i]/n;
        a[i]%=n;
    }
    if(a[len+1]>0) len++;      //修正新的 a 的位数（a+b 最多只能的一个进位）
}
int main()
{    init(a);
    if(check(a)){cout<<0<<endl;return 0;}
    ans=0;          //步数初始化为 0
    while(ans++<=30)
    {          jia(a);
        if(check(a)){cout<<"STEP="<<ans<<endl;return 0;}
    }
    cout<<"Impossible!";    //输出无解信息
    return 0;
}
```

4 [2729] 计算 2 的 N 次方

题目描述
任意给定一个正整数 N(N≤100)，计算 2 的 n 次方的值。

输入：输入一个正整数 N。
输出：输出 2 的 N 次方的值。

样例输入：5
样例输出：32

```
#include <iostream>
#include <cmath>
#include <cstdio>
using namespace std;
int main()
{
	int n;
	while(cin>>n)
	{
		int a[5000]={0};
		a[0]=1;
		int num=1,k;
		for(int i=0;i<n;i++)
		{
			for(int j=0;j<num;j++)
				a[j]=a[j]*2;
			for( k=0;k<num;k++)
			{
				if(a[k]>=10)
				{
					a[k+1]=a[k+1]+a[k]/10;
					a[k]=a[k]%10;
				}
			}
			if(a[k]>0)
				num++;
		}
		for(int i=num-1;i>=0;i--)
			cout<<a[i];
		cout<<endl;
	}
	return 0;
}


#include<iostream>
#include<cstdio>
using namespace std;
int main()
{
	double m=1,n;
	cin>>n;
	for(int i=1;i<=n;i++)
		m*=2;
	printf("%.0lf",m);
	return 0;
```

5 [7049] 字符串编辑

题目描述
从键盘输入一个字符串（长度<=40 个字符），并以字符 '.'结束。
例如：'This is a book.' 现对该字符串进行编辑，编辑功能有：
D：删除一个字符，命令的方式为：
D a 其中 a 为被删除的字符
例如：D s 表示删除字符 's'，若字符串中有多个 's'，则删除第一次出现的。
如上例中删除的结果为： 'Thi is a book.'

I：插入一个字符，命令的格式为：
I a1 a2 其中 a1 表示插入到指定字符前面，a2 表示将要插入的字符。
例如：I s d 表示在指定字符 's' 的前面插入字符 'd'，若原串中有多个 's'，则插入在最后一个字符的前面。
如上例中：
原串：'This is a book.'
插入后：'This ids a book.'

R：替换一个字符，命令格式为：
R a1 a2 其中 a1 为被替换的字符，a2 为替换的字符，若在原串中有多个 a1 则应全部替换。
例如： 原串： 'This is a book.'
输入命令：R o e
替换后的字符串为：'This is a beek.'

在编辑过程中，若出现被改的字符不存在时，则给出提示信息"Not exist"。

输入
每个测试文件只包含一组测试数据，每组输入数据包含两行：
第一行，输入一个字符串，表示原串；
第二行，输入一个字符串，表示命令。

输出
对于每组输入数据，输出编辑后的字符串，如果被改的字符不存在，则输出"Not exist"（引号不输出）。

样例输入
This is a book.
D s
样例输出
Thi is a book.

```
include<cstdio>#include<cstring>using namespace std;char s[100];int main(){
    int i,j,k,len;
    char a,b,c;
    bool flag;

    gets(s);
    a=getchar();
    len=strlen(s);
    for(i=len-1;i>=0;i--)
       if(s[i]=='.'){len=i+1;break;}

    if(a=='D')
       {
          getchar(),b=getchar();
          flag=0;
          for(i=0;i<len;i++)
             {
                if(s[i]!=b || flag)printf("%c",s[i]);
                else flag=1;
             }
       }
    if(a=='I')
        {
           getchar(),b=getchar();
           getchar(),c=getchar();
           j=100;
           for(i=len-1;i>=0;i--)
              if(s[i]==b){j=i;break;}
           for(i=0;i<len;i++)
              {
                 if(i==j)printf("%c",c);
                 printf("%c",s[i]);
              }
        }
     if(a=='R')
       {
          getchar(),b=getchar();
          getchar(),c=getchar();
          for(i=0;i<len;i++)
             if(s[i]==b)printf("%c",c);
             else printf("%c",s[i]);
       }
     printf("\n");
     return 0;
```

```
}

#include<iostream>#include<cstdio>#include<string>#include<cstring>using namespace std;
int main(){
   int i,j,k,len;
   string s,c1,c2,c3;
   size_t p;

   getline(cin,s),len=s.length(),c3=s[len];
   for(i=0;i<len;i++)if(s[i]=='.')break;
   s.erase(i+1),len=s.length();

   cin>>c1;
   switch(c1[0])
     {
        case 'D':cin>>c2;
                 p=s.find_first_of(c2);
                 if(p<len)s.erase(p,1);
                 break;
        case 'I':cin>>c2>>c3;
                 p=s.find_last_of(c2);
                 s.insert(p,c3);
                 break;
        case 'R':cin>>c2>>c3;
                 p=s.find_first_of(c2);
                 while(p!=string::npos)
                   {
                     s[p]=c3[0];
                     p=s.find_first_of(c2,p+1);
                   }
                 break;
     }
   cout<<s<<endl;
   return 0;
}
```

课堂作业列表

[2727] 高精度加法
[2728]高精度减法
[2733]高精度乘法
[2734]高精度除低精度
[2858]高精度除高精度

[3032] 回文数
[2729] 计算 2 的 N 次方

大数习题

1. 执行下面程序，输出是 _____

```
#include<iostream>
using namespace std;
int x = 5, y = 7;
void swap ( )
{       int z ;
        z = x ;   x = y ;   y = z ;
}
int main( )
{
        int x = 3, y = 8;
        swap ( ) ;
        printf ( " %d , %d \n", x , y ) ;
        return 0 ;
}
```

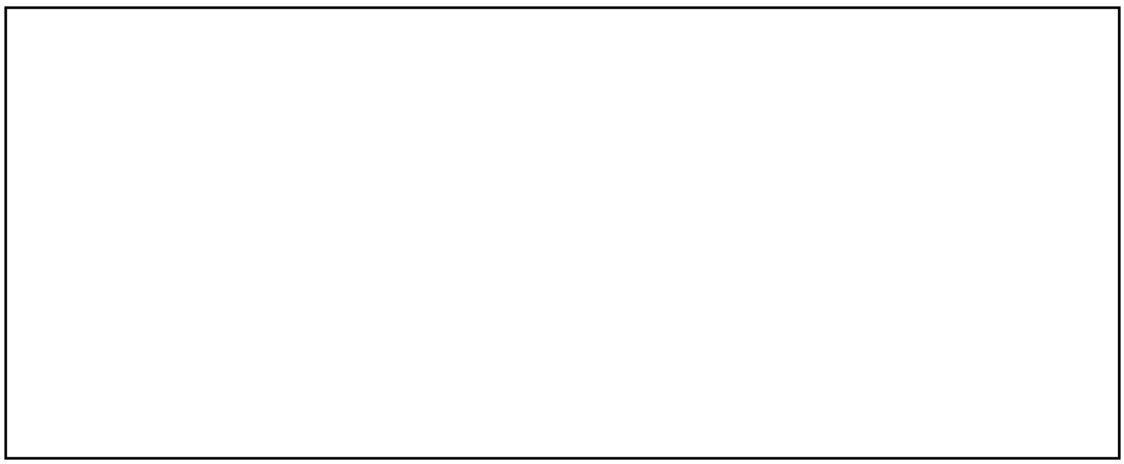
2. 以下程序运行后的输出结果是____。

```
int main( )
{
    char a[8],temp; int j,k;
    for(j=0;j<7;j++) a[j]='a'+j;   a[7]='\0';
    for(j=0;j<3;j++)
    {
        temp=a[6];
        for(k=6;k>0;k--) a[k]=a[k-1];
        a[0]=temp;
        printf("%s\n",a);
    }
}
```

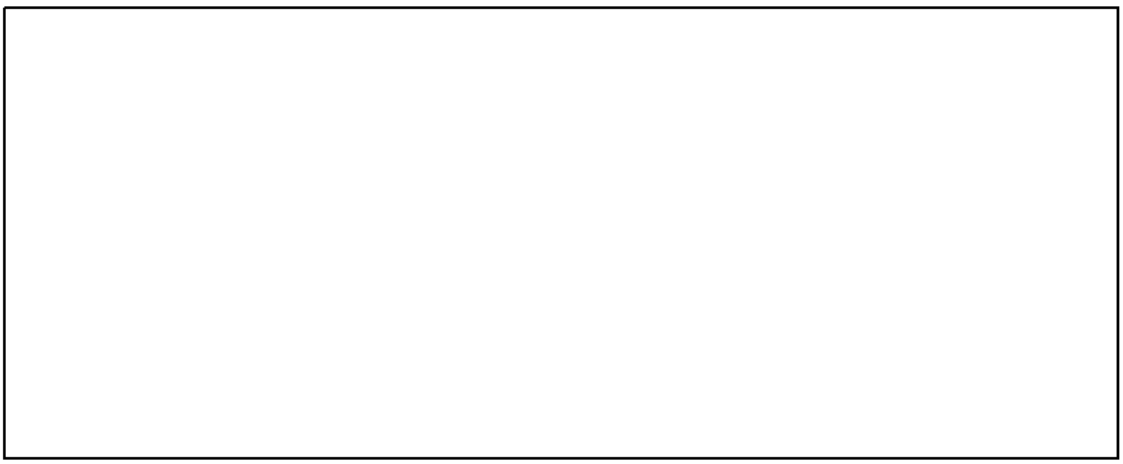
gabcdef fgabcde efgabcd

3. 加法的核心代码

4 减法的核心代码



5 乘法的核心代码



第十一讲 二分

使用二分查找是为了提高效率。例如在含有 10^8 个元素的非下降序列的数组（要求有序）中查找某个特定值，用从头到尾遍历的方法，其时间复杂度为 0(n)。而二分查找，每次循环将 key 值与区间的 mid 点的值比较，mid 点的值刚好等于 key 值，就返回下标，否则更新左边界或者是右边界来使区间减半，这样每次待查区间就会不断减半，最终在某一次，区间会缩小到刚好 mid 点的值等于 key 值，或者区间的 left 端，或者区间的 right 端点就刚好指向 key 值。二分的次数就是需要循环判断的次数，最多判断次数，也就是最坏复杂度，也就是 O(log2n)。

二分查找的解决步骤如下：
第一个步骤： 确定**初始区间**表示法；
第二个步骤: 根据第一步所选的区间表示法，写出对应的**循环条件；**
第三个步骤: 在第一，二步正确的情况下，写对应的**边界更新**语句。

上下界[a, b]确定、函数在[a, b]内单调， 适合用二分。

1 [1108] 查询元素

有一正整数序列 a1,a2,a3,...an, 其中 n<=105 , ai<=109 有 m 个查询（m<=105）,每次查询 m 是否在数组 a 中如果存在输出 1，否则输出 0.

样例输入

```
10 4
16 63 20 30 70 76 73 79 23 40
20
22
40
45
```

样例输出

```
1
0
1
0
```

顺序查找，执行 m 次。时间复杂度为 0(m*n)， 二分把一个区间一分为二，一个区间满足要求，一个区间不满足。二分就是寻找这个要求的边界。

1.对原数组排序，时间复杂度为 0(n*logn)

2. 查找 x 时限于中间的元素比较，等于则结束。中间数小于 x 则在右侧查找（调整左端点）；中间数大于 x 则在左侧查找(调整右端点)，重复上述方法。时间复杂度为 0(logn)。执行 m 次，时间复杂度为 m*logn。

总的时间复杂度为 0(n*logn+m*logn)

```
#include <bits/stdc++.h>
using namespace std;
```

```
int a[100005];

int bitSearch(int l, int r, int x){
    int mid;
    while(l<=r){
        mid=(l+r)/2;
        if(x<a[mid])
            r=mid-1;
        else if(x>a[mid])
            l=mid+1;
        else
            return 1;
    }
    return 0;
}
int main()
{
    int n,m,x;
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        scanf("%d",&a[i]);
    sort(a+1,a+1+n);
    while(m--){
        scanf("%d",&x);
        printf("%d\n",bitSearch(1,n,x));
    }
}
```

2 [1154] 查询元素 2

给出 n 个有序的正整数 a1,a2,a3,...an，其中 n<=105 ， ai<=109 有 m 个查询（m<=105）,每次查询输出刚好大于等于 x 的位置，如果不存在大于 x 的元素输出 n+1。

输入

```
10 4
16 20 20 40 40 63 70 73 76 79
20
100
65
5
```

输出

```
2
11
7
1
```

a 数组有序，可以利用二分法实现查找，注意不存在大于等于 x 的情况可以单独特判。
a[mid]<x 时，mid 不可能为答案，所以 l=mid+1;
否则,mid 可能为答案，所以 r=mid;
此种情况注意 mid=(l+r)>>1;while（l<r）,l,r 点为答案

分析：刚好大于等于 x 的位置(x>=k)

```
int check(int a[],int x){
    if(a[n]<x)return n+1;
    int l=1,r=n;
    while(l<r){
        int mid=(l+r)>>1;
        if(a[mid]<x)//mid不可能是答案
            l=mid+1;//左侧不可能是答案
        else//>= 右侧可能是答案
            r=mid;
    }
    return l;
}
```

3 [1036] 数的范围

给定一个按照升序排列的长度为 n 的整数数组，以及 q 个查询。对于每个查询，返回一个元素 k 的起始位置和终止位置（位置从 0 开始计数）。如果数组中不存在该元素，则返回 -1 -1。

输入第一行包含整数 n 和 q，表示数组长度和询问个数。

第二行包含 n 个整数，表示完整数组。

接下来 q 行，每行包含一个整数 k，表示一个询问元素。

1≤n≤1000001 ≤q≤10000 1≤k≤10000

输出共 q 行，每行包含两个整数，表示所求元素的起始位置和终止位置。

如果数组中不存在该元素，则返回 -1 -1。

输入样例

6 3

1 2 2 3 3 4

3

4

5

输出样例 3 4

5 5

-1 -1

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
int a[maxn];
```

```
#include<bits/stdc++.h>
using namespace std;
const int maxn=1e5+10;
int a[maxn];
```

```
int n,q,x;
int main(){
    cin>>n>>q;
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);
    while(q--){
        int l=0,r=n-1;
        scanf("%d",&x);
        while(l<r){//a[mid]>=x 地最大值
            int mid=(l+r)/2;
            if(a[mid]>=x){
                r=mid;
            }
            else{
                l=mid+1;
            }
        }
        if(a[l]!=x){
            printf("-1 -1\n");
        }
        else{
            printf("%d ",l);
            int l=0,r=n-1;
            while(l<r){
                int mid=(l+r+1)/2;
                if(a[mid]<=x){
                    l=mid;
                }
                else{
                    r=mid-1;
                }
            }
            printf("%d\n",l);
        }

    }
    return 0;
}
```

```
int n,q,x;
int main(){
    cin>>n>>q;
    for(int i=0;i<n;i++)
        scanf("%d",&a[i]);
    while(q--){
        int l=0,r=n-1,ans;
        scanf("%d",&x);
        while(l<=r){//a[mid]>=x 地最大值
            int mid=(l+r)/2;
            if(a[mid]>=x){
                ans=mid,r=mid-1;
            }
            else{
                l=mid+1;
            }
        }
        if(a[ans]!=x){
            printf("-1 -1\n");
        }
        else{
            printf("%d ",l);
            int l=0,r=n-1;
            while(l<=r){
                int mid=(l+r)/2;
                if(a[mid]<=x){
                    ans=mid,l=mid+1;
                }
                else{
                    r=mid-1;
                }
            }
            printf("%d\n",ans);
        }

    }
    return 0;
}
```

4 [3630] 愤怒的牛

农夫 John 建造了拥有 N (2 <= N <= 100,000)个牛舍的小屋，这些牛舍排在一条直线上。这些牛舍依次编号为 x1,...,xN (0 <= xi <= 1,000,000,000). 但是，John 的 C (2 <= C <= N)头牛们并不喜欢这种布局，而且几头牛放

在一个牛舍里，它们会相互攻击。为了不让牛互相伤害，John 决定自己给牛分配舍，使任意两头牛之间的最小距离尽可能的大，那么，这个最大的最小距离是什么呢
输入格式

第一行：空格分隔的两个整数 N 和 C
第二行---第 N+1 行：i+1 行指出了 xi 的位置
输出格式
一个整数，最大的最小值
输入样例
5 3
1
2
8
4
9
输出样例
3

（1）暴力法。从小到大枚举最小距离的值 dis，然后检查，如果发现有一次不行，那么上次枚举的就是最大值。如何检查呢？用贪心法：第一头牛放在 x1，第二头牛放在 xj≥x1+dis 的点 xi,第三头牛放在 xk≥xj+dis 的点 xk，等等，如果在当前最小距离下，不能放 c 条牛，那么这个 dis 就不可取。复杂度 O(nc)。
（2）二分。分析从小到大检查 dis 的过程，发现可以用二分的方法找这个 dis。这个 dis 符合二分法：它有上下边界、它是单调递增的。复杂度 O(nlogn)。

```
#include<bits/stdc++.h>
using namespace std;
int n,c,x[100005];//牛棚数量，牛数量，牛棚坐标

bool check(int dis){      //当牛之间距离最小为 dis 时，检查牛棚够不够
    int cnt=1, place=0;  //第 1 头牛，放在第 1 个牛棚
    for (int i = 1; i < n; ++i)      //检查后面每个牛棚
        if (x[i] - x[place] >= dis){ //如果距离 dis 的位置有牛棚
            cnt++;       //又放了一头牛
            place = i;  //更新上一头牛的位置
        }
    if (cnt >= c) return true;    //牛棚够
    else       return false;  //牛棚不够
}

int main(){
    scanf("%d%d",&n, &c);
    for(int i=0;i<n;i++)    scanf("%d",&x[i]);
    sort(x,x+n);              //对牛棚的坐标排序
    int left=0, right=x[n-1]-x[0];  //R=1000000 也行，因为是 log(n)的，很快
```

```
                                    //优化：把二分上限设置为 1e9/c
    int ans = 0;
    while(left <= right){
        int mid = left + (right - left)/2;      //二分
        if(check(mid)){         //当牛之间距离最小为 mid 时，牛棚够不够?
            ans = mid;          //牛棚够，先记录 mid
            left = mid + 1;     //扩大距离
           }
        else
            right = mid-1;       //牛棚不够，缩小距离
    }

    cout << ans;                //打印答案
    return 0;
}
```

5 [1091] 自动刷题机

陈晖是个小小发明家，最近他发明了一个自动刷题机——一种可以自动 AC 题目的神秘装置。
自动刷题机刷题的方式非常简单：首先会瞬间得出题目的正确做法，然后开始写程序。每秒，自动刷题机的代码生成模块会有两种可能的结果：
1.写了 x 行代码
2.心情不好，删掉了之前写的 y 行代码。（如果 y 大于当前代码长度则相当于全部删除。）
对于一个 OJ，存在某个固定的正整数长度 n，一旦自动刷题机在某秒结束时积累了大于等于 n 行的代码，它就会自动提交并 AC 此题，然后新建一个文件（即弃置之前的所有代码）并开始写下一题。陈晖在某个 OJ 上跑了一天的自动刷题机，得到了很多条关于写代码的日志信息。他突然发现自己没有记录这个 OJ 的 n 究竟是多少。所幸他通过自己在 OJ 上的 Rank 知道了自动刷题机一共成功了解决 k 道题，希望你计算 n 可能的最小值和最大值。
输入格式
第一行两个整数 l, k，表示刷题机的日志一共有 l 行，一共了成功解决了 k 题。
接下来 l 行，每行一个整数 xi，依次表示每条日志。若 xi≥0，则表示写了 xi 行代码，若 xi<0，则表示删除了−xi 行代码。
输出格式
输出一行两个整数，分别表示 n 可能的最小值和最大值。
如果这样的 n 不存在，请输出一行一个整数 −1。
输入输出样例
输入
4 2
2
5
-3
9
输出

3 7
对于 100% 的数据，保证 1≤l≤105，−109≤xi≤109。

容易发现，对于给定的序列，n 越大能过的题是越少的，所以可以二分来求刚好过 k 道题的左右边界。我们可以用两次二分来完成这道题，一次求最小值，一次求最大值。

```
#include <bits/stdc++.h>
using namespace std;

typedef long long LL;
const int N = 100010;
int n, k;
int a[N];
LL l, r;
LL ans1, ans2;

int check(LL u) {
  LL cnt = 0, s = 0;
  for (int i = 1; i <= n; i ++) {
    s += a[i];//当前的累计值 n
    if (s < 0)
      s = 0;
    else if (s >= u){
      s = 0;//重新计数
      cnt ++ ;
    }
  }
  return cnt;
}

int main() {
  scanf("%d%d", &n, &k);
  for (int i = 1; i <= n; i ++)
    scanf("%d", &a[i]);
  ans1 = -1e18, ans2 = 1e18;

  l = 1, r = 1e18;
  while (l <= r) {//求最大值
    LL mid = l + r >> 1;
    if (check(mid) >= k)
      ans1 = mid, l = mid + 1;
    else
      r = mid - 1;
  }
```

```
    l = 1, r = 1e18;
    while (l <= r) {//求最小值
        LL mid = l + r >> 1;
        if (check(mid) <= k)
            ans2 = mid, r = mid - 1;
        else
            l = mid + 1;
    }

    if (ans2 <= ans1)
        printf("%lld %lld\n", ans2, ans1);
    else
        puts("-1");

    return 0;
}
```

第十二讲 递推

重点：

1. 理解递推关系和递推法。
2. 体会递推法解题模型。
3. 熟练应用递推法解决一些实际问题。

递推是计算机数值计算中的一个重要算法。思想是通过数学推导，将复杂的运算化解为若干重复的简单运算，以充分发挥计算机善长于重复处理的特点。

1 递推基础

递推算法以初始{起点}值为基础，用相同的运算规律，逐次重复运算，直至运算结束。这种从"起点"重复相同的方法直至到达一定"边界"，犹如单向运动，用循环可以实现。递推的本质是按规律逐次推出（计算）下一步的结果。

例 1： 植树节那天，有五位参加了植树活动，他们完成植树的棵数都不相同。问第一位同学植了多少棵时，他指着旁边的第二位同学说比他多植了两棵；追问第二位同学，他又说比第三位同学多植了两棵；如此追问，都说比另一位同学多植两棵。最后问到第五位同学时，他说自己植了 10 棵。到底第一位同学植了多少棵树？

分析：设第一位同学植树的棵数为 a1，欲求 a1，需从第五位同学植树的棵数 a5 入手，根据"多两棵"这个规律，按照一定顺序逐步进行推算：

①a5=10;

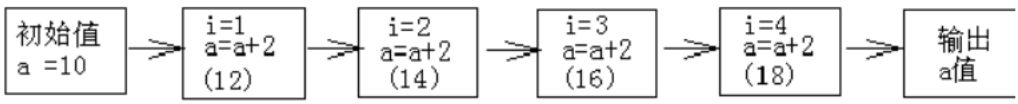
②a4=a5+2=12;

③a3=a4+2=14;

④a2=a3+2=16;

⑤a1=a2+2=18;

本程序的递推运算可用如下图示描述：



程序如下：

```
#include<iostream>
using namespace std;
int main()
{
    int a=10;        //以第五位同学的棵数为递推的起始值
    for (int i=4; i>=1; --i) //还有 4 人，递推计算 4 次
    a+=2;                    //递推运算规律
    cout<<" The Num is "<<a<<endl;
    return 0;
}
```

例满足 F1=F2=1，Fn=Fn-1+Fn-2 的数列称为斐波那契数列（Fibonacci），它的前若干项是 1，1，2，3，5，8，13，21，34……求此数列第 n 项（n>=3）。这也是一个递推问题。用 f (i)表示第 i 项的值，则 f (1) =0，f(2) =1，在 n>2 时，存在递推关系：f (n) = f(n-1) + f(n-2)。

有关 Fibonacci 数列，我们先来考虑一个简单的问题：楼梯有 n 个台阶，上楼可以一步上一阶，也可以一步上两阶。一共有多少种上楼的方法？这是一道计数问题。在没有思路时，不妨试着找规律。n=5 时，一共有 8 种方法：

5=1+1+1+1+1　　5=2+1+1+1

5=1+2+1+1　　5=1+1+2+1

5=1+1+1+2　　5=2+2+1

5=2+1+2　　5=1+2+2

其中有 5 种方法第 1 步走了 1 阶，3 种方法第 1 步走了 2 阶，没有其他可能。假设 f(n)为 n 个台阶的走法总数，把 n 个台阶的走法分成两类。

第 1 类：第 1 步走 1 阶，剩下还有 n-1 阶要走，有 f(n-1)种方法。

第 2 类：第 1 步走 2 阶，剩下还有 n-2 阶要走，有 f(n-2)种方法。

这样，就得到了递推式：f(n)=f(n-1)+f(n-2),不要忘记边界情况：f(1)=1,f(2)=2。把 f(n)的前几项列出：1，2，3，5，8，……。

解决递推问题有三个重点：建立正确的递推关系式；分析递推关系式的性质；根据递推关系式编程求解。递推法分为"顺推"和"倒推"两类模型。所谓顺推，就是从问题的边界条件(初始状态)出发，通过递推关系式依次从前往后递推出问题的解；所谓倒推，就是在不知道问题的边界条件(初始状态)下，从问题的最终解(目标状态或某个中间状态)出发，反过来推导问题的初始状态。

用递推算法求解的题目一般有以下二个特点：

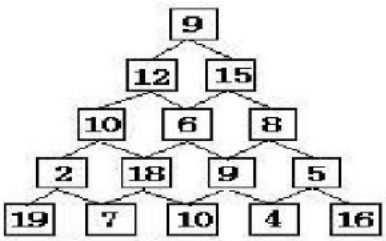
1、问题可以划分成多个状态；

2、除初始状态外，其它各个状态都可以用固定的递推关系式来表示。

在我们实际解题中，题目不会直接给出递推关系式，而是需要通过分析各种状态，找出递推关系式。解决三个问题：目标状态如何描述？递推式是什么？初始条件时什么？

2 [2997] 数塔问题

有如下所示的数塔，要求从顶层走到底层，若每一步只能走到相邻的结点，则经过的结点的数字之和最大是多少？



输入

输入数据首先包括一个整数 C,表示测试实例的个数，每个测试实例的第一行是一个整数 N(1 <= N <= 20)，表示数塔的高度，接下来用 N 个数字表示数塔，其中第 i 行有个 i 个整数，且所有的整数均在区间[0,99]内。

输出

对于每个测试实例，输出可能得到的最大和，每个实例的输出占一行。

样例输入

1

5 9 12 15 10 6 8 2 18 9 5 19 7 10 4 16

样例输出

59

2.1 顺推

从递推的思想出发，当从顶层沿某条路径走到第 i 层向第 i+1 层前进时，我们的选择一定是沿其下两条可行路径中最大数字的方向前进，为此，我们可以采用自上而下的递推方法。

设 f[i][j]表示从第一层走到[i][j]时的最大和，a[i][j]表示每个位置的原始值，则

```
f[i][j]=max(f[i-1][j-1],f[i-1][j])+a[i][j],
```

因为最下面一层第 n 层有多个位置，因此全局最大值为：

```
max(f[i][j]){i=n,j>=1&&j<=n)
```

实现代码如下：

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int f[21][21]={0},a[21][21],n;
    cin>>n;
    for(int i=1;i<=n;i++)
        for(int j=1;j<=i;j++)
            cin>>a[i][j];
    1                    f[1][1]=a[1][1];
    for(int i=2;i<=n;i++){
        for(int j=  2                    1;j<=i;j++){
            f[i][j]=  3                    max(f[i-1][j-1],f[i-1][j])+a[i][j];
        }
    }

    int ans=0;
    for(int i=1;i<=n;i++)
        ans=max(ans,f[n][i]);
    cout<<"max="<<ans;
    return 0;
}
```

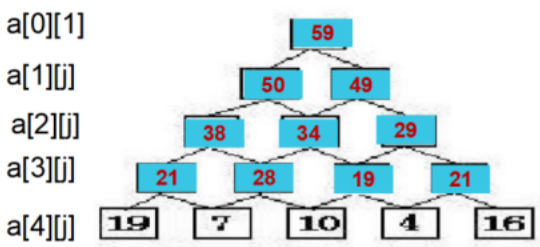
2.2 倒推

因为顺推的值是发散性的，而倒推则是从底层到第一层，不断聚焦。因此，这题用倒推更加方便。设 f[i][j]表示从第一层走到[i][j]时的最大和，a[i][j]表示每个位置的原始值，则

```
f[i][j]=max(f[i+1][j],f[i+1][j+1])+a[i][j];
```

实现代码如下：

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int a[25][25],f[25][25],n;
```

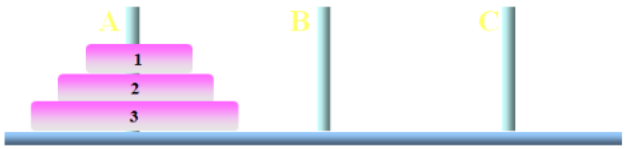


```
    scanf("%d",&n);
    memset(f,0,sizeof(f));
    for(int i=1;i<=n;i++)
        for(int j=1;j<=i;j++)
            scanf("%d",&a[i][j]);
    for(int i=__1_____n;i>=1;i--)
        for(int j=__2_____1;j<=i;j++)
            __3_____f[i][j]=max(f[i+1][j],f[i+1][j+1])+a[i][j];
    printf("max=%d",f[1][1]);
    return 0;
}
```

3 [2998] 汉诺塔 1

题目描述

Hanoi 塔由 n 个大小不同的圆盘和三根木柱 a,b,c 组成。开始时，这 n 个圆盘由大到小依次套在 a 柱上，如图 1 所示。要求把 a 柱上 n 个圆盘按下述规则移到 c 柱上：



(1)一次只能移一个圆盘；

(2)圆盘只能在三个柱上存放；

(3)在移动过程中，不允许大盘压小盘。

问将这 n 个盘子从 a 柱移动到 c 柱上，总计需要移动多少个盘次？

输入 n，输出需要移动多少个盘次

样例输入

3

样例输出

7

设 f(n)为 n 个盘子从 1 柱移到 3 柱所需移动的最少盘次。

当 n=1 时，f(1)=1。

当 n=2 时，f(2)=3。

以此类推，当 1 柱上有 n(n>2)个盘子时，我们可以利用下列步骤：

第一步：先借助 3 柱把 1 柱上面的 n-1 个盘子移动到 2 柱上，所需的移动次数为 f(n-1)。

第二步：然后再把 1 柱最下面的一个盘子移动到 3 柱上，只需要 1 次盘子。

第三步：再借助 1 柱把 2 柱上的 n-1 个盘子移动到 3 上，所需的移动次数为 f(n-1)。

由以上 3 步得出总共移动盘子的次数为：f(n-1)+1+ f(n-1)。

所以：f(n)=2 f(n-1)+1

$h_n=2h_{n-1}+1=2^n-1$

边界条件：$h_1=1$

```
#include<bits/stdc++.h>
using namespace std;
int main(){
```

```
    int n;
    int m = 1;
    cin>>n;
    for(int i = 1;i<=n;i++){
        m = m * 2;
    }
    m--;
    cout<<m;
    return 0;
}
```

4 [2974]传球游戏

上体育课的时候，小蛮的老师经常带着同学们一起做游戏。这次，老师带着同学们一起做传球游戏。游戏规则是这样的：n 个同学站成一个圆圈，其中的一个同学手里拿着一个球，当老师吹哨子时开始传球，每个同学可以把球传给自己左右的两个同学中的一个（左右任意），当老师再次吹哨子时，传球停止，此时，拿着球没传出去的那个同学就是败者，要给大家表演一个节目。

聪明的小蛮提出一个有趣的问题：有多少种不同的传球方法可以使得从小蛮手里开始传的球，传了 m 次以后，又回到小蛮手里。两种传球的方法被视作不同的方法，当且仅当这两种方法中，接到球的同学按接球顺序组成的序列是不同的。比如有 3 个同学 1 号、2 号、3 号，并假设小蛮为 1 号，球传了 3 次回到小蛮手里的方式有 1->2->3->1 和 1->3->2->1，共 2 种。

输入

共一行，有两个用空格隔开的整数 n，m（3<=n<=30，1<=m<=30）。

4.1 问题分析

设 f[i,k]表示经过 k 次传到编号为 i 的人手中的方案数，传到 i 号同学的球只能来自于 i 的左边一个同学和右边一个同学，这两个同学的编号分别是 i-1 和 i+1，所以可以得到以下的递推公式：

经过 k 次传到编号为 i 的人手中的方案数=经过 k-1 次传到左手边同学的方案数+经过 k-1 次传到右手边同学的方案数

f[i][k]=f[i-1][k-1]+f[i+1][k-1]

观察递推式，需要考虑哪些边界条件？

f[1][k]=f[n][k-1]+f[2,k-1],　　当 i=1 时

f[n][k]=f[n-1][k-1]+f[1,k-1],　　当 i=n 时

边界条件：f[1][0]=1；

answer=f[1][m]

4.2 代码

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

```
    int n,m;
    scanf("%d %d",&n,&m);
    int f[31][31]={0};
    f[1][0]=1;//初始
    for(int j=1;j<=m;j++)
    {//m 次传球，n 个同学
        for(int i=1;i<=n;i++)
        {
            if(i==1)
                f[i][j]=f[n][j-1]+f[2][j-1];
            else if (i==n)
                f[i][j]=f[n-1][j-1]+f[1][j-1];
            else
                f[i][j]=f[i-1][j-1]+f[i+1][j-1];
        }
    }
    printf("%d",f[1][m]);
    return 0;
}
```

5 [2864]昆虫繁殖

科学家在热带森林中发现了一种特殊的昆虫，这种昆虫的繁殖能力很强。每对成虫过 x 个月产 y 对卵，每对卵要过两个月长成成虫。假设每个成虫不死，第一个月只有一对成虫，且卵长成成虫后的第一个月不产卵(过 X 个月产卵)，问过 Z 个月以后，共有成虫多少对？0≤X≤20,1≤Y≤20,X≤Z≤50。

输入

x,y,z 的数值。

输出

过 Z 个月以后，共有成虫对数。

样例输入

1 2 8

样例输出

37

分析：

用 a[i]表示第 i 个月拥有的成虫数目，b[i]表示第 i 个月产生的新增卵。由题目可知，前 x 个月成虫数量始终为 1，新增卵为 0。

而以后的第 i 个月，a[i]=a[i-1]+b[i-2]，即第 i 个月的成虫等于第 i-1 个月的成虫数加上第 i-2 个月的新增卵（因为卵两个月后变为成虫）。而 b[i]=a[i-x]*y，即第 i-x 月的成虫在 x 个月后产下 y 个卵。

考虑边界，第一个月只有一只成虫　a[1]=1,b[1]=0；第一个月到第 x 个月都没有产卵 a[i]=1,b[i]=0

```
#include<iostream>
using namespace std;
long long int a[111],b[111];
int main(){
```

```
  int x,y,z,i;
  while(cin>>x>>y>>z)
  {
  for(i=1;i<=x;i++)
  {
    a[i]=1;
    b[i]=0;
  }
  for(i=x+1;i<=z+1;i++){
    b[i]=a[i-x]*y;
    a[i]=a[i-1]+b[i-2];
  }
  cout<<a[z+1]<<endl;
}
  return 0;
}
```

6 [6875] 铺瓷砖

用红色的 1×1 和黑色的 2×2 两种规格的瓷砖不重叠地铺满 n×3 的路面，求出有多少种不同的铺设方案。

[输入格式]一行一个整数 n，0<n<1000。

[输出格式]一行一个整数，为铺设方案的数量模 12345 的结果。

[输入样例] 2

[输出样例] 3

[问题分析]

用 f(n) 表示 n×3 的路面有多少种不同的铺设方案。把路面看成 n 行 3 列，则问题可以分成两种情况考虑，一种是最后一行用 3 块 1×1 的瓷砖铺设；另一种是最后两行用 1 块 2×2 和 2 块 1×1 的瓷砖铺设(最后两行就有两种铺法)，第一种铺法就转换为 f(i-1) 的问题了，第二种铺法就转换成 f(i-2) 的问题了。根据加法原理，得到的递推关系式为 f(i) = f(i-1) +f(i-2)×2，边界为 f(0)=1，f(1)=1。

```
#include<bits/stdc++.h>
using namespace std;
int main(){

    int n,a[1000];
    cin>>n;
    a[0]=1,a[1]=1;
    for(int i=2;i<=n;i++)
        a[i]=(a[i-1]+a[i-2]*2)%12345;
    cout<<a[n];

    return 0;
}
```

7 [3443] 筷子

A 先生有很多双筷子。确切的说应该是很多根，因为筷子的长度不一，很难判断出哪两根是一双的。这天，A 先生家里来了 K 个客人，A 先生留下他们吃晚饭。加上 A 先生，A 夫人和他们的孩子小 A，共 K+3 个人。每人需要用一双筷子。A 先生只好清理了一下筷子，共 N 根，长度为 T1,T2,T3,……,TN.现在他想用这些筷子组合成 K+3 双，使每双的筷子长度差的平方和最小。

输入

共有两行，第一行为两个用空格隔开的整数，表示 N,K(1≤N≤100, 0<K<50)，第二行共有 N 个用空格隔开的整数，为 Ti.每个整数为 1～50 之间的数。

输出

仅一行。如果凑不齐 K+3 双，输出-1，否则输出长度差平方和的最小值。

样例输入

10 1 1 1 2 3 3 3 4 6 10 20

样例输出

5

7.1 问题分析

此题的目标状态是使每双的筷子长度差的平方和最小，用 f[i][j]表示前 i 根组成 j 双筷子每双长度差的和的最小值。在考虑第 i 根筷子时，需要做出的决策是：要不要把这只筷子加入到最优解中去,

若加入，则其与第(i-1)根筷子组成一对，则

f[i][j]=f[i-2][j-1]+(a[i]-a[i-1])*(a[i]-a[i-1]),否则 f[i][j]=f[i-1][j] 。

因为 i 与 i-1 配对，产生一双筷子，这样剩下的筷子就是从 1~i-2。该递推式的意思就是“前 i 根组成 j 双筷子每双长度差的和的最小值 等于前 i-2 根组成 j-1 双筷子每双长度差的和的最小值+最后一双筷子长度差的平方”。 即可写出递推表达式：

f[i][j]=min(f[i-1][j],f[i-2][j-1]+(a[i]-a[i-1])*(a[i]-a[i-1]));

7.2 代码解析

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int a[101],f[101][101];
    int k,i,j,n,t;
    scanf("%d%d",&n,&k);
    k=k+3;
    for (i=1;i<=n;i++) scanf("%d",&a[i]);
    sort(a+1,a+1+n);//将筷子按照从小到大排序
    if (k*2>n) {
        printf("-1");//如果筷子不够则输出-1
        return 0;
    }
```

```
    for(i=0;i<=n;i++)//初始化，f[i][j]表示 i 根筷子里取 j 双的最优值
    {
        f[i][0]=0;
        for (j=1;j<=k;j++)
            f[i][j]=99999999;
    }
    for (i=2;i<=n;i++)//因为一根筷子没意义，所以从 2 根筷子开始更新
        for (j=1;j<=k;j++)
            f[i][j]=min(f[i-1][j],f[i-2][j-1]+(a[i]-a[i-1])*(a[i]-a[i-1]));
    printf("%d",f[n][k]);
}
```

课堂作业列表

[2997] 数塔问题	[2998] 汉诺塔 1
[2974] 传球游戏	[2864] 昆虫繁殖
[6875] 铺瓷砖	[6876] 彩带
[2757] 移动路线	[2459] 递推求猴子吃桃

递推习题

1. 输出：_____

```
#include <iostream>
using namespace std;
int main(    )
{
    int    x[8]={37,43,56,28,90,13,55,79 },i,j,t;
    i=0;    j=7;
    while(i<j)
    {     t=x[i];x[i]=x[j]; x[j]=t; i++;    j--; }

    for( i=0;i<3;i++ )    printf("%5d",x[i]);
}
```

2. [2008]

```
#include<iostream>
using namespace std;
int main()
{
    int i, a, b, c, d, f[4];
    for(i = 0; i < 4; i++) cin >> f[i];
    a = f[0] + f[1] + f[2] + f[3];
    a = a / f[0];
    b = f[0] + f[2] + f[3];
```

```
    b = b / a;
    c = (b * f[1] + a) / f[2];
    d = f[(b / c ) % 4];
    if(f[(a + b + c + d) % 4] > f[2])
        cout << a + b<< endl;
    else
        cout << c + d << endl;
    return 0;
}
```

输入：9 19 29 39

输出：_____

3 [2759]: 有一个方格矩阵，矩阵边界在无穷远处。我们做如下假设：

a.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；

b.走过的格子立即塌陷无法再走第二次；c.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。2 种走法只要有一步不一样，即被认为是不同的方案。

写出递推式_____

4 [2756]: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。

写出递推式_____

第十三讲 动态规划基础

重点：

1. 体会动态规划算法的基本思想。
2. 了解动态规划算法中的一些基本概念。
3. 初步应用动态规划算法求解一些实际问题。

前面讲的分治法是将原问题分解为若干个规模较小、形式相同的子问题，然后求解这些子问题，合并子问题的解得到原问题的解。在分治法中，各个子问题是互不相交的，即相互独立。如果各个子问题有重叠，不是相互独立的，那么用分治法就重复求解了很多子问题，根本显现不了分治的优势，反而降低了算法效率。此时就可以通过规划方法解决。动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法，没有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。

1 动态规划的基本模型

动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值（最大值或最小值）的那个解。设计一个动态规划算法，通常可以按以下几个步骤进行：

（1）找出最优解的性质，并刻画其结构特征。

（2）递归地定义最优值。

（3）以自底向上的方式计算出最优值。

（4）根据计算最优值时得到的信息，构造一个最优解。

其中（1）—（3）步是动态规划算法的基本步骤。在只需要求出最优值的情形，步骤（4）可以省去。若需要求出问题的一个最优解，则必须执行步骤（4）。此时，在步骤（3）中计算最优值时，通常需记录更多的信息，以便在步骤（4）中，根据所记录的信息，快速构造出一个最优解。

动态规划算法的有效性依赖于问题本身所具有的两个重要性质：

1、最优子结构：当问题的最优解包含了其子问题的最优解时，称该问题具有最优子结构性质。

2、重叠子问题：在用递归算法自顶向下解问题时，每次产生的子问题并不总是新问题，有些子问题被反复计算多次。动态规划算法正是利用了这种子问题的重叠性质，对每一个子问题只解一次，而后将其解保存在一个表格中，在以后尽可能多地利用这些子问题的解。

2 [4003]城市交通

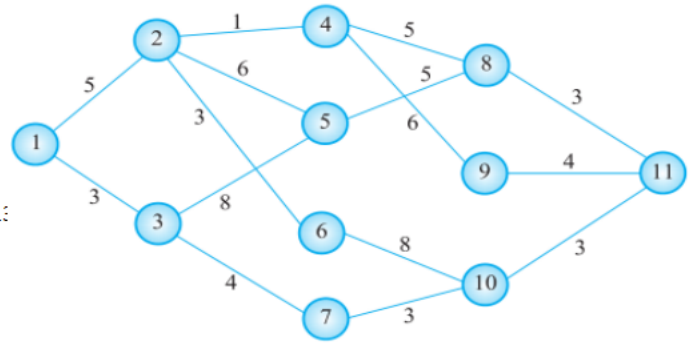
有 n 个城市，编号 1~n，有些城市之间有路相连，有些则没有，有路则会有一个距离。图 所示为一个含有 11 个城市的交通图，连线上的数（权）表示距离

现在规定只能从编号小的城市到编号大的城市。问：

从编号为 1 的城市到编号为 n 的城市之间的最短距离是多少？

【输入格式】

第 1 行为 n，表示城市数，n≤100。



下面的 n 行是一个 n×n 的邻接矩阵 map[i,j]，其中 map[i,j]=0 表示城市 i 和城市 j 之间没有路相连，否则为两者之间的距离。

【输出格式】

一行一个数，表示最短距离。数据保证一定可以从城市 1 到城市 n。

【输入样例】

```
11
0 5 3 0 0 0 0 0 0 0 0
5 0 0 1 6 3 0 0 0 0 0
3 0 0 0 8 0 4 0 0 0 0
0 1 0 0 0 0 0 5 6 0 0
0 6 8 0 0 0 0 5 0 0 0
0 3 0 0 0 0 0 0 0 8 0
0 0 4 0 0 0 0 0 0 3 0
0 0 0 5 5 0 0 0 0 0 3
0 0 0 6 0 0 0 0 0 0 4
0 0 0 0 0 8 3 0 0 0 3
0 0 0 0 0 0 0 3 4 3 0
```

【输出样例】

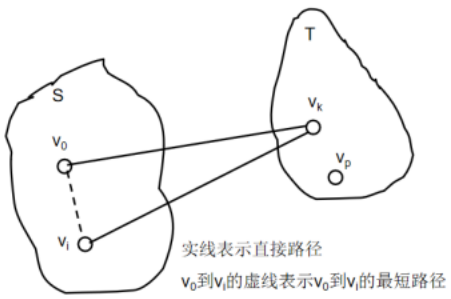
```
13
```

2.1 问题分析

逆向思考，现在想要从城市 1 到达城市 11，则只能从城市 8、9 或 10 中转过去，如果知道了从城市 1 到城市 8、9 和 10 的最短距离，那么只要把这 3 个最短距离分别加上这 3 个城市与城市 11 之间的距离，再从中取一个最小值即可。设 dis[i]为城市 1 到城市 i 的最短距离，a[i][j]表示城市 i 到城市 j 的直达距离，则

dis[11]=min(dis[8]+a[8][11],dis[9]+a[9][11],dis[10]+a[10][11];

这样一来，问题就变成了求城市 1 到城市 8、9、10 的最短距离，而这 3 个子问题与原问题是完全致的，只是问题的规模缩小了一点。如何求城市 1 到城市 8 的最短距离呢？想法与刚才一样，如果知道了城市 1 到城市 4 和 5 的最短距离，那么到城市 8 的最短距离就是到城市 4 的最短距离加上 5 以及到城市 5 的最短距离加上 5 当中较小的那个值。而如何求城市 4 和 5 的最短距离呢？...如此下去，直到求城市 1 到城市 2 和 3 的最短距离，而这个值是已知的（因为城市 1 和 2、3 之间有直接的边相连）。这种解题思想就是“动态规划”。



可以证明 v_0 到 T 中顶点 v_k 的最短路径，要么是从 v_0 到 v_k 的直接路径；要么是从 v_0 经 S 中某个顶点 vi 再到 v_k 的路径。

假设 dis[i]表示从城市 1 到城市 i 的最短距离，求解过程可以用下列式子归纳：

dis[1]=0

dis[i]=min{dis[i],dis[j]+map[j,i]}（i>1，j 从 1 到 i-1 且 map[j,i] 不等于 0）

最后，只要输出 dis[n]。

2.2 代码详解

```
#include<iostream>
using namespace std;
const int inf=0xffffff;
int main()
{
    int n;
    int dis[105][105],p[105];
    scanf("%d",&n);//输入[i]到[j]的距离
    for(int i=1;i<=n;i++)
        for(int j=1;j<=n;j++)
            scanf("%d",&dis[i][j]);
    //初始化[1]到[i]的距离最大，在由小的慢慢代替
    p[1]= _1_____0;
    for(int i=2;i<=n;i++)
       p[i]= _2_____;
    //找出[1]到[i]的最小距离
    for(int i=2;i<=n;i++)
        for(int j= _3_____1;j<i;j++)
            if(dis[i][j]!=0)
            p[i]=min( _4_____p[i],p[j]+dis[i][j]);
    printf("%d",p[n]);
    return 0;
}
```

3 [2881]最长上升降序列

设有由 n(1≤n≤200)个不相同的整数组成的数列，记为:b(1)、b(2)、……、b(n)且 b(i)≠b(j)(i≠j)，若存在 $i_1<i_2<i_3<...<i_n$ 且有 $b(i_1)<b(i_2)<...<b(i_n)$则称为长度为 n 的上升序列。程序要求，当原数列出之后，求出最长的不下降序列。

例如 13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中 13，16，18，19，21，22，63 就是一个长度为 7 的不下降序列，同时也有 7 ，9，16，18，19，21，22，63 组成的长度为 8 的不下降序列。

输入第一行为 n，第二行为用空格隔开的 n 个整数。输出第一行为输出最大个数 max(形式见样例)；第二行为 max 个整数形成的不下降序列,答案可能不唯一，输出一种就可以了，本题进行特殊评测。

样例输入

14

13 7 9 16 38 24 37 18 44 19 21 22 63 15

样例输出

max=8

3.1 问题分析

根据动态规划的原理，从前往后进行搜索(当然也可以从后往前也)。

1. 对 b(1)来说，由于它是第一个数，所以当从 b(1)开始查找时，只存在长度为 1 的不下降序列；
2. 若从 b(2)开始查找，则存在下面的两种可能性：

①若 b(2)<b(1)则存在长度为 1 的不下降序列 b(n-1)或 b(n)。

②若 b(2)>b(n)则存在长度为 2 的不下降序列 b(1)，b(2)。

3. 一般若从 b(i)开始，此时最长上升降序列应该按下列方法求出:

在 b(,1),b(2),…,b(i-1)中，找出一个比 b(i)小的且最长的上升序列，作为它的前驱。

定义两个整数类型数组 a[N]、b[N]，a 表示第 i 个数的数值本身；b 表示从第一个位置到达第 i 个位置时的最长上升序列长度，求解过程：

①从数第二项开始计算，前面仅有 1 项，比较一次，因 7<13，不符合要求，长度仍为 1。

②从第三项开始前面有 2 项，需做两次比较，得到目前最长的上升序列为 2，如下表：

一般处理过程是：

①在 1,2,…,i-1 项中，找出比 a[i]小的最长长度 L;

②若 L>0，则 b[i]=L+1;

最后本题经过计算，其数据存储表如下：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a[i]	13	7	9	16	38	24	37	18	44	19	21	22	63	15
b[i]	1	1	2	3	4	4	5	4	6	5	6	7	8	3

3.2 代码详解

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
   int a[205],b[205],i,j,n;
   cin>>n;
   for(i=1;i<=n;i++){
        cin>>a[i];
        __1__ b[i]=1;//初始化
   }
   for(i=1;i<=n;i++){//求解每个值对应的最长上升子序列
        int k=0;
        for(j=1;j<i;j++){//扫描前面所有的数
            if(__2__a[j]<a[i]&&b[j]>k)
                  k=__3__b[j];
        }
```

```
        4                b[i]=k+1;
    }
    sort(b+1,b+1+n);
    cout<<"max="<<b[n]<<endl;
     return 0;
}
```

思考：从后往前怎么写？

4 [3031] 拦截导弹

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

每个测试文件只包含一组测试数据，每组输入若干个整数，表示导弹依次飞来的高度（雷达给出的高度数据是不大于 30000 的正整数）。

输出

对于每组输入数据，第一行输出这套系统最多能拦截多少导弹，第二行输出如果要拦截所有导弹最少要配备多少套这种导弹拦截系统。

样例输入

389 207 155 300 299 170 158 65

样例输出

6

2

第一问即经典的最长不下降子序列问题，可以用一般的 DP 算法，也可以用高效算法，　　第二问用贪心法即可。每颗导弹来袭时，使用能拦截这颗导弹的防御系统中上一次拦截导弹高度最低的那一套来拦截。若不存在符合这一条件的系统，则使用一套新系统。

```
#include<iostream>
    using namespace std;
    int n,ans1,ans2;
    int h[105];
    void dp1()//求最长上升子列
    {
        int dp[105];
        ans1=0;
        for(int i=1;i<=n;i++)
        {
            dp[i]=1;
            for(int j=1;j<i;j++)
            {
                if(h[j]<h[i])
                    dp[i]=max(dp[j]+1,dp[i]);
            }
            ans1=max(ans1,dp[i]);
```

```
        }
    }
    void dp2()
    {
        int dp[105];ans2=0;
        for(int i=1;i<=n;i++)
        {
            dp[i]=1;
            for(int j=1;j<i;j++)
                if(h[j]>h[i])
                    dp[i]=max(dp[j]+1,dp[i]);
            ans2=max(ans2,dp[i]);
        }
    }
    int main()
    {
        int i=1,a;
        while(scanf("%d",&a)!=EOF)
        {
            h[i]=a;
            i++;
        }
        n=i-1;
        dp1();
        dp2();
        cout<<ans2<<endl<<ans1<<endl;
        return 0;
    }
```

课堂作业列表

[4003]城市交通
[2997]数塔问题
[2881]最长不下降序列
[2841]拦截导弹
[2817]最长的公共子序列
[1165]求最长不下降子序列的长度
[2835]登山

动态规划基础习题

1 下面程序的功能是将两个以按从小到大排序的数组合并成一个有序数组。

请填空。

```
main(   )
{       int   a[5]={ 2,7,13,28,76 },i,j,k;
        int   b[5]={ 3,8,11,35,49 },c[10];
        i=0;   j=0;   k=0;
        while( i<5 && j<5 )
            if(  1          ) {   c[k]=a[i];   i++;   k++; }
            else ( c[k]=b[j];   j++;   k++; )
        while( i<5 ) {  2           ; i++;   k++;   }
        while( j<5 ) {  3           ;   j++;   k++   }
        for( i=0;i<10;i++   )   printf("%4d",c[i]);
```

2 下面程序的功能是在有序数组 w 中插入数据 x，使插入后的数组仍保持有序。请填空。

```
main()
{     int w[11]={12,16,23,54,60,79,104,163,199,253},i,j,x;
      scanf("%d",&x);            /*输入要插入的数据*/
      i=0;  w[10]=x;
while(   1       ) i++;
for(j=10;j>1;j--)      w[j]=    2          ;
w[i]=x;
for(j=0;j<  3           ;j++) printf("%4d",w[j]);
}
```

3. 设有由 n(1≤n≤200 个不相同的整数组成的数列，记为:b(1)、b(2)、……、b(n)且 b(i)≠b(j)，若存在 i1<i2<i3<…<ie 且有 b(i1)<b(i2)<…<b(ie)则称为长度为 e 的不下降序列。程序要求，当原数列出之后，求出最长的不下降序列。

例如 13，7，9，16，38，24，37，18，44，19，21，22，63，15。例中 13，16，18，19，21，22，63 就是一个长度为 7 的不下降序列，同时也有 7 ，9，16，18，19，21，22，63 组成的长度为 8 的不下降序列。

输入第一行为 n,第二行为用空格隔开的 n 个整数。

输出第一行为输出最大个数 max(形式见样例)；

第二行为 max 个整数形成的不下降序列,答案可能不唯一，输出一种就可以了，本题进行特殊评测。

样例输入

```
14
13 7 9 16 38 24 37 18 44 19 21 22 63 15
```

样例输出

```
max=8
```

```
#include<iostream>
using namespace std;
int main()
{
   int n,i,j,l,k,b[200][10];
```

```
    cin>>n;
    for (i=1;i<=n;i++)
      //输入序列的初始值
     {
       cin>>b[i][1];
        1
     }
    for (i=n-1;i>=1;i--)
  //求最长不下降序列
      {
        2
        for (j=i+1;j<=n;j++)
         if ( 3 )
          {
             l=b[j][2];
             k=j;
          }
        if (l>0)
         {
           b[i][2]=l+1;b[i][3]=k;
         }
      }
      k=1;//下面找出最长不下降序列
      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])   4
      cout<<"max="<<b[k][2]<<endl;  //输出结果
}
```

第十四讲 01 背包

重点：

1. 理解 01 背包算法的基本思想。
2. 应用 01 背包算法求解一些实际问题。
3. 了解完全背包和多重背包。

背包问题(Knapsack problem)是一种组合优化问题，是动态规划中的一个经典题型。问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。背包问题可以分为 01 背包、完全背包、多重背包等多重情况。

01 背包（ZeroOnePack）：有 N 件物品和一个容量为 V 的背包。（每种物品均只有一件）第 i 件物品的费用是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使价值总和最大。

完全背包(CompletePack)：有 N 种物品和一个容量为 V 的背包，每种物品都有无限件可用。第 i 种物品的费用是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

多重背包(MultiplePack)：有 N 种物品和一个容量为 V 的背包。第 i 种物品最多有 n[i]件可用，每件费用是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。比较三个题目，会发现不同点在于每种背包的数量，01 背包是每种只有一件，完全背包是每种无限件，而多重背包是每种有限件。

1 理解 01 背包

在正式讲解01背包问题前，我们先来看一个国王的故事。

有一个国家，所有的国民都非常老实憨厚，某天他们在自己的国家发现了十座金矿，并且这十座金矿在地图上排成一条直线，国王知道这个消息后非常高兴，他希望能够把这些金子都挖出来造福国民，首先他把这些金矿按照在地图上的位置从西至东进行编号，依次为1、2、3、4、5、6、7、8、9、10，然后他命令他的手下去对每一座金矿进行勘测，以便知道挖取每一座金矿需要多少人力以及每座金矿能够挖出多少金子，然后动员国民都来挖金子。

那么，国王究竟如何知道在只有 10000 个人的情况下最多能挖出多少金子呢？国王是如何思考这个问题的呢？

国王首先来到了第 10 个金矿的所在地（注意，第 10 个就是最后一个，因为是从 1 开始编号的，最西边的那个金矿是第 1 个），他的臣子告诉他，如果要挖取第 10 个金矿的话就需要 1500 个人，并且第 10 个金矿可以挖出 8888 个金子。听到这里国王哈哈大笑起来，因为原先他以为要知道十个金矿在仅有 10000 个人的情况下最多能挖出多少金子是一件很难思考的问题，但是，就在刚才听完他的臣子所说的那句话时，国王已经知道总共最多能挖出多少金子了，国王是如何在不了解其它金矿的情况下知道最多能挖出多少金子的呢？他的臣子们也不知道这个谜，因此他的臣子们就问他了："最聪明的国王陛下，我们都没有告诉您其它金矿的情况，您是如何知道最终答案的呢？"

得意的国王笑了笑，然后把他最得意的"左、右手"叫到跟前，说到："我并不需要考虑最终要挖哪些金矿才能得到最多的金子，我只需要考虑我面前的这座金矿就可以了，对于我面前的这座金矿不外乎仅有两种选择，要么挖，要么不挖，对吧？"

"当然，当然"大臣们回答倒。

国王继续说道："如果我挖取第 10 座金矿的话那么我现在就能获得 8888 个金子，而我将用去 1500 个人，那么我还剩下 8500 个人。我亲爱的左部下，如果你告诉我当我把所有剩下的 8500 个人和所有剩下的

其它金矿都交给你去开采你最多能给我挖出多少金子的话，那么我不就知道了在第 10 个金矿一定开采的情况下所能得到的最大金币数吗？″
″是啊，是啊……如果第 10 座金矿一定开采的话……″大臣们点头说到。

国王笑着继续对着他的右部下说到：″亲爱的右部下，也许我并不打算开采这第 10 座金矿，那么我依然拥有 10000 个人，如果我把这 10000 个人和剩下的金矿都给你的话，你最多能给我挖出多少个金子呢？″
国王的右部下聪明地说道：″尊敬的国王陛下，我明白您的意思了，如果我回答最多能购开采出 y 个金币的话，那您就可以在 y 和 x+8888 之间选择一个较大者，而这个较大者就是最终我们能获得的最大金币数，您看我这样理解对吗？″

国王笑得更灿烂了，问他的左部下：″那么亲爱的左部下，我给你 8500 个人和其余金矿的话你能告诉我最多能挖出多少金子吗？″
″请您放心，这个问题难不倒我″。左部下向国王打包票说到。
国王高兴地继续问他的右部下：″那右部下你呢，如果我给你 10000 个人和其余金矿的话你能告诉我最多能挖出多少金子吗？″
″当然能了！交给我吧！″右部下同左部下一样自信地回答道。

国王走后，国王的左、右部下来到了第 9 座金矿，早已在那里等待他们的金矿勘测兵向两位大臣报道：″聪明的两位大臣，您们好，第 9 座金矿需要 1000 个人才能开采，可以获得 7000 个金子″。
因为国王仅给他的左部下 8500 个人，所以国王的左部下叫来了两个人，对着其中一个人问到：″如果我给你 7500 个人和除了第 9、第 10 的其它所有金矿的话，你能告诉我你最多能挖：″如果我给你 7500 个人和除了第 9、第 10 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？″
然后国王的左部下继续问另一个人：″如果我给你 8500 个人和除了第 9、第 10 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？″
国王的左部下在心里想着：″如果他们俩都能回答我的问题的话，那国王交给我的问题不就解决了吗？哈哈哈！″

因为国王给了他的右部下 10000 个人，所以国王的右部下同样也叫来了两个人，对着其中一个人问：″如果我给你 9000 个人和除了第 9、第 10 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？″
然后国王的右部下继续问他叫来的另一个人：″如果我给你 10000 个人和除了第 9、第 10 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？″
此时，国王的右部下同左部下一样，他们都在为自己如此聪明而感到满足。
当然，这四个被叫来的人同样自信地回答没有问题，因为他们同样地从这两个大臣身上学到了相同的一点，而两位自认为自己一样很聪明的大臣得意地笑着回到了他们的府邸，等着别人回答他们提出来的问题，现在你知道了这两个大臣是如何解决国王交待给他们的问题了吗？
那么你认为被大臣叫去的那四个人又是怎么完成大臣交给他们的问题的呢？答案当然是他们找到了另外八个人！

没用多少功夫，这个问题已经在全国传开了，更多人的人找到了更更多的人来解决这个问题，而有些人却不需要去另外找两个人帮他，哪些人不需要别人的帮助就可以回答他们的问题呢？
很明显，当被问到给你 z 个人和仅有第 1 座金矿时最多能挖出多少金子时，就不需要别人的帮助，因为你知道，如果 z 大于等于挖取第 1 座金矿所需要的人数的话，那么挖出来的最多金子数就是第 1 座金矿能够挖出来的金子数，如果这 z 个人不够开采第 1 座金矿，那么能挖出来的最多金子数就是 1，因为这唯一的金矿不够人力去开采。

故事讲到这里先暂停一下，我们现在重新来分析一下这个故事，让我们对动态规划有个理性认识。

子问题：
国王需要根据两个大臣的答案以及第 10 座金矿的信息才能判断出最多能够开采出多少金子。为了解决自己

面临的问题，他需要给别人制造另外两个问题，这两个问题就是子问题。

思考动态规划的第一点----最优子结构：

国王相信，只要他的两个大臣能够回答出正确的答案，再加上他的聪明的判断就一定能得到最终的正确答案。我们把这种子问题最优时母问题通过优化选择后一定最优的情况叫做"最优子结构"。

思考动态规划的第二点----子问题重叠：

实际上国王也好，大臣也好，所有人面对的都是同样的问题，即给你一定数量的人，给你一定数量的金矿，让你求出能够开采出来的最多金子数。我们把这种母问题与子问题本质上是同一个问题的情况称为"子问题重叠"。

思考动态规划的第三点----边界：

想想如果不存在前面我们提到的那些底层劳动者的话这个问题能解决吗？永远都不可能！我们把这种子问题在一定时候就不再需要提出子子问题的情况叫做边界，没有边界就会出现死循环。

思考动态规划的第四点----子问题独立：

要知道，当国王的两个大臣在思考他们自己的问题时他们是不会关心对方是如何计算怎样开采金矿的，因为他们知道，国王只会选择两个人中的一个作为最后方案，另一个人的方案并不会得到实施，因此一个人的决定对另一个人的决定是没有影响的。我们把这种一个母问题在对子问题选择时，当前被选择的子问题两两互不影响的情况叫做"子问题独立"。

这就是动态规划，具有"最优子结构"、"子问题重叠"、"边界"和"子问题独立"，当你发现你正在思考的问题具备这四个性质的话，那么恭喜你，你基本上已经找到了动态规划的方法。同时这也是一个典型的 01 背包问题，

01 背包的本质就是一个物体的取或不取。

设有 N 件物品（等同于 10 个金矿）和一个容量为 V 的背包（等同挖金矿的人数）。（每种物品均只有一件）第 i 件物品的费用是 c[i]，价值是 w[i]。求解将哪些物品装入背包可使价值总和最大。

这是最基础的背包问题，特点是：每种物品仅有一件，可以选择放或不放。

子问题定义： f[i][v]表示前 i 件物品放入一个容量为 v 的背包可以获得的最大价值，那么我们可以很容易分析得出 f[i][v]的计算方法，

（1）v< w[i] 的情况，这时候背包容量不足以放下第 i 件物品，只能选择不拿 f[i][v]=f[i-1][v]

（2）v>=w[i] 的情况，这时背包容量可以放下第 i 件物品，我们就要考虑拿这件物品是否能获取更大的价值。

如果拿取，f[i][v] =f[i-1][v-w[i]] + c[i]。 这里的 f[i-1][v-w[i]] + c[i]指的就是考虑了 i-1 件物品，背包容量为 v-w[i]时的最大价值，也是相当于为第 i 件物品腾出了 w[i]的空间。

如果不拿， f[i][v] = f[i-1][v]，同（1）。究竟是拿还是不拿，自然是比较这两种情况那种价值最大。由此可以得到状态转移方程：

```
if(v>=w[i])
  f[i][v]=max(m[i-1][v],f[i-1][v-w[i]]+c[i]);
else
  m[i][v]=m[i-1][v];
```

即：f[i][v]=max{f[i-1][v],f[i-1][v-w[i]]+c[i]}。

这个方程非常重要，基本上所有跟背包相关的问题的方程都是由它衍生出来的。所以有必要再将它详细解释一下："将前 i 件物品放入容量为 v 的背包中"这个子问题，若只考虑第 i 件物品的策略（放或不放），那么就可以转化为一个只牵扯前 i-1 件物品的问题。如果不放第 i 件物品，那么问题就转化为"前 i-1 件物品放入容量为 v 的背包中"，价值为 f[i-1][v]；如果放第 i 件物品，那么问题就转化为"前 i-1 件物品放入剩下的容量为 v-c[i]的背包中"，此时能获得的最大价值就是 f[i-1][v-c[i]]再加上通过放入第 i 件物品获得的价值 w[i]。

假设价值数组 C= {8, 10, 6, 3, 7, 2}，重量数组 w = {4, 6, 2, 2, 5, 1}，背包容量 V = 12 时对应的 f[i][v]数组。可用如下表格描述：

0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	8	8	8	8	8	8	8	8	8
2	0	0	0	8	8	10	10	10	10	18	18	18
3	0	6	6	8	8	14	14	16	16	18	18	24
4	0	6	6	9	9	14	14	17	17	19	19	24
5	0	6	6	9	9	14	14	17	17	19	21	24
6	2	6	8	9	11	14	16	17	19	19	21	24

2 [7277] 国王的金矿

国王在他的国家发现了 n 座金矿，为了描述方便，我们给他们从 1 到 n 编号。对于第 i 个金矿，需要投入 w(i) 个的费用，能挖出来 V(i)个单位的金子。

现在国王想开挖这些金矿，但是最多只有 M 个人力用于投入，问最多可以挖出来多少单位的金子。

输入第一行两个整数，分别为 N 和 M；接下来 N 行每行两个整数，第 i+1 行为 w(i)和 v(i)。

输出一行一个整数，为最多可以挖出来多少单位的金子。

样例输入

5 10

3 350

5 500

5 400

4 300

3 200

样例输出

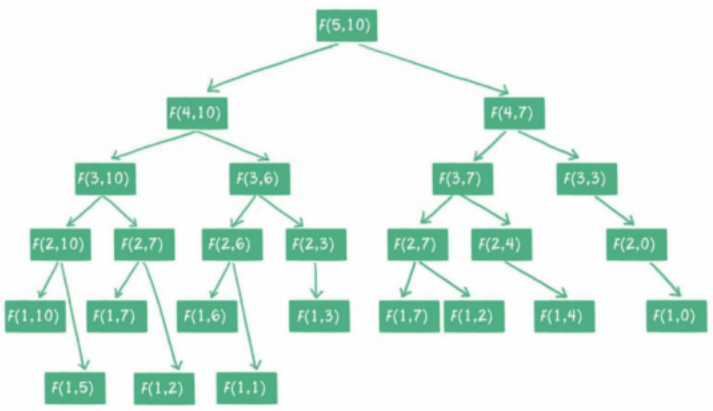
900

2.1 问题分析

假设如果最后一个金矿注定不被挖掘，那么问题会转化成 10 个工人在前 4 个金矿中做出最优选择；

相应地，假设最后一个金矿一定会被挖掘，那么问题转化成 7 个工人在前 4 个金矿中做出最优选择（由于最后一个金矿消耗了 3 个工人）

最后一个金矿到底该不该挖呢？ 那就要看 10 个工人在前 4 个金矿的收益，和 7 个工人在前 4 个金矿的收益+最后一个金矿的收益谁大谁小了。



设 f[i][v]表示前 i 座金矿被 v 个人挖掘后，可以获得的最大价值，那么我们可以很容易分析得出 f[i][v]的计算方法：

f[5][10]=max(f[4][10],f[4][7]+350);

设 w[i]表示挖掘第 i 座金矿需要的人数，挖掘后能获得的价值是 c[i]，则

f[i][v]=max{f[i-1][v],f[i-1][v-w[i]]+c[i]}。

同样的道理，对于前 4 个金矿的选择，我们还可以做进一步简化。

就这样，问题一分为二，二分为四，一直把问题简化成在 0 个金矿或 0 个 工人时的最优选择，这个收益结果显然是 0，也就是问题的边界。

2.2 代码解析

```
#include<bits/stdc++.h>
using namespace std;
int f[205][3005],w[205],c[205];
int main(){
    int n,v;
    scanf("%d %d",&n,&v);
    for(int i=1;i<=n;i++){
        scanf("%d %d",&w[i],&c[i]);
    }
    memset(f,0,sizeof(f));
    for(int i=1;i<=n;i++){
        for(int j=  1                          ){
            if(  2                  )
                f[i][j]=  3                    );
            else
                  4                 
        }
    }
    printf("%d",f[n][v]);
    return 0;
}
```

思考： for(int i=1;i<=n;i++){

... ...

}

有没有简写的方式?

for(int j= 1){

2 ;

}

3 [3894] 大卖场购物车

央视有一个大型娱乐节目—购物街，舞台上模拟超市大卖场，有很多货物，每个嘉宾分配一个购物车，可以尽情地装满购物车，购物车中装的货物价值最高者取胜。假设有 n 个 物品和 1 个购物车，每个物品 i 对应价值为 vi，重量 wi，购物车的容量为 W（你也可以将重 量设定为体积）。每个物品只有 1 件，要么装入，要么不装入，不可拆分。在购物车不超重的情况下，如何选取物品装入购物车，使所装入的物品的总价值最大？最大价值是多少？装 入了哪些物品？

输入 T，表示有 T 组数据,输入物品的个数 n ,输入购物车的容量 W(1<=W<=20) 依次输入每个物品的重量 w 和价值 v，用空格分开；输出装入购物车的最大价值是多少。

样例输入

1

5

10

2 6 5 3 4 5 2 4 3 6

样例输出

17

3.1 问题分析

n 个物品、物车的容量 W，每个物品的重量为 w[i]，价值为 v[i] 。选若干个物品放入购物车，使价值最大，可表示如下，该问题就是经典的 0-1 背包问题。

$$\text{约束条件：}\begin{cases}\sum_{i=1}^{n} w_i x_i \leqslant W \\ x_i \in \{0,1\}\ ,1 \leqslant i \leqslant n\end{cases}$$

$$\text{目标函数：}\ \max\sum_{i=1}^{n} v_i x_i$$

假设现在有 5 个物品，每个物品的重量为（2，5，4，2，3），价值为（6，3，5，4，6）， 如图所示。购物车的容量为 10，求在不超过购物车容量的前提下，把哪些物品放入购物车，才能获得最大价值

	1	2	3	4	5
w[ ]	2	5	4	2	3

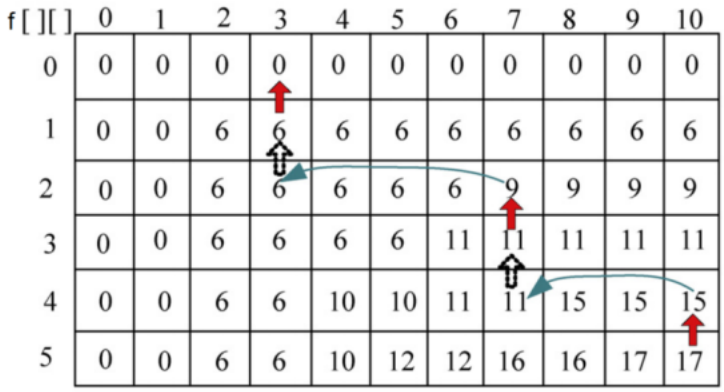
	1	2	3	4	5
v[ ]	6	3	5	4	6

物品的重量和价值

打表输出选择结果：

f[ ][ ]	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	6	6	6	9	9	9	9
3	0	0	6	6	6	6	11	11	11	11	11
4	0	0	6	6	10	10	11	11	15	15	15
5	0	0	6	6	10	12	12	16	16	17	17

构造最优解：



首先读取 f[5][10]>f[4][10]，说明第 5 个物品装入了购物车，即 j=10−w[5]=7；
去找 f[4][7]=f[3][7]，说明第 4 个物品没装入购物车；
去找 f[3][7]>f[2][7]，说明第 3 个物品装入了购物车，即 j= j−w[3]=3；
去找 f[2][3]=f[1][3]，说明第 2 个物品没装入购物车；
去找 f[1][3]>f[0][3]，说明第 1 个物品装入了购物车，即 j= j−w[1]=1。

3.2 代码分析

```
#include <bits/stdc++.h>
using namespace std;
#define maxn 10005
#define M 105
int f[M][maxn];//f[i][j] 表示前 i 个物品放入容量为 j 购物车获得的最大价值
int w[M],v[M];//w[i] 表示第 i 个物品的重量，v[i] 表示第 i 个物品的价值
int x[M];  //x[i]表示第 i 个物品是否放入购物车
int main()
{
   int T;
   cin>>T;
   while(T--)
   {
      int i,j,n,W;//n 表示 n 个物品，W 表示购物车的容量
```

```
    cin >> n >> W;
    for(i=1; i<=n; i++)
      cin>>w[i]>>v[i];
    memset(f,0,sizeof(f));
    for(i=1; i<= n; i++) //计算 c[i][j]
      for(j=1; j<=W; j++)
        if(j<w[i])  //当物品的重量大于购物车的容量，则不放此物品
          f[i][j] = f[i-1][j];
        else    //否则比较此物品放与不放是否能使得购物车内的价值最大
          f[i][j] = max(f[i-1][j],f[i-1][j-w[i]] + v[i]);
    cout<<f[n][W]<<endl;
  }
  return 0;
}
```

以上方法的时间和空间复杂度均为 O(N*V)，其中时间复杂度基本已经不能再优化了，但空间复杂度却可以优化到 O(V)。

先考虑上面讲的基本思路如何实现，肯定是有一个主循环 i=1..N，每次算出来二维数组 f[i][0..V]的所有值。那么，如果只用一个数组 f [0..V]，能不能保证第 i 次循环结束后 f[v]中表示的就是我们定义的状态 f[i][v]呢？f[i][v]是由 f[i-1][v]和 f[i-1][v-w[i]]两个子问题递推而来，能否保证在推 f[i][v]时能够得到 f[i-1][v]和 f[i-1][v-w[i]]的值呢？事实上，这要求在每次主循环中我们以 v=V..0 的逆序推 f[v]，这样才能保证推 f[v]时 f[v-w[i]]保存的是状态 f[i-1][v-w[i]]的值。

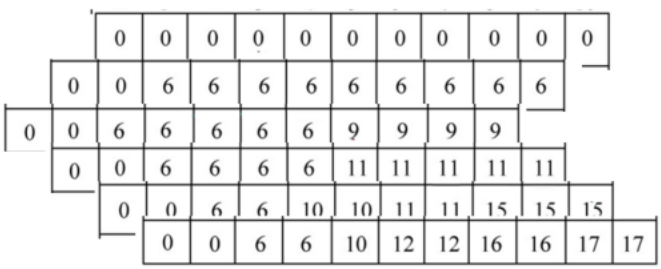
伪代码如下：

```
for i=1..N
  for v=V..0
    f[v]=max{f[v],f[v-w[i]]+c[i]};
```

其中 f[v]=max{f[v],f[v-w[i]]+c[i]}相当于转移方程 f[i][v]=max{f[i-1][v],f[i-1][v-w[i]]+c[i]}，*这里 f[v]就相当于二维数组的 f[i][v]。因为现在的 f[v-w[i]]就相当于原来的 f[i-1][v-w[i]]。如果将 v 的循环顺序从上面的逆序改成顺序的话，那么则成了 f[i][v]由 f[i][v-w[i]]推知，与本题意不符，但它却是另一个重要的完全背包问题最简捷的解决方案。

把二维转化为一维可以理解为把二维表格压缩成一维。二维表格相当于把把所有值都输出，一维表格只保留更新的值。

f[][]	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	6	6	6	6	6	6	6	6	6
2	0	0	6	6	6	6	6	9	9	9	9
3	0	0	6	6	6	6	11	11	11	11	11
4	0	0	6	6	10	10	11	11	15	15	15
5	0	0	6	6	10	12	12	16	16	17	17



把二维更新为一维的代码为：

```
#include<bits/stdc++.h>
#define maxn 1005
using namespace std;
```

```
int w[maxn],v[maxn],dp[maxn];
int n,W,m;
int main(){
    cin>>m;
    while(m--){
        memset(dp,0,sizeof dp);
        cin>>n>>W;
        for(int i=1;i<=n;i++)
            cin>>w[i]>>v[i];
        for(int i=1;i<=n;i++){
            for(int j=__1_____ W;j>=w[i];j--)
                dp[j]=__2_____ max(dp[j],dp[j-w[i]]+v[i]);
        }
        cout<<dp[W];
    }
    return 0;
}
```

认真思考：为什么转化为一维的时候需要倒过来循环 **for(int j=W;j>=w[i];j--)**？

4 [2819] 01 背包问题

一个旅行者有一个最多能装 M 公斤的背包，现在有 n 件物品，它们的重量分别是 W1，W2，...,Wn 它们的价值分别为 C1,C2,...,CnC1,C2,...,Cn，求旅行者能获得最大总价值。

输入

第一行：两个整数，M(背包容量，M≤200)和 N(物品数量，N≤30)；

第 2..N+1 行：每行二个整数 Wi，Ci，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

10 4

2 1

3 3

4 5

7 9

样例输出

12

二维数组求解：

```
#include<bits/stdc++.h>
using namespace std;
int c[1005][1005];
int w[1001],v[1001];
```

```
int main ()
{
   int m,n;
   cin>>m>>n;
   memset(c,0,sizeof(c));
   for(int i=1;i<=n;i++)
    cin>>w[i]>>v[i];
   for(int i=1;i<=n;i++)
    for(int j=1;j<=m;j++)
    {
      if(j>=w[i])
        c[i][j]=max(c[i-1][j],c[i-1][j-w[i]]+v[i]);
      else
        c[i][j]=c[i-1][j];
    }
   cout<<c[n][m];
   return 0;
}
```

一维数组求解：

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
  int dp[222],w[33],v[33];
  int n,m;
  while(cin>>m>>n)
  {
     memset(dp,0,sizeof(dp));
     for(int i=0;i<n;i++)
      cin>>w[i]>>v[i];
     for(int i=0;i<n;i++)     //简单的 01 背包。。
      for(int j=m;j>=w[i];j--)
         dp[j]=max(dp[j],dp[j-w[i]]+v[i]);
     cout<<dp[m]<<endl;
  }
  return 0; }
```

5 [3783] 集装箱装载

有一批共 n 个集装箱要装上艘载重量为 c 的轮船，其中集装箱 i 的重量为 wi。找出一种最优装载方案，将轮船尽可能装满，即在装载体积不受限制的情况下，将尽可能重的集装箱装上轮船。

输入

第一行有 2 个正整数 n 和 c。n 是集装箱数，c 是轮船的载重量。第 2 行中有 n 个正整数，表示集装箱的重量（0<n<10000,0<c<32767)。

输出
计算出的最大装载重量输出。

样例输入
5 10
7 2 6 5 4
样例输出
10

```
#include<stdio.h>
#include<string.h>
#include<algorithm>
using namespace std;
int a[10010],dp[40000];
int main(){
   int n,c;
   while(scanf("%d %d",&n,&c)!=EOF){
      memset(a,0,sizeof(a));
      memset(dp,0,sizeof(dp));
      for(int i=0;i<n;i++){
         scanf("%d",&a[i]);
      }
      for(int i=0;i<n;i++){
         for(int j=c;j>=a[i];j--){
            dp[j]=max(dp[j],dp[j-a[i]]+a[i]);
         }
      }
      printf("%d\n",dp[c]);
   }
   return 0;
}
```

15.6 [2965]采药

辰辰是个天资聪颖的孩子，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？

输入第一行有两个整数 T（1 <= T <= 1000）和 M（1 <= M <= 100），用一个空格隔开，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 到 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。

输出包括一行，这一行只包含一个整数，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

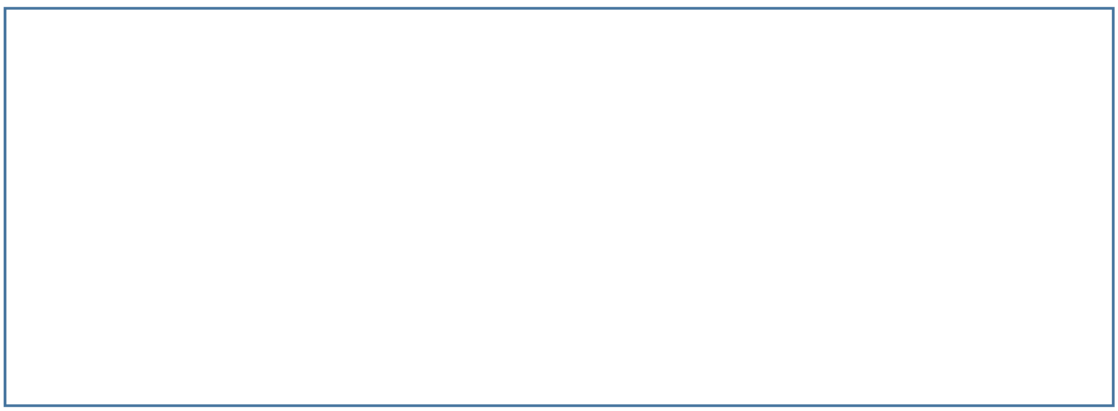
69 1

1 2

样例输出

3

```
#include <bits/stdc++.h>
using namespace std;
int w[105],val[105];
int dp[1005];
int main()
{
    int n,m;
    cin>>m>>n;
    for(int i=1;i<=n;i++)
    {
        cin>>w[i]>>val[i];
    }
```



```
    cout<<dp[m]<<endl;

    return 0;
}
```

10.6 [6922] zb 的生日西瓜

今天是阴历七月初五，acm 队员 zb 的生日。zb 正在和 C 小加、never 在武汉集训。他想给这两位兄弟买点什么庆祝生日，经过调查，zb 发现 C 小加和 never 都很喜欢吃西瓜，而且一吃就是一堆的那种，zb 立刻下定决心买了一堆西瓜。当他准备把西瓜送给 C 小加和 never 的时候，遇到了一个难题，never 和 C 小加不在一块住，只能把西瓜分成两堆给他们，为了对每个人都公平，他想让两堆的重量之差最小。每个西瓜的重量已知，你能帮帮他么？

输入

多组测试数据（<=1500）。数据以 EOF 结尾
第一行输入西瓜数量 N (1 ≤ N ≤ 20)
第二行有 N 个数，W1, …, Wn (1 ≤ Wi ≤ 100000)分别代表每个西瓜的重量
输出
输出分成两堆后的质量差
样例输入
5 5 8 13 27 14
样例输出
3
思考：如何转化为 01 背包？

课堂作业列表

2819　01 背包问题
3783　集装箱装载
2965　采药
3096　开心的金明
3539　NASA 的食物计划
1934　搬寝室
3784　小飞侠的游园方案

背包习题

```
1. #include <iostream>
using namespace std;
void swap(int & a, int & b)
{ int t;
  t = a; a = b; b = t;}
int main()
{
	int a1, a2, a3, x;
	cin>>a1>>a2>>a3;
	if (a1 > a2) swap(a1, a2);
	if (a2 > a3) swap(a2, a3);
	if (a1 > a2)swap(a1, a2);
	cin>>x;
	if (x < a2)
		if (x < a1)
			cout<<x<<' '<<a1<<' '<<a2<<' '<<a3<<endl;
		else
			cout<<a1<<' '<<x<<' '<<a2<<' '<<a3<<endl;
	else
		if (x < a3)
			cout<<a1<<' '<<a2<<' '<<x<<' '<<a3<<endl;
```

```
        else
            cout<<a1<<' '<<a2<<' '<<a3<<' '<<x<<endl;
    return 0;
}
```

输入：

91 2 20

77

输出：_____

2. 0-1 背包的状态转移方程是什么：f[i][v]=_____

f[v]=_____

[2965]采药 辰辰是个天资聪颖的孩子，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？

输入第一行有两个整数 T（1 <= T <= 1000）和 M（1 <= M <= 100），用一个空格隔开，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 到 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。

输出包括一行，这一行只包含一个整数，表示在规定的时间内，可以采到的草药的最大总价值。

```
#include <iostream>
using namespace std;
int dp[1005];//背包结果
int v[105];//草药价值
int w[105];//采摘草药所花时间
int main()
{
  int t,n;
  cin>>t>>n;
  for(int i=1;i<=n;i++)
    ___1_____
  for(int i=1;i<=n;i++) //遍历药品种类
  {
    for(_2_____) //将拥有时间从大到小遍历（类比背包容量）
      //如果采摘这颗草药能获得更大收益，则更新结果
      ___3_____
  }
  cout<<dp[t];
  return 0;
  }
```

第十五讲 背包综合

1 [3539]NASA 的食物计划

航天飞机的体积有限,当然如果载过重的物品,燃料会浪费很多钱,每件食品都有各自的体积、质量以及所含卡路里,在告诉你体积和质量的最大值的情况下,请输出能达到的食品方案所含卡路里的最大值,当然每个食品只能使用一次.

输入

第一行两个数体积最大值(<400)和质量最大值(<400)

第二行 一个数 食品总数 N(<50).

第三行一第 3+N 行

每行三个数 体积(<400) 质量(<400) 所含卡路里(<500)

输出

一个数 所能达到的最大卡路里(int 范围内)

样例输入

320 350 4 160 40 120 80 110 240 220 70 310 40 400 22

样例输出

550

航天飞机的体积有限，当然如果载过重的物品，燃料会浪费很多钱,每件食品都有各自的体积、质量以及所含卡路里，在告诉你体积和质量的最大值的情况下，请输出能达到的食品方案所含卡路里的最大值，当然每个食品只能使用一次.

本题是变形的 01 背包，该题目中体积最大值和质量最大值都属于“背包容量”，即选择物品的花费分为两个方面。结果应用二维数组储存结果，做法上在正常 01 背包循环中两层循环遍历容量即可。

```
#include<iostream>
using namespace std;
int main(){
    int v,w,n;
    int a[50],b[50],c[50];
    int f[401][401];
    scanf("%d%d%d",&v,&w,&n);
    for(int i=0;i<n;i++)
        scanf("%d%d%d",&a[i],&b[i],&c[i]);
    for(int i=0;i<n;i++)
        for(int j=__1_____________________)
            for(int k=__2_____________________)
                f[j][k]=__3_____________________);
    printf("%d",f[v][w]);
    return 0;
}
```

2 [6922] zb 的生日西瓜

今天是阴历七月初五，acm 队员 zb 的生日。zb 正在和 C 小加、never 在武汉集训。他想给这两位兄弟买点什么庆祝生日，经过调查，zb 发现 C 小加和 never 都很喜欢吃西瓜，而且一吃就是一堆的那种，zb 立刻下定决心买了一堆西瓜。当他准备把西瓜送给 C 小加和 never 的时候，遇到了一个难题，never 和 C 小加不在一块住，只能把西瓜分成两堆给他们，为了对每个人都公平，他想让两堆的重量之差最小。每个西瓜的重量已知，你能帮帮他么？

输入

多组测试数据（<=1500）。数据以 EOF 结尾

第一行输入西瓜数量 N (1 ≤ N ≤ 20)

第二行有 N 个数，W1, …, Wn (1 ≤ Wi ≤ 100000)分别代表每个西瓜的重量

输出

输出分成两堆后的质量差

样例输入

5 5 8 13 27 14

样例输出

3

把两堆西瓜分成差值最小的两堆,就可以把问题转化为 01 背包问题。

把背包的最大容量设为西瓜的总质量的二分之一,然后计算背包最大能装多少。

```
for(int i=1;i<=n;i++){
    for(int j=sum/2;j>=a[i];j--){
        dp[j]=max(dp[j],dp[j-a[i]]+a[i]);
    }
}
```

```
#include <bits/stdc++.h>
using namespace std;
int dp[5000005];
int main(){
    int n,sum=0;
    int a[1005];
    int v=0,w1,w2;
    while(cin>>n){
        v=0,sum=0;
        memset(dp,0,sizeof(dp));
        for(int i=1;i<=n;i++) {
            cin>>a[i];
            sum+=a[i];
        }
        v=sum/2;
        for(int i=1;i<=n;i++)
            for(int __1_____________________   )
                dp[j]=__2_____________________);
         w1=__3_______,w2=__4_________________;
```

```
        printf("%d\n",abs(w1-w2));
    }
    return 0;
}
```

搜索写法：

```
#include <iostream>
#include <cstdio>
#include <string.h>
using namespace std;
int a[21], v, n, m;
void dfs(int i, int ans)
{
    if (ans>v) return;//超过一半退出
    if (i>n)
    {//判断完毕
        if (m<ans) m = ans;
        return;
    }//对于每个西瓜两种选择，取和不取
    dfs(i+1, ans+a[i]);
    dfs(i+1, ans);
}
int main()
{
    int sum;
    while (~scanf("%d", &n))
    {
        sum = 0;
        for (int i=1; i<=n; i++)
        {
            scanf("%d", &a[i]);
            sum += a[i];
        }
        v = sum/2;//把一半体积作为边界
        m = 0;
        dfs(1, 0);
         cout<<sum-2*m<<endl;
    }
    return 0;
}
```

3 [2820]完全背包

有 N 种物品和一个容量为 V 的背包，每种物品都有无限件可用。第 i 种物品的费用是 c[i]，价值是 w[i]。

求解将哪些物品装入背包可使这些物品的费用总和不超过背包容量，且价值总和最大。

先看问题：在 n 种物品中选取若干件（同一种物品可多次选取）放在空间为 v 的背包里，每种物品的体积为 c1，c2，…，cn，与之相对应的价值为 w1,w2，…，wn.求解怎么装物品可使背包里物品总价值最大看完这个问题，你也许会觉得这个不就是 01 背包的升级版吗，其实就是这样，完全背包问题与 01 背包问题的区别在于完全背包每一件物品的数量都有无限个，而 01 背包每件物品数量只有 1 个。所以说与它相关的策略已经不是只有取和不取这两种策略了，而是有取 0 件、取 1 件、取 2 件……等等很多种策略

如果我们用和 01 背包一样的状态，f[i][v]表示前 i 种物品恰放入一个容量为 v 的背包的最大价值，那我们应该用 k 表示当前容量下可以装第 i 种物品的件数，那么 k 的范围应该是 0≤k≤v/c[i]，既然要用当前物品 i 把当前容量装满，那需要 0≤k≤v/c[i]件，其中 k 表示件数。

下面给出状态转移方程：

f[i][j] = max{f[i-1][v],f[i-1][v - k * c[i]] + k * w[i]}(0<=k*c[i]<=v)

我们可以对其进行优化：如果有两件物品 a、b 满足 c[a]<=c[b]且 w[a]>=w[b]，则将物品 b 去掉，不用考虑。因为你可以用 占用体积小的物品 得到 比 占用体积大的物品还要多的价值。对于完全背包，可以再优化，首先将容量大于 v 的物品去掉，然后排序计算出容量相同的物品中价值最高的是哪个，我们只要价值大的就可以了。

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=v;j++){
        for(int k=0;k*c[i]<=j;k++){
            if(c[i]<=j)//如果能放下
                f[i][j]=max(f[i-1][j],f[i-1][j-k*c[i]+k*w[i]]);
                //要么不取，要么去 0 件，1 件，2 件…
            else//如果放不下
                f[i][j]=f[i-1][j];
        }
    }
}
```

我们再进行优化，改变一下思路，让 f[i][j]表示在前 i 种物品中选取若干件物品放入容量为 j 的背包所得的最大价值。所以说，对于第 i 件物品有放或不放两种情况，而放的情况里又分为放 1 件、2 件、……v/c[i]件。如果不放那么 f[i][j]=f[i-1][j]；如果确定放，那么当前背包中应该出现至少一件第 i 种物品，所以 f[i][j]中至少应该出现一件第 i 种物品,即 f[i][j]=f[i][j-c[i]]+w[i]，为什么会是 f[i][j-c[i]]+w[i]？而不是 f[i-1][j-c[i]]+w[i]。如果我们用的是 f[i-1][j-c[i]]，f[i-1][j-c[i]]的意思是说，我们只有一件当前物品 i，所以我们在放入物品 i 的时候需要考虑到第 i-1 个物品的价值(f[i-1][j-c[i]])；但是现在我们有无限件当前物品 i，我们不用再考虑第 i-1 个物品了，我们所要考虑的是在当前容量下是否再装入一个物品 i，而[j-c[i]]的意思是指要确保 f[i][j]至少有一件第 i 件物品，所以要预留 c[i]的空间来存放一件第 i 种物品。换一种说法，放在当前物品 i，可以放在 0 件、1 件、2 件…放在下一件物品也是物品 i，而不是物品 i-1。此所以说状态转移方程为：

```
f[i][j]=max(f[i-1][j],f[i][j-c[i]]+w[i])
```

```
for(int i=1;i<=n;i++){
    for(int j=1;j<=v;j++){
            if(c[i]<=j)//如果能放下
                f[i][j]=max(f[i-1][j],f[i][j-c[i]]+w[i]);
            else//如果放不下
                f[i][j]=f[i-1][j];
        }
```

```
    }
}
```

我们可以继续优化此算法，用一维数组写。假设用 f[j]表示当前可用体积 j 的价值，我们可以得到和 01 背包一样的递推式：

```
        f[j] = max(f[j],f[j-c[i]]+w[i])

for(int i=1;i<=n;i++){
    for(int j=c[i];j<=v;j++){
            if(c[i]<=j)//如果能放下
                f[j]=max(f[[j],f[j-c[i]]+w[i]);
    }
}
```

对于 01 背包来说，是逆序，装不装当前物品 i 取决于第 i-1 个物品(f[i][]只和 f[i-1][]有关)；而对于完全背包来说，装不装物品取决于他前一个物品 i(因为可以放无数个物品 i)，因为当前物品 i 是无限的，不用去考虑 f[i-1][]，而应当考虑当前状态下是否应当再装入当前一个物品 i。或者换一种说法：完全背包考虑的是第 i 种物品的出现的问题(该不该放入当前物品 i)，第 i 种物品一旦出现它势必应该对第 i 种物品还没出现的各状态造成影响，也就是说，原来没有第 i 种物品的情况下可能有一个最优解，现在第 i 种物品出现了，而它的加入有可能得到更优解，所以之前的状态需要进行改变，故需要正序。

也就是说完全背包是说，在当前体积下，是否要放入或者再放入一个当前物品 i，而 01 背包是说，在当前体积下，是否要放入一个当前物品 i。

[2820]设有 n 种物品，每种物品有一个重量及一个价值。但每种物品的数量是无限的，同时有一个背包，最大载重量为 M，今从 n 种物品中选取若干件(同一种物品可以多次选取)，使其重量的和小于等于 M，而价值的和为最大。

输入

第一行：两个整数，M(背包容量，M≤200)和 N(物品数量，N≤30)；

第 2..N+1 行：每行二个整数 Wi,Ci，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

```
10 4
2 1
3 3
4 5
7 9
```

样例输出

```
max=12
```

```
[#include <iostream>
using namespace std;
int main()
{
    int m,n;
```

```
    int a[55],b[55];
    int dp[1005];
    while(cin>>m>>n)
    {
        int i,j;
        for(i=0;i<n;i++)
            cin>>a[i]>>b[i];
        memset(dp,0,sizeof(dp));
        for(i=0;i<n;i++)
        {
            for(j=a[i];j<=m;j++)
            {
                dp[j]=max(dp[j],dp[j-a[i]]+b[i]);
            }
        }
       printf("max=%d\n",dp[m]);
    }
    return 0;
}
```

4 [6917]竞赛总分

学生在我们 USACO 的竞赛中的得分越多我们越高兴。
我们试着设计我们的竞赛以便人们能尽可能的多得分,这需要你的帮助。
我们可以从几个种类中选取竞赛的题目,这里的一个"种类"是指一个竞赛题目的集合,解决集合中的题目需要相同多的时间并且能得到相同的分数。你的任务是写一个程序来告诉 USACO 的职员,应该从每一个种类中选取多少题目,使得解决题目的总耗时在竞赛规定的时间里并且总分最大。输入包括竞赛的时间,M(1 <= M <= 10,000)(不要担心,你要到了训练营中才会有长时间的比赛)和 N,"种类"的数目 1 <= N <= 10,000。后面的每一行将包括两个整数来描述一个"种类":
第一个整数说明解决这种题目能得的分数(1 <= points <= 10000),第二整数说明解决这种题目所需的时间(1 <= minutes <= 10000)。
你的程序应该确定我们应该从每个"种类"中选多少道题目使得能在竞赛的时间中得到最大的分数。
来自任意的"种类"的题目数目可能是任何非负数(0 或更多)。
计算可能得到的最大分数。
输入
第 1 行: M, N--竞赛的时间和题目"种类"的数目。
第 2-N+1 行: 两个整数:每个"种类"题目的分数和耗时。
输出
单独的一行包括那个在给定的限制里可能得到的最大的分数。
样例输入

```
300 4
100 60
250 120
120 100
```

35 20
样例输出
605

这是一道典型的完全背包的题目 。与 01 背包不同，完全背包可以反复的取一件物品。因此在递推的时侯，最外层的循环用来控制物品顺序 ，内层循环用来控制取物品的上限 。与 01 背包不同在于，完全能背包可以反复取一件物品，因此在内层循环上，需要从物品的单件价值开始进行累加，直到背包放不下为止。通过对内层循环对每个物品取一件取两件的每一状态进行枚举，外层循环控制物品的顺序，最后求得整个问题的最优解。

```
#include <bits/stdc++.h>
using namespace std;
int dp[10005],n,t,w[10005],c[10005];
int main(){
    cin>>t>>n;
    for(int i=1;i<=n;i++)
        cin>>c[i]>>w[i];
    for(int i=1;i<=n;i++){
        for(int j=w[i];j<=t;j++)
            dp[j]=max(dp[j],dp[j-w[i]]+c[i]);
    }
    cout<<dp[t];

    return 0;
}
```

5 [7328]纪念品

小伟有一种超能力，他知道未来 T 天 N 种纪念品每天的价格。某个纪念品的价格是指购买一个该纪念品所需的金币数量，以及卖出一个该纪念品换回的金币数量。
每天，小伟可以进行以下两种交易无限次：
1．任选一个纪念品，若手上有足够金币，以当日价格购买该纪念品；
2．卖出持有的任意一个纪念品，以当日价格换回金币。
每天卖出纪念品换回的金币可以立即用于购买纪念品，当日购买的纪念品也可以当日卖出换回金币。当然，一直持有纪念品也是可以的。
T 天之后，小伟的超能力消失。因此他一定会在第 T 天卖出所有纪念品换回金币。
小伟现在有 M 枚金币，他想要在超能力消失后拥有尽可能多的金币。
【输入】
第一行包含三个正整数 T, N, M，相邻两数之间以一个空格分开，分别代表未来天数 T，纪念品数量 N，小伟现在拥有的金币数量 M。
接下来 T 行，每行包含 N 个正整数，相邻两数之间以一个空格分隔。第 i 行的 N 个正整数分别为 Pi,1,Pi,2,……,Pi,NPi,1,Pi,2,……,Pi,N，其中 P_{i,j} 表示第 i 天第 j 种纪念品的价格。
【输出】
输出仅一行，包含一个正整数，表示小伟在超能力消失后最多能拥有的金币数量。

【输入样例】
6 1 100
50
20
25
20
25
50
【输出样例】
305
【提示】
【输入输出样例 1 说明】
最佳策略是：
第二天花光所有 100 枚金币买入 5 个纪念品 1；
第三天卖出 5 个纪念品 1，获得金币 125 枚；
第四天买入 6 个纪念品 1，剩余 5 枚金币；
第六天必须卖出所有纪念品换回 300 枚金币，第四天剩余 5 枚金币,共 305 枚金币。
超能力消失后，小伟最多拥有 305 枚金币。
【输入输出样例 2】
3 3 100
10 20 15
15 17 13
15 25 16
【输入输出样例 2】
217
【输入输出样例 2 说明】
最佳策略是：
第一天花光所有金币买入 10 个纪念品 1；
第二天卖出全部纪念品 1 得到 150 枚金币并买入 8 个纪念品 2 和 1 个纪念品 3，剩余 1 枚金币；
第三天必须卖出所有纪念品换回 216 枚金币，第二天剩余 1 枚金币，共 217 枚金币。
超能力消失后，小伟最多拥有 217 枚金币。

5.1 问题分析

这题不太好理解，不过可以断定是 DP 题，而且是背包题。做如下考虑：
1、我不买
2、我买完 第二天就卖
3、我买完过几天卖

若只考虑前两种，我们就可以把题目转换成当天花多少钱，第二天赚了多少钱，即追求每一天的最大升值，也就分成 T 个完全背包问题。

那如何表示第三种呢，也很简单。比方说我买了 3 天卖，因为当天价格不变所以，我完全可以分为 3 天的单独买卖。这样一定会被完全背包所包含。那么这样一来就是个简单的完全背包问题。对每天进行完全背包就行，之后 m+=f[m]，改变背包总容量。

5.2 代码分析

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e2+7;
int t,n,m,price[N][N];
int f[10001],dis[N];
void solve(int x)
{
    memset(f,0,sizeof(f));//f[i]表示背包容量为 i 时的价值
    int today=x,yesterday=x-1;
    for(int i=1; i<=n; i++)
        dis[i]=price[today][i]-price[yesterday][i]; //得到物品 i 今天和昨天的价值差
    for(int i=1; i<=n; i++) {
    //对 n 个物品跑一遍完全背包
        for(int j=price[yesterday][i]; j<=m; j++) {
            if(f[j-price[yesterday][i]]+dis[i]>=f[j])
            { //如果入昨天的物品并且在今天卖出能盈利就更新背包
                f[j]=f[j-price[yesterday][i]]+dis[i];
            }
       }
    }
    m+=f[m];//加上今天的利润
}
int main() {
  cin>>t>>n>>m;
  for(int i=1;i<=t;i++) {
     for(int k=1;k<=n;k++)
        cin>>price[i][k];
     }
  for(int i=2;i<=t;i++)
     solve(i);//得到每天的利润
  cout<<m<<endl;
}
```

第十六讲 递归

重点:

1. 理解递归思想。
2. 掌握经典汉诺塔问题。

递归是一种重要的算法思想。递归既可以实现递推过程，也是实现求解诸多问题如搜索等问题的通用思路。一般来说，能够用递归解决的问题应该满足以下三个条件：

- 需要解决的问题可以转化为一个或多个子问题来求解，而这些子问题的求解方法与原问题完全相同，只是在数量规模上不同。
- 递归调用的次数必须是有限的。
- 必须有结束递归的条件来终止递归。

1 [2749]汉诺塔问题

汉诺塔问题是递归问题的一个经典案例。约 19 世纪末，在欧州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

输入为一个整数(小于 20）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->3->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

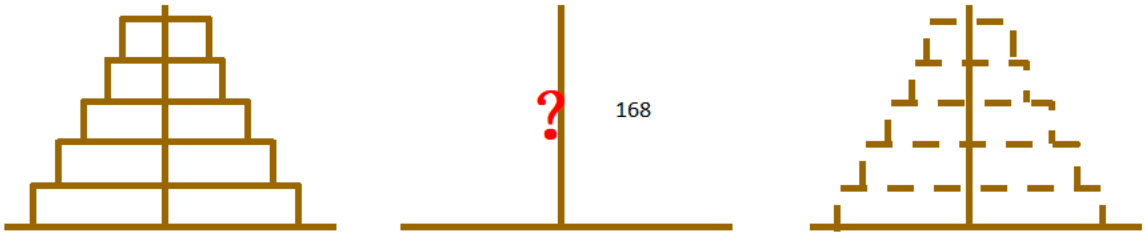
a->1->c

a->2->b

c->1->b

1.1 分析

盘片移动时必须遵守以下规则：每次只能移动一个盘片；盘片可以插在 X、Y 和 Z 中任一塔座；任何时候都不能将一个较大的盘片放在较小的盘片上。



如果盘子数量很少，通过心算便可以直接想出移动盘子的具体步骤。
比如：只有 2 个盘子的情况，此问题就变得相当的简单，只需要 3 步就可以完成整个移动操作。
A—>B（表示将 A 柱 最上面的 1 个盘子移到 B 柱 上）
A—>C（将 A 此时最上面的 1 个盘子移到 C 上）
B—>C（将 B 上的盘子移到 C 上）

假设现在有 64 个盘子，显然是不可能心算的，我们采用递归方式解决。
递归的结束条件：
最后 1 个人（第 64 个人）只需要移动 1 个盘子。
注意：
第 1 个人完成任务的前提是：第 2 个人完成了任务
第 2 个人完成任务的前提是：第 3 个人完成了任务
……
第 63 个人完成任务的前提是：第 64 个人完成了任务
所以：
只有当第 64 个人完成任务后，第 63 个人才能完成任务；只有第 2~64 个人完成任务后，第 1 个人才能完成任务！

将 n 个盘子从 A 移到 C 可以分解为以下 3 个步骤：
将 A 上 n-1 个盘子借助 C 先移到 B 上；
把 A 上剩下的 1 个盘移到 C 上；
将 n-1 个盘从 B 借助 A 移到 C 上。
上面第 1 步和第 3 步，都是把 n-1 个盘子从 1 个座移到另 1 个座上， 采用的办法是相同的，只是座的名字不同而已。为使之一般化，可以将第 1 步和第 3 步表示为：
将 one 座上 n-1 个盘子移到 two 座（借助 three 座）。
只是在第 1 步和第 3 步中，one、two、three 和 A、B、C 的对应关系不同。
对第 1 步，对应关系是：one—A，two—B，three—C 。
对第 3 步，对应关系是：one—B，two—C，three—A 。
因此，将上面 3 个步骤分为 2 类操作：
将 n-1 个盘子从 1 个座移到另 1 个座上 （当 n>1 时）；
将最后 1 个盘子从 1 个座上移到另 1 个座上 。

1.2 设计程序

分别用 2 个函数实现以上 2 类操作：
设计 hanoi 函数实现第 1 类操作；
函数 hanoi(n, one, two, three) 将实现把 n 个盘子从 one 座借助 two 座移到 three 座的过程；
设计 move 函数实现第 2 类操作；

函数 move(x, y) 将实现把 1 个盘子从 x 座移到 y 座的过程。x 和 y 是代表 A、B、C 座之一，根据每次不同情况分别以 A、B、C 代入。

```
#include<iostream>
using namespace std;
void hanno(int n,char one ,char two ,char three){
  if(___1_____)
    //直接把盘子从 one 移动到 three a->1->c
    cout<<one<<"->"<<n<<"->"<<two<<endl;
  else{
    hanno(__2_____);//one->two 借助 three
    cout<<one<<"->"<<n<<"->"<<two<<endl;
    hanno(___3_____);
  }
}
int main(){
  int n;
  char a,b,c;
  cin>>n>>a>>b>>c;
  hanno(n,a,b,c);
  return 0;
}
```

2 [2998]汉诺塔 1

Hanoi 塔由 n 个大小不同的圆盘和三根木柱 a,b,c 组成。开始时，这 n 个圆盘由大到小依次套在 a 柱上，如图 1 所示。要求把 a 柱上 n 个圆盘按下述规则移到 c 柱上：

(1)一次只能移一个圆盘；

(2)圆盘只能在三个柱上存放；

(3)在移动过程中，不允许大盘压小盘。

问将这 n 个盘子从 a 柱移动到 c 柱上，总计需要移动多少个盘次？

输入 n，输出需要移动多少个盘次

样例输入

3

样例输出

7

分析递归算式：f(n)=_____

```
#include<bits/stdc++.h>
using namespace std;
int f(int n)
{
        if(n==1)
```

```
        return 1;
        else
        return ___1_____________________;
}
int main()
{
        int n;
        cin>>n;
        cout<<f(n);
        return 0;
}
```

3 [2867] 集合的划分

设 S 是一个具有 n 个元素的集合，S＝⟨a1，a2，……，an⟩S＝⟨a1，a2，……，an⟩，现将 S 划分成 k 个满足下列条件的子集合 S1，S2，……，SkS1，S2，……，Sk，且满足：

1．Si ≠ Ø

2．Si ∩ Sj ＝ Ø　　(1≤i，j≤k　i≠j)

3．S1 ∪ S2 ∪ S3 ∪ … ∪ Sk ＝ S

则称 S1，S2，……，SkS1，S2，……，Sk 是集合 S 的一个划分。它相当于把 S 集合中的 n 个元素 a1，a2，……，ana1，a2，……，an 放入 k 个(0＜k≤n＜30)无标号的盒子中，使得没有一个盒子为空。请你确定 n 个元素 a1，a2，……，ana1，a2，……，an 放入 k 个无标号盒子中去的划分数 S(n,k)。

输入：给出 n 和 k。

输出：n 个元素 a1，a2，……，ana1，a2，……，an 放入 k 个无标号盒子中去的划分数 S(n,k)。

样例输入

10 6

样例输出

22827

3.1 分析

先举个例子，设 S＝｛1，2，3，4｝，k＝3，不难得出 S 有 6 种不同的划分方案，即划分数 S(4，3)=6，具体方案为：

｛1，2｝∪｛3｝∪｛4｝　｛1，3｝∪｛2｝∪｛4｝　｛1，4｝∪｛2｝∪｛3｝

｛2，3｝∪｛1｝∪｛4｝　｛2，4｝∪｛1｝∪｛3｝　｛3，4｝∪｛1｝∪｛2｝

考虑一般情况，对于任意的含有 n 个元素 a1 ，a2，……，an 的集合 S，放入 k 个无标号的盒子中去，划分数为 S(n，k)，我们很难凭直觉和经验计算划分数和枚举划分的所有方案，必须归纳出问题的本质。其实对于任一个元素 an，则必然出现以下两种情况：

1、｛an｝是 k 个子集中的一个，于是我们只要把 a1，a2，……，an-1 划分为 k－1 子集，便解决了本题，这种情况下的划分数共有 S(n－1，k－1)个；

2、{an}不是 k 个子集中的一个，则 an 必与其它的元素构成一个子集。则问题相当于先把 a1，a2，……，an-1 划分成 k 个子集，这种情况下划分数共有 S(n－1，k)个；然后再把元素 an 加入到 k 个子集中的任一个中去，共有 k 种加入方式，这样对于 an 的每一种加入方式，都可以使集合划分为 k 个子集，因此根据乘法

原理，划分数共有 k * S(n－1，k)个。

综合上述两种情况，应用加法原理，得出 n 个元素的集合｛a1，a2，……，an｝划分为 k 个子集的划分数为以下递归公式：S(n，k)＝S(n－1，k－1) + k * S(n－1，k) (n>k，k>0)。

下面，我们来确定 S(n，k)的边界条件,首先不能把 n 个元素不放进任何一个集合中去，即 k=0 时，S(n，k)＝0；也不可能在不允许空盒的情况下把 n 个元素放进多于 n 的 k 个集合中去，即 k＞n 时,S(n，k)＝0；再者，把 n 个元素放进一个集合或把 n 个元素放进 n 个集合，方案数显然都是 1，即 k=1 或 k=n 时，S(n,k)=1。

因此，我们可以得出划分数 S(n，k)的递归关系式为：

S(n，k)＝S(n－1，k－1) + k * S(n－1，k)　　(n>k，k>0)

S(n，k)＝0　　(n<k)或(k＝0)

S(n，k)＝1　　(k=1)或(k＝n)

3.2 代码

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

int s(int n, int k)                //数据还有可能越界，请用高精度计算
{
      if ((n < k) || (k == 0)) return 0;   //满足边界条件，退出
      if ((k == 1) || (k == n)) return 1;
      return s(n-1,k-1) + k * s(n-1,k);     //调用下一层递归
}
int main()
{
    int n,k;
      cin >> n >> k;
      cout << s(n,k);
      return 0;
}
```

4 [2745]凑数字

有 n 个数字，a[1],a[2],a[3]……a[n]，以及一个数字 m。问 n 个数字中取出一些数字，这些数字的和能否等于 m。

输入

多组测试数据，读入到文件尾结束。

第一行输入 n，m。（1<=n<=20, 1<=m<=100）

第二行输入 n 个数字 a[1],a[2],a[3]……a[n]。(1<=a[i]<=100)

输出

如果可以，输出 YES，否则输出 NO。

样例输入

5 5
1 1 1 1 1
5 5
2 2 2 2 2
样例输出
YES
NO

分析：

```
#include<stdio.h>
#include<string.h>
int n,m,a[20],flag,sum;
void f(int i)
{

    if(i>=n)
        return;
    sum+=a[i];
    if(sum==m)
    {
        flag=1;
        return;
    }
    f(i+1);
    sum-=a[i];
    f(i+1);
}
int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        memset(a,0,sizeof a);
        int i;
        for(i=0;i<n;i++)
            scanf("%d",&a[i]);
        flag=0,sum=0;
        f(0);
        if(flag)
            printf("YES\n");
        else
            printf("NO\n");
    }
```

```
    return 0;
}
```

课堂作业列表

[2749]汉诺塔问题
[2998]汉诺塔 1
[3860]汉诺塔 II
[3860]汉诺塔 II
[2764] 放苹果
[2867] 集合的划分
[2745] 凑数字
[2723] 阿克曼(Ackmann)函数

递归综合习题

1. 以下程序的运行结果是_____

```
fun(int k)
{ if(k>0) fun(k-1);
   printf("%d ",k);
}
int main()
{ int w=5;fun(w);printf("\n");}
```

2. [2014]

```
#include <iostream>
using namespace std;
int fun(int n)
{
    if(n == 1)
    return 1;
    if(n == 2)
    return 2;
    return fun(n -2) - fun(n - 1);
}
int main()
{
    int n;
    cin >> n;
    cout << fun(n) << endl;
```

```
        return 0;
}
```
输入：7
输出：_____ _____

3. 有以下程序

```
int fun(int n)
{
if(n==1) return 1;
else
return (n+fun(n-1));
}
main()
{
int x;
scanf("%d",&x); x=fun(x); printf("%d\n",x);
}
```
程序执行时，给变量 x 输入 10，程序的输出结果是 _____

3. 阅读程序，选择程序的运行结果_____

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

int try1(int n)
{
    if(n>0)
          return(n*try1(n-2));
    else
        return(1);
}
int main()
{
   int x;
   x=try1(5);
   printf("%d\n", x);
}
```

5．[2011]

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

int solve(int n,int m)
{
```

```
    int i,sum;
    if(m==1) return 1;
    sum=0;
    for(i=1;i<n;i++)
        sum+= solve(i,m-1);
    return sum;
}

int main()
{
    int n,m;
    cin>>n>>m;
    cout<<solve(n,m)<<endl;
    return 0;
}
```

输入：7 4
输出：_____

6. [2749]: 2749: 将汉诺塔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。

输入为一个整数(小于 20）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->3->b 的形式，即把编号为 3 的盘子从 a 杆移至 b 杆。

```
#include<iostream>
using namespace std;
void hanno(int n,char one ,char two ,char three){
  if(n==1)
        cout<<one<<"->"<<n<<"->"<<two<<endl;
  else{
    hanno( 1_____ );
    cout<<one<<"->"<<n<<"->"<<two<<endl;
    hanno( 2_____);

  }
}
int main(){
  int n;
  char a,b,c;
  cin>>n>>a>>b>>c;
  __3_____
  return 0;
}
```

样例输入
2 a b c
样例输出
a->1->c
a->2->b
c->1->b

第十七讲 搜索基础

重点：

1. 掌握深度优先搜索的基本思想和算法框架。
2. 熟练应用深度优先搜索求解一些实际问题。

搜索算法是利用计算机的高性能来有目的地穷举一个问题的部分或所有的可能情况，从而求出问题的解的一种方法。很多问题无法根据某种确定的计算法则来求解，可以利用搜索与回溯的技术求解。回溯是搜索算法中的一种控制策略。它的基本思想是：为了求得问题的解，先选择某一种可能情况向前探索，在探索过程中，一旦发现原来的选择是错误的，就退回一步重新选择，继续向前探索，如此反复进行，直至得到解或证明无解。

搜索过程实际上是根据初始条件和扩展规则构造一棵状态集合，并在集合中寻找符合目标状态节点的过程。常见的搜索算法主要有暴力枚举法、深度优先搜索(DFS)与回溯、广度优先搜索（BFS)、双向广度优先搜索等。

枚举算法是最直接的搜索方法，其思想是列举问题的所有状态，然后从中找出问题的解；深度优先搜索法是从初始结点开始扩展，按照某中顺序不断的向下扩展,直到找到目标状态或者是无法继续扩展；广度优先搜索是从初始状态开始，通过规则来生成第一层结点，同时检查生成结点中是否有目标结点，若没有则生成下一层接点,并检查是否有目标结点…

1 [3369] 全排列问题

输出自然数 1~n 所有不重复的排列，即 n 的全排列，要求所产生的任一数字序列中不允许出现重复数字。

[输入格式]

1<=n<=9

[输出格式]

由 1~n 组成的所有不重复的数字序列。每行一个序列

[输入样例]

3

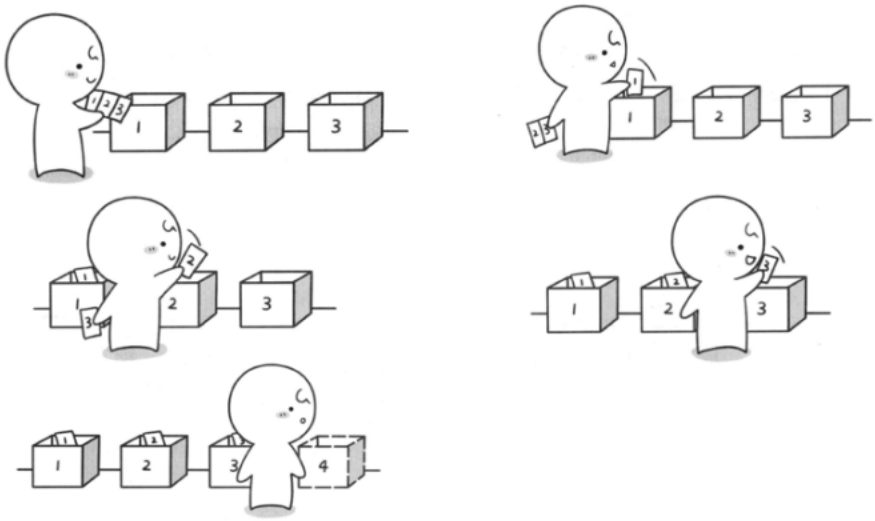
[输出样例]

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

1.1 问题分析

把 n 的全排列理解为把 n 张牌放入 n 个盒子里不同的放置种数。根据样例数据，需要将这 3 张扑克牌

分别放到 3 个盒子里面，并且每个盒子有且只能放一张扑克牌，那么一共有多少种不同的放法呢？



如图所示，小哼来到第一个盒子面前，先把 1 放入第一个盒子，然后来到第二个盒子；接着把 2 放入第二个盒子，然后来到第三个格子；把 3 放到第三个盒子后，来到虚拟的第四个格子面前，这是一家完整一次放置。

产生了一种排列之后小哼需要立即返回。现在小哼需要退一步重新回到 3 号盒子面前，需要取回之前放在 3 号盒子中的扑克牌，再去尝试看看还能否放别的扑克牌，从而产生一个新的排列。于是小哼取回了 3 号扑克牌。当小哼再想往 3 号盒子放别的扑克牌的时候，却发现手中仍然只有 3 号扑克牌，没有别的选择。于是小哼不得不再往回退一步，回到 2 号盒子面前。小哼回到 2 号盒子后，收回了 2 号扑克牌。现在小哼手里面有两张扑克牌了，分别是 2 号和 3 号扑克牌。按照之前约定的顺序，现在需要往 2 号盒子中放 3 号扑克牌（上一次放的是 2 号扑克牌）。放好之后小哼又向后走一步，再次来到了 3 号盒子面前。小哼再次来到 3 号盒子后，将手中仅剩的 2 号扑克牌放入了 3 号盒子。又来到 4 号盒子面前。当然了，这里并没有 4 号盒子。此时又产生了一个新的排列“132”。接下来按照刚才的步骤去模拟，便会依次生成所有排列：“213”、“231”、“3 12”和“321”。

具体操作过程如下：

c.1 往第 i 号盒子放扑克牌

```
for(int i=1;i<=n;i++){
        a[step]=i;  //将 i 号扑克牌放到 step 号盒子中
 }
```

c.2 往第 i 号盒子牌前判断该牌是否存在并记录

```
for(i=1;i<=n;i++)
{
    if(book[i]==0)//如果 i 号扑克牌还在手上
    {
        a[step]=i;//将 i 号扑克牌放到 step 号盒子中
        book[i]=1;//i 号扑克牌已经不在手中
    }
}
```

C.3 进入下一轮搜索

```
for(i=1;i<=n;i++)
{
    if(book[i]==0)//如果 i 号扑克牌还在手上
    {    ......
        dfs(step+1);//处理下一个盒子，即“下一步和这一步是一样的”
        book[i]=0;//收回放入的扑克牌，保证正常返回
    }
}
```

1.2 代码讲解

```
#include <stdio.h>
#include <stdlib.h>
int a[10],book[10],n;//a 数组表示盒子，book 数组为标记数组，n 为数字个数
void dfs(int step)
{
    int i;
    if(1_____)
            //判断边界，如果到了第 n+1 个盒子，就说明前 n 个盒子都已经放好
    {
        for(i=1;i<=n;i++)
        {
            printf("%d ",a[i]); //打印一个排列
        }
        printf("\n");
        return;//返回上一层
    }

    for(i=1;i<=n;i++)
    {//如果未到达边界，就开始尝试每一种可能
        if(2_____)//如果 i 号扑克牌还在手上
        {
            3_____;//将 i 号扑克牌放到 step 号盒子中
            4_____;//i 号扑克牌已经不在手中
            5_____
                    //处理下一个盒子，即“下一步和这一步是一样的”
            6_____;//收回放入的扑克牌，保证正常返回
        }
    }
    return;//循环结束也返回上一层
}
int main()
{
    scanf("%d",&n);
```

```
    dfs(1);
    return 0;
}
```

2 搜索框架

分析全排列问题，可以归纳出深度优先搜索可以采用如下框架：

```
int Search(int k)
{
    if  (到目的地) {
        输出解;
        return;
    }
    for (i=1;i<=算符种数;i++){
      if  (满足条件)
        {
            保存结果;
            Search(k+1);
            恢复：保存结果之前的状态{回溯一步}
        }
    }
}
```

“到目的地”对应“完成一次全排列”；“算符种数”对应“每一个位置有 n 种选择”；“满足条件”对应“第 i 号扑克牌还在手上，即第 i 张牌没有被用过”；“保存结果”对应“记录把第 i 牌放入了编号为 step 的盒子里以及记录第 i 牌被使用过”；“恢复回溯”对应“收回放入的扑克牌”。

在搜索框架中，也常常会加入判断不满足条件就会结束当前这次搜索，提前进入下一轮搜索的代码，如：

```
int Search(int k)
{
    if  (到目的地) {
        输出解;
        return;
    }
    If(条件不满足)
        continue;
    for (i=1;i<=算符种数;i++){
      if  (满足条件)
        {
            保存结果;
            Search(k+1);
            恢复：保存结果之前的状态{回溯一步}
        }
    }
}
```

如下框架，是另一种深度优先搜索的常见写法：

```
int Search(int k)
{
  for (i=1;i<=可能搜索的位置总数;i++){
    if (满足条件)
    {
        保存结果
        if (到目的地) 输出解;
        else Search(k+1);
        恢复：保存结果之前的状态{回溯一步}返回到上一步的状态
     }
  }
}
```

3 [3302]素数环

素数环:从 1 到 n 这 n 个数摆成一个环，要求相邻的两个数的和是一个素数。如，n=8 是，素数环为：

1 2 3 8 5 6 7 4

1 2 5 8 3 4 7 6

1 4 7 6 5 8 3 2

1 6 7 4 3 8 5 2

总数为 4

3.1 问题分析

搜索的目的地是什么？ _____

搜索的“满足条件”时什么？ _____

从 1 开始，每个空位有 n 种可能，只要填进去的数与前面的数不相同、与左边相邻的数的和是一个素数就可以，最后判断第 n 个数与第 1 个数的和是否是素数，具体流程如下：

1. 数据初始化，每个数字可用；为了避免出现重复，第一个位置固定为 1；

2. 递归填数：判断第 i 个数填入是否合法；

a.如果合法：填数；判断是否到达目标（n 个已填完）：是，打印结果；不是，递归填下一个；

b.如果不合法：选择下一种可能；

以 n=6 加以解释

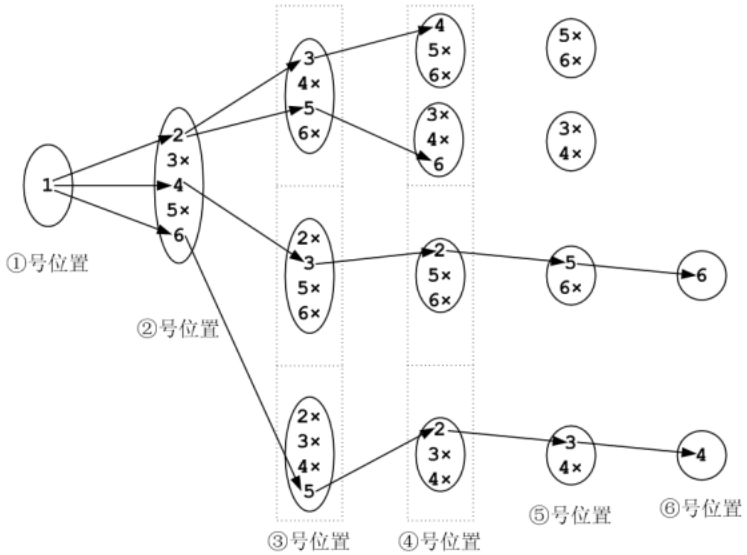
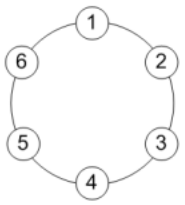
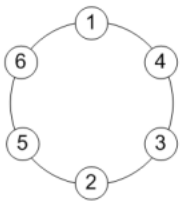


图8.19 素数环搜索策略(n=6)



(a) 环上的n个位置



(b) 在n个位置放置自然数1~n

在①号位置上放置数 1。在②号位置上可供选择的数为 2~6，其中 3 和 5 是不可行的，对 2、4 和 6 一一试探，先试探 2。

②号位置上放置 2 以后，③号位置上可供放置的数为 3~6，其中 4 和 6 是不可行的，对 3 和 5 也是一一试探，先试探 3。③号位置上放置 3 以后，④号位置上可供放置的数为 4~6，其中只有 4 是可行的，所以放置 4。这时可供⑤号位置上放置的数为 5 和 6，均不可行。这些方案都不可行。

再考虑③号位置上放置 5，④号位置上只能放置 6，此后⑤号位置上放置 3 和 4，均不可行。这些方案都排除。

再考虑②号位置上放置 4，③号位置上只能放置 3，④号位置上只能放置 2，⑤号位置上只能放置 5，⑥号位置只能放置 6，这个方案是可行的。

…

如此搜索，直到试探所有方案为止。具体实现只要理解题意后按照框架编码即可。

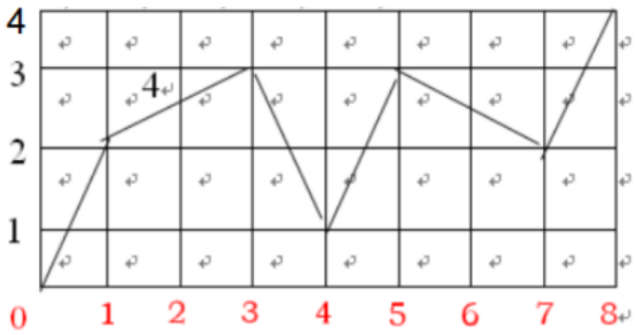
3.2 代码详解

```
#include<bits/stdc++.h>
using namespace std;
bool used[100]={0};//放在 n 个位置上的数；  =0 表示当前数字 i 没有用过
int a[100]={0},n;//使用第 1~n 个元素，每个数是否被选用的标志  a[i]表示第 i 个位置放置的值
int sum=0;
int isPrime(int a,int b){
    int c=a+b;//判断是否是素数
    if(c==2) return 1;
    for(int i=2;i*i<=c;i++){
        if(c%i==0) return 0;
    }
    return 1;
}
void search(int k){//k 表示处理第几个数
    if(1_____){
        sum++;
        return;
    }
    for(int i=2_____){
        if(3_____)){
            a[k]=i;
            4_____;//记录状态
            search(k+1);
```

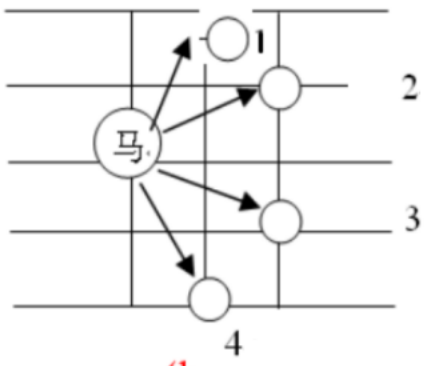
```
            5_____//回溯
        }
    }

}
int main()
{
    cin>>n;
    6_____;
    search(2);
    cout<<sum<<endl;
    return 0;
}
```

4 [3306]马的遍历

中国象棋半张棋盘如图所示。马自左下角往右上角跳。今规定只许往右跳，不许往左跳。比如图（a）中所示为一种跳行路线，路线为：0,0->2,1->3,3->1,4->3,5->2,7->4,8...



(a)



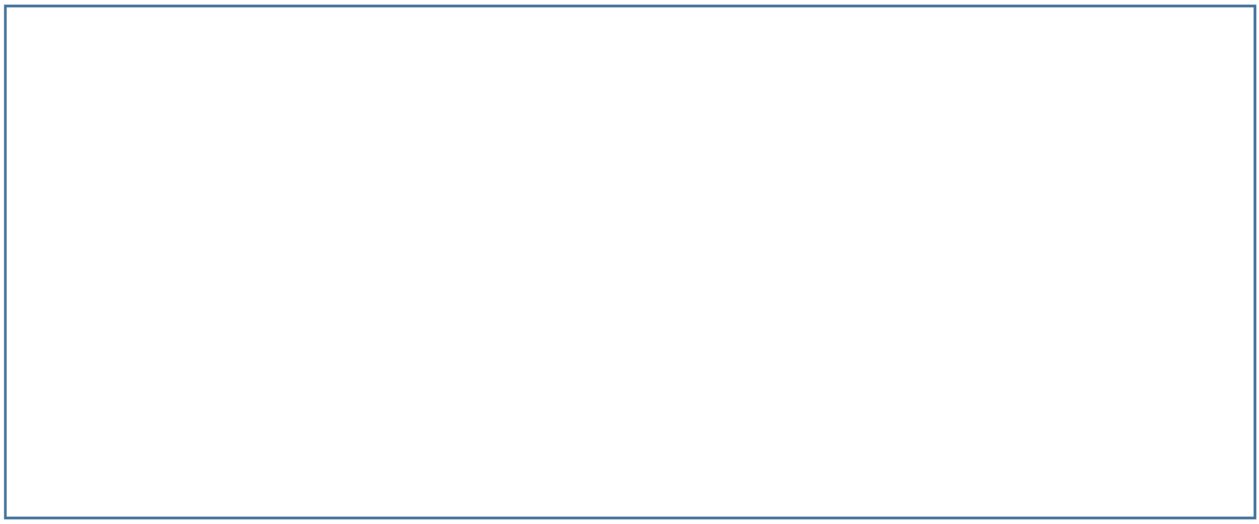
(b)

输出
输出上述棋盘的可以从（0,0）走到（4,8）点的方案数。

搜索的目的地是什么？ _____
搜索的“满足条件”时什么？ _____

```
#include<iostream>
#include<cstring>
using namespace std;
int sum=0;
int dir[4][2]={{1,-2},{2,-1},{2,1},{1,2}};
int vis[9][5];
int isOk(int x,int y){//条件判断
  if(x<=8&&y<=4&&x>=0&&y>=0&&vis[x][y]==0)
    return 1;
```

```
    else
        return 0;
}
void search(int x, int y){
```



```
    for(int i=0;i<4;i++){
        x=x+dir[i][0],y=y+dir[i][1];
        if(isOk(x,y)){
            vis[x][y]=1;//保存结果
            if(x==8&&y==4)sum++;//if (到目的地) 输出解;
            else search(x,y);//else Search(k+1);
            vis[x][y]=0;//恢复
        }
        x=x-dir[i][0],y=y-dir[i][1];

    }

}
int main(){
    memset(vis,0,sizeof(vis));
    search(0,0);
    cout<<sum<<endl;
    return 0;
}
```

5 [2868]组合的输出

排列与组合是常用的数学方法，其中组合就是从 n 个元素中抽出 r 个元素(不分顺序且 r≤n)，我们可以简单地将 n 个元素理解为自然数 1，2，…，n，从中任取 r 个数。
现要求你用递归的方法输出所有组合。
例如 n＝5，r＝3，所有组合为：
1 2 3　1 2 4　1 2 5　1 3 4　1 3 5　1 4 5　2 3 4　2 3 5　2 4 5　3 4 5
输入
一行两个自然数 n、r(1<n<21，1≤r≤n)。
输出
所有的组合，每一个组合占一行且其中的元素按由小到大的顺序排列，每个元素占三个字符的位置，所有的组合也按字典顺序。
样例输入
5 3

搜索的目的地是什么？ _____
搜索的“满足条件”时什么？ _____

```
include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
int a[100],b[100],n,r;
void print()
{
   int j;
   for (j=1;j<=r;++j)
     printf("%3d",a[j]);
   printf("\n");
   return;
}

void dfs(int dep)
{
   int i;
   for (i=a[dep-1]+1;i<=n;++i)//因为是组合，所以 i 从上一个选定的数+1 开始循环，完成去重工作
    if (!b[i])//判断有没有被访问过
    {
      a[dep]=i;//a 是记录组合的数组
      b[i]=1;
      if (dep==r) print();//如果组合中的数的数量达到要求，则输出
      else dfs(dep+1);
      b[i]=0;//回溯一步
```

```
    }
}
int main()
{
   scanf("%d%d",&n,&r);
   dfs(1);
   return 0;
}
```

6 [2869]自然数的拆分

2869:任何一个大于 1 的自然数 n，总可以拆分成若干个小于 n 的自然数之和。
当 n=7 共 14 种拆分方法：
7=1+1+1+1+1+1+1
7=1+1+1+1+1+2
7=1+1+1+1+3
7=1+1+1+2+2
7=1+1+1+4
7=1+1+2+3
7=1+1+5
7=1+2+2+2
7=1+2+4
7=1+3+3
7=1+6
7=2+2+3
7=2+5
7=3+4
total=14
输入
输入 n。
输出
按字典序输出具体的方案。
样例输入
7

```
#include<cstdio>
#include<iostream>
#include<cstdlib>
using namespace std;
int a[10001]={1},n,total;
int search(int,int);
int print(int);
int search(int s,int t)
{
```

```
    int i;
    for (i=a[t-1];i<=s;i++)
     if (i<n)//当前数 i 要大于等于前 1 位数，且不过 n
      {
        a[t]=i;                                  //保存当前拆分的数 i
        s-=i;                                    //s 减去数 i，  s 的值将继续拆分
        if (s==0) print(t);          //当 s=0 时，拆分结束输出结果
          else search(s,t+1); //当 s>0 时，继续递归
        s+=i;                                    //回溯：加上拆分的数，以便
         //   产分所有可能的拆分
      }
}
int print(int t)
{
    cout<<n<<"=";
    for (int i=1;i<=t-1;i++)
//输出一种拆分方案
       cout<<a[i]<<"+";
    cout<<a[t]<<endl;
    total++;
//方案数累加 1
}
int main()
{
    cin>>n;
    search(n,1);
          //将要拆分的数 n 传递给 s
    cout<<"total="<<total<<endl;
          //输出拆分的方案数
}
```

7 [2997]数塔

有如下所示的数塔，要求从顶层走到底层，若每一步只能走到相邻的结点，则经过的结点的数字之和最大是多少？

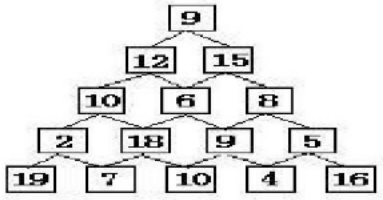
输入数据首先包括一个整数C,表示测试实例的个数,每个测试实例的第一行是一个整数N(1 <= N <= 20)，表示数塔的高度，接下来用N个数字表示数塔，其中第i行有个i个整数，且所有的整数均在区间[0,99]内。

对于每个测试实例，输出可能得到的最大和，每个实例的输出占一行。

样例输入

1

5 9 12 15 10 6 8 2 18 9 5 19 7 10 4 16



写法 1：

```
#include<iostream>
using namespace std;
int a[25][25];
int n,ans;
void dfs(int x,int y, int curr){ //(x,y)表示的位置，curr 表示的是当前位置对应的权重和
  if(x==n){//到达目标，也就是最后一行的时候
    if(curr>ans)ans=curr;//求最大的权值和
    return;//1 求解完一轮，返回到上一层
  }
  dfs(x+1,y,curr+a[x+1][y]);
  dfs(x+1,y+1,curr+a[x+1][y+1]);
}
int main(){
  cin>>n;//scanf("%d",&n);
  for(int i=1;i<=n;i++)
    for(int j=1;j<=i;j++)
      cin>>a[i][j];
  ans=0;
  dfs(1,1,a[1][1]);
  cout<<"max="<<ans<<endl;//printf("max=%d",ans);
  return 0;
}
```

写法 2：

```
#include <iostream>
using namespace std;
int a[20+1][20+1];
int n;
int dfs(int i,int j)
{
  int suml=0,sumr=0;
  if(i==n+1)
  {
    return 0;
  }
  suml=a[i][j]+dfs(i+1,j);
  sumr=a[i][j]+dfs(i+1,j+1);
  return max(suml,sumr);
}
int main()
{
  cin>>n;
  for(int i=1;i<=n;i++)
```

```
    {
        for(int j=1;j<=i;j++)
        {
            cin>>a[i][j];
        }
    }
    int res=dfs(1,1);
    cout<<"max="<<res<<endl;
    return 0;
}
```

写法 3:

```
#include <iostream>
using namespace std;
int a[20+1][20+1],max_sum=0;
void dfs(int n,int step,int j,int sum)
{
    if(step==n+1)
    {
        if(sum>max_sum)
        {
            max_sum=sum;
        }
        return ;
    }
    for(int i=0;i<=1;i++)
    {
        dfs(n,step+1,j+i,a[step+1][j+i]+sum);
    }
}

int main()
{
    int n;
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=i;j++)
        {
            cin>>a[i][j];
        }
    }
    dfs(n,1,1,a[1][1]);
    printf("max=%d",max_sum);
```

```
   return 0;
}
```

课堂作业列表

[3306]马的遍历　[3302]素数环 [3303]排列
[2868]组合的输出 [2869]自然数的拆分
[2997]数塔　[2747] 判断元素是否存在　[2757]移动路线

搜索习题

1. 执行下面程序，输出是 _____

```
#include<stdio.h>
int x;
int f(int n)
{
    int x = 1;
    return x;
}
int main(void)
{
    printf("%d %d", f(3),x);
    return 0;
}
```

2. 以下程序的运行结果是_____

```
fun(int k)
{ if(k>0) fun(k-1);
   printf("%d ",k);
}
int main()
{ int w=5;fun(w);printf("\n");}
```

```
3[2004]#include <stdio.h>
char c[3][200];
int s[10], m, n;
void numara(){
        int i, j, cod, nr;
for (j = 0; j < n; j++){
                nr = 0; cod = 1;
                for (i = 0; i < m; i++){
```

```
                if (c[i][j] == '1'){
                    if (!cod){cod = 1; s[nr]++; nr = 0;}
                }
                else{
                    if (cod){nr = 1; cod = 0;}
                    else nr++;
                }
            }
            if (!cod) s[nr]++;
        }
}
int main(){
        int i;
        scanf("%d %d\n", &m, &n);
        for (i = 0; i < m; i++) gets(c[i]);
        numara();
        for (i = 1; i <= m; i++)
if (s[i] != 0) printf("%d %d ", i, s[i]);
        return 0;
}
```

输入：

```
3 10
1110000111
1100001111
1000000011
```

输出：1 4 2 1 3 3

5. 搜索框架

```
int Search(int k)
  {
    if  (1                    ) 输出解;
    else
        for (  2                  )
          if  (满足条件)
            {
              保存结果;
                3
              恢复：保存结果之前的状态{回溯一步}
            }
  }
```

6. 从 1 到 20 这 20 个数摆成一个环，要求相邻的两个数的和是一个素数。

```
#include<bits/stdc++.h>
```

```
using namespace std;
bool b[100]={0},a[100]={0},n;
int sum=0;
bool f(int a,int b){
    int c=a+b;
    if(c==2) return 1;
    for(int i=2;i*i<=c;i++){
        if(__1__) return 0;
    }
    return 1;
}
int search(int k){
    for(int i=2;i<=n;i++){
        if(f(__2__)){
            __3__    //记录状态
            if(__4__){//到达目的题
                if(f(a[n],a[1])) sum++;
            }
            else search(k+1);
            __5__    //回溯
        }
    }
 }
int main()
{
cin>>n;
   a[1]=1;
   search(2);
   cout<<sum<<endl; return 0;}
```

第十八讲 DFS 搜索应用*

重点：

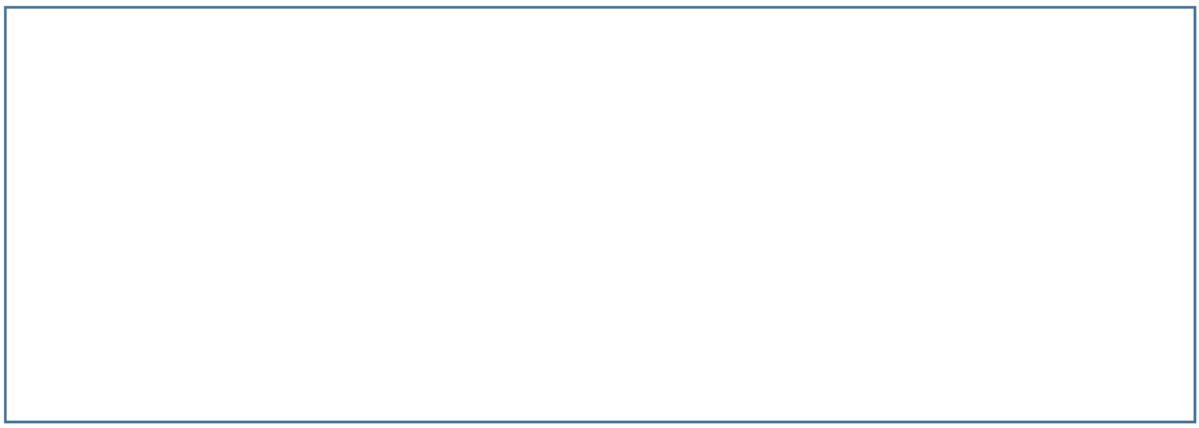
1. 熟练应用深度优先搜索求解一些实际问题。
2. 理解 n 皇后问题求解。

1 [2747] 判断元素是否存在

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) 除了上述二种情况外，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 100000，如果是，则输出 YES，否则，输出 NO。

输入
输入整数 k 和 x, 逗号间隔。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

```
#include<iostream>
using namespace std;
int jie(int x,int y)
{
```



```
    if(x==y)return 1;
    if(x>y)return 0;
    if(jie(x*2+1,y))return 1;
    if(jie(x*3+1,y))return 1;
```

```
    return 0;
}
int main()
{
    int a,b;
    scanf("%d,%d",&a,&b);
    if(jie(a,b))cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
    return 0;
}
```

2 [2757]移动路线

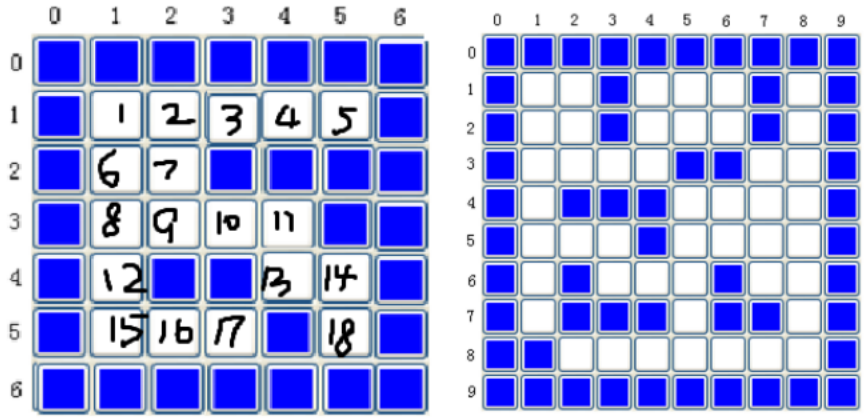
X 桌子上有一个 m 行 n 列的方格矩阵，将每个方格用坐标表示，行坐标从下到上依次递增，列坐标从左至右依次递增，左下角方格的坐标为(1,1)，则右上角方格的坐标为(m,n)。
小明是个调皮的孩子，一天他捉来一只蚂蚁，不小心把蚂蚁的右脚弄伤了，于是蚂蚁只能向上或向右移动。小明把这只蚂蚁放在左下角的方格中，蚂蚁从
左下角的方格中移动到右上角的方格中，每步移动一个方格。蚂蚁始终在方格矩阵内移动，请计算出不同的移动路线的数目。
对于 1 行 1 列的方格矩阵，蚂蚁原地移动，移动路线数为 1；对于 1 行 2 列（或 2 行 1 列）的方格矩阵，蚂蚁只需一次向右（或向上）移动，移动路线数也为 1……对于一个 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) → (1,2) → (1,3) → (2,3)
路线 2：(1,1) → (1,2) → (2,2) → (2,3)
路线 3：(1,1) → (2,1) → (2,2) → (2,3)
输入
输入只有一行，包括两个整数 m 和 n（0 < m+n ≤ 20），代表方格矩阵的行数和列数，m、n 之间用空格隔开。
输出
输出只有一行，为不同的移动路线的数目。
样例输入
2 3
样例输出
3

```
int m,n;
int sum=0;
void dfs(int x,int y){

if(x>m||y>n)return;//不出边界
    if(x==m&&y==n){//到达终点
        sum=sum+1;
        return;
    }
```

```
    dfs(x+1,y);//搜索
    dfs(x,y+1);
}
int main(){
    cin>>m>>n;
    dfs(1,1);
    cout<<sum<<endl;
    return 0;
}
```

3 [3727]迷宫问题

给定一个 M×N 的迷宫图，求从指定入口（1,1）到出口（M,N）有多少种走法。例如迷宫图如图所示（M=10，N=10），其中的方块图表示迷宫。对于图中的每个方块，用空白表示通道，用阴影表示墙。要求所求路径必须是简单路径，即在求得的路径上不能重复出现同一通道块。



如图所示，蓝色的表示墙壁，白色的表示可走的方块，题目求解从入口位置 1（坐标（1,1））出发，到达位置 18（坐标（5,5））。

3.1 问题分析

1. 确定搜索"到达目的地"的条件。假设(xe,ye)表示迷宫出口位置，则到达迷宫出口的代码可以表示成：
   ```
   if(xe==m&&ye==n){sum++;//根据题目做相关操作，本题到达目的题做求和操作}
   ```
2. 确定搜索方向。对于每一个方块（位置）而言，共有四种走法：上下左右。对于计算机而言所有的操作都是有序的，因此需要规定搜索过程中，寻路的顺序，如上下左右，左右上下等，本解题规定采用的搜索顺序为左上右下，定义方向数组：
   ```
   int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
   ```
3. 记录搜索过程中走过位置的状态。当一个位置被走过以后，为了避免来回重复走，需要记录当前的位置有没有被走过，如果被走过，这个位置就不能再走。假设用 vis[][]数组记录位置的状态，初始时 vis 数组的值全部初始化为 0，vis[x][y]=0 **表示该位置没有走过，vis[i][j]=1 表示该位置已经被走过。**实现代码如下：
   ```
   vis[x][y]=1;//用 1 表示该位置没有被走过
   ```

4. 确定"满足条件"。在什么状态下可以继续搜索下一个位置呢？很显然就是下一个位置是可走的并且是没有被走过的

3.2 代码详解

```
#include<iostream>
#include<cstring>
using namespace std;
int a[12][12];//定义迷宫
int dir[4][2]={{-1,0},{0,1},{1,0},{0,-1}};//定义方向
int vis[12][12];//标记有没有走过
int sum=0,xe,ye;//xe,ye 终点
void dfs(int x,int y){
    if(x==xe&&y==ye){
        sum++;//到达目的地
    }
    else{
        for(int i=0;i<4;i++){
            int xx=x+dir[i][0];//下一个点
            int yy=y+dir[i][1];
            if(a[xx][yy]==0&&vis[xx][yy]==0){
                vis[xx][yy]=1;//记录走过
                dfs(xx,yy);
                vis[xx][yy]=0;//退格，下一次还可以走
            }
        }
    }
}
int main(){
    int m,n,i,j;
    cin>>m>>n;
    xe=m,ye=n;
    memset(a,1,sizeof(a));
    for(i=1;i<=m;i++)
        for(j=1;j<=n;j++)
            cin>>a[i][j];

    vis[1][1] = 1;//记录第一点没访问过
    dfs(1,1);
    cout<<sum<<endl;
    return 0;
}
```

4 [3232]棋盘

有一个 m × m 的棋盘，棋盘上每一个格子可能是红色、黄色或没有任何颜色的。你现在要从棋盘的最左上角走到棋盘的最右下角。

任何一个时刻，你所站在的位置必须是有颜色的（不能是无色的）， 你只能向上、 下、左、 右四个方向前进。当你从一个格子走向另一个格子时，如果两个格子的颜色相同，那你不需要花费金币；如果不同，则你需要花费 1 个金币。

另外， 你可以花费 2 个金币施展魔法让下一个无色格子暂时变为你指定的颜色。但这个魔法不能连续使用， 而且这个魔法的持续时间很短，也就是说，如果你使用了这个魔法，走到了这个暂时有颜色的格子上，你就不能继续使用魔法； 只有当你离开这个位置，走到一个本来就有颜色的格子上的时候，你才能继续使用这个魔法，而当你离开了这个位置（施展魔法使得变为有颜色的格子）时，这个格子恢复为无色。现在你要从棋盘的最左上角，走到棋盘的最右下角，求花费的最少金币是多少？

数据的第一行包含两个正整数 m， n，以一个空格分开，分别代表棋盘的大小，棋盘上有颜色的格子的数量。

接下来的 n 行，每行三个正整数 x， y， c， 分别表示坐标为（ x， y）的格子有颜色 c。

其中 c=1 代表黄色， c=0 代表红色。 相邻两个数之间用一个空格隔开。 棋盘左上角的坐标为（ 1,1），右下角的坐标为（ m,m）。

棋盘上其余的格子都是无色。保证棋盘的左上角，也就是（ 1， 1） 一定是有颜色的。

输出

输出一行，一个整数，表示花费的金币的最小值，如果无法到达，输出-1。

样例输入

```
5 7
1 1 0
1 2 0
2 2 1
3 3 1
3 4 0
4 4 1
5 5 0
```

样例输出

```
8
```

```
#include <cstdio>
#include <cstring>
#include <cmath>
using namespace std;
#define inf 0x7fffffff
int fx[4] = {-1, 0, 1, 0}; // x 偏移量
int fy[4] = {0, -1, 0, 1}; // y 偏移量
int f[110][110]; // 存储每个点的最优解
int mp[110][110]; // 存储初始棋盘
int m, n, ans = inf;
```

```
void dfs(int x, int y, int sum, bool frog)
{
        if(x < 1 || y < 1 || x > m || y > m) return; // 超出边界
        if(sum >= f[x][y]) return; // 不符合最优解
                // 以上两种情况剪枝
        f[x][y] = sum;
        if(x==m && y==m) // 终点，坐标为(m, m)
        {
                if(sum < ans) ans = sum; // 更新最优解
                return;
        }
        for(int i = 0; i < 4; ++i)
        {
                int xx = x + fx[i];
                int yy = y + fy[i];
                if(mp[xx][yy]) // 若下一格有色
                {
                        if(mp[xx][yy] == mp[x][y]) // 若颜色相同
                                dfs(xx, yy, sum, false); // 则不花钱
                        else dfs(xx, yy, sum+1, false); // 否则花费+1
                }       else // 否则（若下一格无色）
                if(!frog) // （且）没有用过魔法
                {
                        mp[xx][yy] = mp[x][y];
                        dfs(xx, yy, sum+2, true); // 使用魔法
                        mp[xx][yy] = 0; // 回溯
                }
        }
}
int main()
{
        memset(f, 0x7f, sizeof(f));
        scanf("%d %d", &m, &n);
        for(int i = 1; i <= n; ++i)
        {
                int x, y, c;
                scanf("%d %d %d", &x, &y, &c);
                mp[x][y] = c + 1;
                // 1 红色 2 黄色 0 无色（未赋值，其初值为 0）
        }
        dfs(1, 1, 0, false);
        if(ans == inf) cout << -1;
        else cout << ans;
        return 0;
```

}

5 [3299] n 皇后问题

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 n 个皇后放在棋盘上（有 n × n 个方格），使它们谁也不能被吃掉！这就是著名的八皇后问题。输入 n 输出，输出 放置的具体方法，以及总共的方法数。

样例输入

4

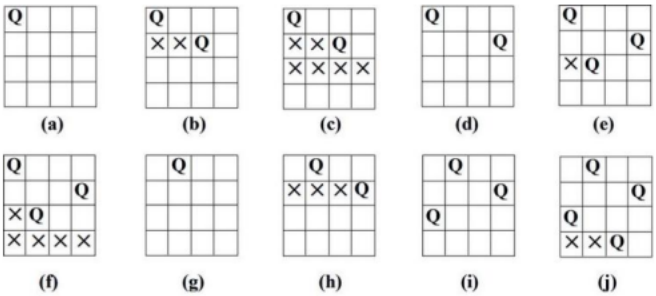
样例输出

2 4 1 3

3 1 4 2

2

20.4.1 问题分析



如图为 4 皇后的放置过程到达 f 后，回溯。解决该问题的关键在于如何判定某个皇后所在的行、列、斜线上是否有别的皇后；可以从矩阵的特点上找到规律。

如果在同一行，则行号相同；如果在同一列上，则列号相同；

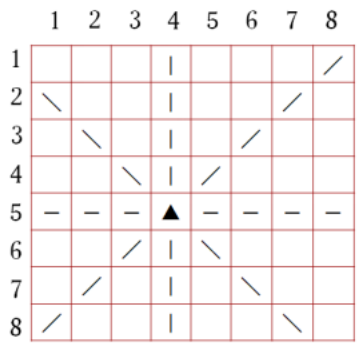
array[i] == array[k]

如果同在 / 斜线上的行列值之和相同；如果同在 \ 斜线上的行列值之差相同；

abs(array[i] - array[k]) == i - k

5.1 代码详解

```
#include<iostream>//八皇后
#include<cstdlib>
using namespace std;
int sum=0;
void Print(int n, int array[]){
  sum++;
  for (int j = 1; j <= n; j++)
    cout<<array[j]<<" ";  //输出最终的棋盘布局，每个数组元素值 表示各行棋盘所在的该列放置皇后
  cout<<endl;
}
bool isOk(int i, int array[]) { //检查当前棋盘布局是否合理
```



```
  for (int k =  1                    )
    if(  2                              )
      return false;
  return true;
}
void search(int i,int n, int array[]) { //进入本函数前，前 i-1 行已经放置了互不攻击的 i-1 个皇后
  if(i > n) Print(n, array);  //如果 i 已经大于最大的行数了，那么说明布局完成，则打印结果
  else {
    for(int j = 1; j <= n; j++) {
        3                  //在第 i 行第 j 列放置一个皇后
      if(isOk(i, array))
              4                  //当前布局合理，进行下一行的布局
        5                  //移走第 i 行第 j 列放置的皇后， 回溯。
    }
  }
}
int main() {
  int n;
  cin>>n;
  int Chess[100] = {0}; //棋盘的初始状态，棋盘上无任何皇后
  search(1, n, Chess); //摆放皇后
cout<<sum<<endl;
return 0;
}
```

6 [3378]八数码

有八个棋子，每个棋子上标有 1 至 8 的某一数字。棋盘中留有一个空格，空格用 0 来表示。空格周围的棋子可以移到空格中。要求解的问题是：给出一种初始布局（初始状态）和目标布局（为了使题目简单，设目标状态为 123804765），找到一种最少步骤的移动方法，实现从初始布局到目标布局的转变。

输入

输入初试状态，一行九个数字，空格用 0 表示

输出

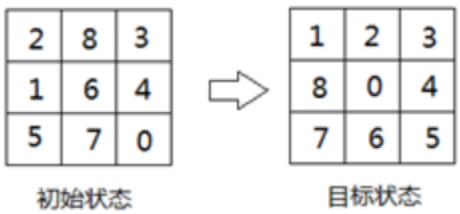
只有一行，该行只有一个数字，表示从初始状态到目标状态需要的最少移动次数(测试数据中无特殊无法到达目标状态数据)

样例输入

283104765

样例输出

4




```
#include<iostream>
#include<cmath>
#include<cstdio>
#include<cstdlib>
```

```
using namespace std;

int now[10];
char c;
int to[10]={0,1,2,3,8,0,4,7,6,5};
int pos0=0,deep=0,h;
int f(){
   int fnow[10];
   int fto[10];
   int tot=0;
   for(int i=1;i<=9;++i){
      fnow[now[i]]=i-1;
      fto[to[i]]=i-1;
   }
   for(int i=1;i<9;++i){
      tot+=abs(fnow[i]/3-fto[i]/3)+abs(fnow[i]%3-fto[i]%3);
   }
   return tot;
}

void dfs(int ans,int pos){

   h=f();
   if(h+ans>deep) return;
   if(ans==deep) {
      cout<<ans<<endl;
      exit(0);
   }
   if(pos!=1&&pos!=4&&pos!=7){
      swap(now[pos],now[pos-1]);
      dfs(ans+1,pos-1);
      swap(now[pos],now[pos-1]);
   }
   if(pos!=3&&pos!=6&&pos!=9){
      swap(now[pos],now[pos+1]);
      dfs(ans+1,pos+1);
      swap(now[pos],now[pos+1]);
   }
   if(pos>3){
      swap(now[pos],now[pos-3]);
      dfs(ans+1,pos-3);
      swap(now[pos],now[pos-3]);
   }
```

```
    if(pos<7){
        swap(now[pos],now[pos+3]);
        dfs(ans+1,pos+3);
        swap(now[pos],now[pos+3]);
    }
}
int main(){
    for(int i=1;i<=9;++i){
        cin>>c;
        now[i]=(int)c-48;
        if(now[i]==0) pos0=i;
    }
    while(1){
        dfs(0,pos0);
        ++deep;
    }

}
```