算法集

- 1适合算法竞赛或兴趣了解
- 2本套算法均是已实现算法
- 3 算法均有本人查阅和自己写
- 4 未经允许不得做为商业用途 **违者必究**

作者——徐一粟(男)

QQ: 964092342

目录

1.	10 进制转 2 进制	4		
2.	啤酒和饮料	5		
3.	圆的面积	-5		
4.	切面条	6		
5.	01 字符串	-7		
6.	字母图形	8		
7.	求 n 个数的最大值,最小值,和	-9		
8.	杨辉三角形	10		
9.	(2个数)公约数,公倍数(三种算法)	11		
10.	歌手大奖赛	-16		
11.	输出斐波那契数列第 n 项的数值	17		
12.	输出斐波那契数列每一项的数值	-18		
13.	Fibonacci 数列其它问题	-19		
14.	求前 n 项的和 1+2++n	-20		
15.	序列求和	21		
16.	图形显示	22		
17.	星期几	22		
18.	16 进制转 10 进制	23		
19.	10 进制转 16 进制	24		
20.	16 进制转 8 进制	25		
21.	判断是否是回文	26		
22.	闰年的判断	27		
23.	输出 c 字母图形	28		
24.	巴斯卡三角形	28		
25.	三色旗	30		
26.	回文数	32		
27.	特殊回文数(普)	32		
28.	特殊回文数(经)	34		
29.	特殊的数字	35		
30.	查找整数	35		
31.	操作格子	36		
32.	高精度阶乘 n!	39		
33.	老鼠走迷宫	40		
34.	逆序对	42		
35.	数列排序	45		
36.	数列排序(经)	45		
37.	第 39 台阶	48		
38.	第 39 台阶(非递归)50			
39.	最短路径 ——Dijkstar 算法	51		
40.	最短路径 ——Floyd 算法	53		

41.	区间 K 大数查询	55
42.	八皇后递归算法	57
43.	八皇后回溯算法	61
44.	八皇后回溯算法 2	62
45.	2n 皇后问题	63
46.	前缀表达式	65
47.	(3个数)最大最小公倍数	66
48.	2 的次幂问题	67
49.	数的全排列问题	69
50.	猴子吃桃	72
51.	角谷定理	72
52.	高斯日记	73
53.	马虎的算式	74
54.	黄金分连数	75
55.	前缀判断	77
56.	三部排序	78
57.	翻硬币	79
58.	李白打酒(递归)	81
59.	李白打酒(二进制)	82
60.	普利姆算法	84
61.	深度遍历	85
62.	广度优先遍历	87
63.	两个物种	89
64.	选手答题	90
65.	比酒量	91
66.	盒子取球方法(一)	92
67.	盒子取球方法(二)	93
68.	大数相乘	95
69.	字母转换为 6 位数字	96
70.	打印图形	98
71.	奇怪的分式	100
72.	六角填数	102
73.	蚂蚁感冒	104
74.	地宫取宝	106
75.	高精度加法	109
76.	Huffman 树	110
77.	报时助手	111
78.	回形取数	113
79.		
80.	芯片测试	116
81.	FJ 的字符串	
82.	Since 之舞	118
	数的读法	119
84.	完美的代价	121

85.	矩形的面积交		·123
86.	矩阵乘法		-125
87.	质因数分解		-126
88.	字符串对比		-128
89.	时间转换		·129
90.	出现次数最多	的整数	129
91.	捉鬼大师		.131

有些算法在你不明白时,最好在稿纸上走一遍,这样可以更好地理解 算法。有些算法可能已优化,有些未优化,但结果是正确的,可能时 间上和空间上有点浪费。

纯属个人整理,如有差错还请见谅!

算法实现

一,将 10 进制转为二进制

```
/*
如输入: 13
输出: 1101
*/
#include<stdio.h>
int fact(int n)
{
    if(n<2)//将 2 换成其它数如 8 就可输出 8 进制的结果
    return n;
    else
    {
        return fact(n/2)*10+n%2;//将二进制结果整个输出
    }
}
int main(void)
{
    int n;
4/133
```

```
printf("Enter n:");
scanf("%d",&n);
printf("%d",fact(n));
return 0;
}
```

二,标题:啤酒和饮料

```
啤酒每罐 2.3 元,饮料每罐 1.9 元。小明买了若干啤酒和饮料,一共花了 82.3
元。
      我们还知道他买的啤酒比饮料的数量少,请你计算他买了几罐啤酒。
      注意: 答案是一个整数。请通过浏览器提交答案。
       不要书写任何多余的内容(例如:写了饮料的数量,添加说明文字等)。
  */
  #include<stdio.h>
  int main(void)
  {
   for(int i=1;i*2.3<82.3;i++)
      for(int j=i+1;i*2.3+j*1.9<=82.3;j++)//因为 i 比 j 小,所以 j 从 i+1 开始
      {
         if(i^2.3+j^1.9>=82.3-0.000001\&i^2.3+j^1.9<=82.3+0.000001)
            printf("%d\n",i);
      return 0;
  }
```

三,圆的面积

```
/*
```

问题描述

给定圆的半径 r, 求圆的面积。

输入格式

输入包含一个整数 r, 表示圆的半径。

输出格式

输出一行,包含一个实数,四舍五入保留小数点后7位,表示圆的面积。

说明: 在本题中,输入是一个整数,但是输出是一个实数。

对于实数输出的问题,请一定看清楚实数输出的要求,比如本题中要求保留小数点后**7**位,

```
则你的程序必须严格的输出 7 位小数,输出过多或者过少的小数位数都是不行的,都会被认为错误。
实数输出的问题如果没有特别说明,舍入都是按四舍五入进行。
```

样例输入 4 样例输出 50.2654825 数据规模与约定 1 <= r <= 10000。 提示

本题对精度要求较高,请注意 π 的值应该取较精确的值。你可以使用常量来表示 π ,比如 PI=3.14159265358979323,也可以使用数学公式来求 π ,比如 PI=atan(1.0)*4。

```
*/
#include<stdio.h>
#define PI 3.14159265358979323
int main(void)
{
    int r;
    double result;
    scanf("%d",&r);
    result=PI*(r*1.0)*(r*1.0);
    printf("%.7f",result);
    return 0;
}
```

四,标题:切面条

```
一根高筋拉面,中间切一刀,可以得到 2 根面条。如果先对折 1 次,中间切一刀,可以得到 3 根面条。如果连续对折 2 次,中间切一刀,可以得到 5 根面条。那么,连续对折 10 次,中间切一刀,会得到多少面条呢?答案是个整数,请通过浏览器提交答案。不要填写任何多余的内容。*/
#include<stdio.h>
int main(void)
{
    int num=2;
    for(int i=1;i<=10;i++)
```

```
{
     num=num*2-1;
     printf("%d : %d\n",i,num);
}
return 0;
}
```

五,01字符串

```
对于长度为 5 位的一个 01 串,每一位都可能是 0 或 1,一共有 32 种可能
。它们的前几个是:
     00000
     00001
     00010
     00011
     00100
     请按从小到大的顺序输出这32种01串。
输入格式
     本试题没有输入。
输出格式
     输出 32 行, 按从小到大的顺序每行一个长度为 5 的 01 串。
*/
常用算法
#include<stdio.h>
int main(void)
{
int count=0;
int i,j,k,l,m;
for(i=0;i<=1;i++)
    for(j=0;j<=1;j++)
       for(k=0;k<=1;k++)
           for(l=0;l<=1;l++)
               for(m=0;m<=1;m++){
                  printf("%d%d%d%d%d\n",i,j,k,l,m);
                  count++;
               printf("count=%d\n",count);
               return 0;
}
经典算法
#include <stdio.h>
int main(void)
7/133
```

六,字母图形

```
ABCDEFG
      BABCDEF
      CBABCDE
      DCBABCD
      EDCBABC
这是一个5行7列的图形,请找出这个图形的规律,并输出一个n行m列的图形。
*/
#include<stdio.h>
#include<stdlib.h>
#define MAXINE 26
void print(char a[MAXINE][MAXINE],int m,int n)
{
    int i,j;
    for(i=0;i<m;i++)
    {
         for(j=0;j<n;j++)
             printf("%c",a[i][j]);
         printf("\n");
    }
}
void creat(char a[MAXINE][MAXINE],int m,int n)//主要是这段代码
{
    int j;
    char str;
    for(int i=0;i<m;i++)
    {
         str='A';
        for(j=i;j<n;j++)
         {
             a[i][j]=str++;
        }
         str='A';
        for(j=i-1;j>=0;j--)
8/133
```

```
a[i][j]=++str;

}

print(a,m,n);

}

int main(void)

{

int n,m;

char a[MAXINE][MAXINE];

printf("Enter m and n(m 行 n 列):");

scanf("%d%d",&m,&n);

creat(a,m,n);

return 0;

}
```

七,求n个数的最大值,最小值,和

```
/*
输入格式
      第一行为整数 n,表示数的个数。
      第二行有 n 个数,为给定的 n 个数,每个数的绝对值都小于 10000。
如输入:
Enter n:5
-159322
输出
max=32
min=-1
sum=47
*/
#include<stdio.h>
#include<stdlib.h>
#define MAXINE 10000
int main(void)
int i,n,min=MAXINE,max=-MAXINE;
int sum=0,a[MAXINE];
printf("Enter n:");
scanf("%d",&n);
for(i=0;i<n;i++)
    scanf("%d",&a[i]);
    if(a[i]<min)
        min=a[i];
```

八,杨辉三角形

/*它的一个重要性质是: 三角形中的每个数字等于它两肩上的数字相加。

下面给出了杨辉三角形的前 4 行:

1

11

121

1331

给出 n,输出它的前 n 行。

输入格式

输入包含一个数 n。

输出格式

*/

输出杨辉三角形的前 n 行。每一行从这一行的第一个数开始依次输出,中间使用一个空格分隔。请不要在前面输出多余的空格

```
#include<stdio.h>
#include<stdib.h>
#define MAXINE 100

int main(void)
{
    int j,i;
    int n;
    int a[MAXINE][MAXINE];
    printf("Enter n:");
    scanf("%d",&n);
    a[0][0]=1;//第一个数置为 1
```

九,输入两个数,求这两个数的最大公约数,及最小公倍数(三种算法)

```
/*
输入:
        Enter a and b:9 27
输出:
       最大公约数是:9
       最小公倍数是:27
*/
#include"stdio.h"
/*方法一 辗转相除法
int main(void)
{
    int m,n,a,b,c;
    printf("Enter m,n:");
    scanf("%d%d",&m,&n);
    a=m,b=n;
    while(n!=0)
    {
        c=m%n;
        m=n;
        n=c;
    printf("最大公约数是:%d\n",m);
    printf("最小公倍数是:%d\n",a*b/m);
    return 0;
11/133
```

```
}
*/
/* 方法二
            相减法
int main(void)
{
    int a,b,m,n;
    printf("Enter a and b:");
    scanf("%d%d",&a,&b);
    m=a,n=b;
    while(a!=b)
         if(a>b)
            a=a-b;
         else
            b=b-a;
    printf("最大公约数是:%d\n",a);//这里 a 或 b 都可以
    printf("最小公倍数是:%d\n",m*n/a);
    return 0;
}
*/
//方法三
             穷举法
int main(void)
{
    int i=1,a,b,t,m,n;
    printf("Enter a and b:");
    scanf("%d%d",&a,&b);
    m=a,n=b;
    while(i<=a)
    {
         if(a%i==0&&b%i==0)
               t=i;
         i++;
    }
    for(i=1;i<=a;i++)
       if(a%i==0&&b%i==0)
               t=i;
    */
    printf("最大公约数是:%d\n",t);
    printf("最小公倍数是:%d\n",m*n/t);
```

```
return 0;
```

最小公倍数与最大公约数的三种算法

最小公倍数:数论中的一种概念,两个整数公有的倍数成为他们的公倍数,其中一个最小的公倍数是他们的最小公倍数,同样地,若干个整数公有的倍数中最小的正整数称为它们的最小公倍数,维基百科:定义点击打开链接

求最小公倍数算法:

最小公倍数=两整数的乘积÷最大公约数

求最大公约数算法:

(1)辗转相除法

有两整数 a 和 b:

- ① a%b 得余数 c
- ② 若 c=0,则 b即为两数的最大公约数
- ③ 若 c≠0,则 a=b, b=c,再回去执行①

例如求 27 和 15 的最大公约数过程为:

```
27÷15 余 1215÷12 余 312÷3 余 0 因此,3 即为最大公约数 #include<stdio.h>
void main() /* 辗转相除法求最大公约数 */

{
    int m, n, a, b, t, c;

    printf("Input two integer numbers:\n");

    scanf("%d%d", &a, &b);
    m=a;    n=b;

while(b!=0) /* 余数不为 0,继续相除,直到余数为 0 */

{ c=a%b; a=b; b=c;}

13 / 133
```

```
printf("The largest common divisor:%d\n", a);
    printf("The least common multiple:%d\n", m*n/a);
  }
  (2) 相减法
  有两整数 a 和 b:
  ① 若 a>b,则 a=a-b
  ② 若 a<b, 则 b=b-a
  ③ 若 a=b,则 a (或 b)即为两数的最大公约数
  ④ 若 a≠b,则再回去执行①
  例如求 27 和 15 的最大公约数过程为:
  27-15=12(15>12)15-12=3(12>3)
  12-3=9(9>3)9-3=6(6>3)
  6-3=3(3==3)
  因此,3即为最大公约数
  #include<stdio.h>
  void main () /* 相减法求最大公约数 */
  {
  int m, n, a, b, c;
    printf("Input two integer numbers:\n");
    scanf ("%d,%d", &a, &b);
m=a; n=b;
       /* a, b 不相等,大数减小数,直到相等为止。*/
    while (a!=b)
```

```
if (a>b) a=a-b;
else b=b-a;
 printf("The largest common divisor:%d\n", a);
printf("The least common multiple:%d\n", m*n/a);
}
(3)穷举法
有两整数 a 和 b:
① i=1
② 若 a, b 能同时被 i 整除,则 t=i
③ j++
④ 若 i <= a(或 b),则再回去执行②
⑤ 若 i>a(或 b),则 t 即为最大公约数,结束
改进:
① i= a(或 b)
② 若 a, b 能同时被 i 整除,则 i 即为最大公约数,
结束
③ i--, 再回去执行②
有两整数 a 和 b:
① i=1
② 若 a, b 能同时被 i 整除,则 t=i
③ j++
④ 若 i <= a(或 b),则再回去执行②
⑤ 若 i > a(或 b),则 t 即为最大公约数,结束
15/133
```

```
改进:
   ① i= a(或 b)
   ② 若 a, b 能同时被 i 整除,则 i 即为最大公约数,
   结束
   ③ i--, 再回去执行②
   #include<stdio.h>
   void main () /* 穷举法求最大公约数 */
   {
   int m, n, a, b, i, t;
      printf("Input two integer numbers:\n");
      scanf ("%d,%d", &a, &b);
m=a; n=b;
      for (i=1; i<= a; i++)
           if ( a\%i == 0 \&\& b\%i == 0 )
                      t=i;
      printf("The largest common divisor:%d\n", t);
       printf("The least common multiple:%d\n", m*n/t);
   }
   /* 改进后的
      for (t= a; t>0; t--)
           if ( a%t == 0 && b%t ==0 )
                                      break;
```

/ } /

输入:

在歌星大奖赛中,有 10 个评委为参赛的选手打分,分数为 1~100 分。选手最后得分为:去掉一个最高分和一个最低分后其余 8 个分数的平均值。请编写一个程序实现。题目条件不变,但考虑同时对评委评分进行裁判,即在 10 个评委中找出最公平和最不公平···

在歌星大奖赛中,有 10 个评委为参赛的选手打分,分数为 1~100 分。选手最后得分为:去掉一个最高分和一个最低分后其余 8 个分数的平均值。请编写一个程序实现。

```
Input number1=90
   Input number2=91
   Input number3=93
   Input number4=94
   Input number5=90
   Input number6=99
   Input number7=97
   Input number8=92
   Input number9=91
   Input number10=95
   输出:
   Canceled max score:99
   Canceled min score:90
   Averagescore:92
   */
#include<stdio.h>
int main(void)
{
    int integer,i,max,min,sum;
    max=-32768;
    min=32767;
    sum=0;
    for(i=1;i<=10;i++)
    printf("Input number %d=",i);
    scanf("%d",&integer);
    sum+=integer;
    if(integer>max)max=integer;
    if(integer<min)min=integer;
    }
    printf("Canceled max score:%d\nCanceled min score:%d\n",max,min);
    printf("Averagescore:%d\n",(sum-max-min)/8);
}
```

```
/*
输入:
     Enter n:10
输出:
      55
*/
#include "stdio.h"
int feibo(int n)
{
     if(n<=2)
         return 1;
     else
      return feibo(n-1)+feibo(n-2);
}
int main(void)
{
     int n;
     printf("Enter n:");
     scanf("%d",&n);
     printf("%d\t",feibo(n));
     return 0;
}
```

十二,输出斐波那契数列每一项的数值

```
输入:
     Enter n:10
 输出:
          2
                   3
                               5
                                      8
1
       1
                                              13
                                                      21
                                                              34
                                                                     55
*/
 #include "stdio.h"
 int feibo(int n)
 {
     if(n==2||n==1)
        return 1;
```

十三, Fibonacci 数列其它问题

/*

问题描述

Fibonacci 数列的递推公式为: Fn=Fn-1+Fn-2, 其中 F1=F2=1。

当 n 比较大时, Fn 也非常大, 现在我们想知道, Fn 除以 10007 的余数是多少。

输入格式

输入包含一个整数 n。

输出格式

输出一行,包含一个整数,表示 Fn 除以 10007 的余数。

说明:在本题中,答案是要求 Fn 除以 10007 的余数,因此我们只要能算出这个余数即可,而不需要先计算出 Fn 的准确值,再将计算的结果除以 10007 取余数,直接计算余数往往比先算出原数再取余简单。

```
样例输入
10
样例输出
55
样例输入
22
样例输出
7704
*/
#include<stdio.h>
int a[1000001];
int main()
{
```

```
int i,n;
     a[1]=1;
     a[2]=1;
    for(i=3;i<=1000000;i++)
           a[i]=(a[i-1]+a[i-2])%10007;
     }
    scanf("%d",&n);
     printf("%d\n",a[n]);
     return 0;
}
```

十四, 求前 n 项的和 1+2+...+n

```
问题描述
求 1+2+3+...+n 的值。
输入格式
输入包括一个整数 n。
输出格式
输出一行,包括一个整数,表示 1+2+3+...+n 的值。
样例输入
样例输出
10
样例输入
100
样例输出
5050
数据规模与约定
1 <= n <= 1,000,000,000<sub>o</sub>
说明:有一些试题会给出多组样例输入输出以帮助你更好的做题。
一般在提交之前所有这些样例都需要测试通过才行,但这不代表这几组样例数据都正确
了你的程序就是完全正确的,
潜在的错误可能仍然导致你的得分较低。
#include <stdio.h>
int main()
{
   long long n, sum;
   scanf("%I64d", &n);
   sum = (n + 1) * n / 2;//利用等差数列公式
   printf("%I64d", sum);
   return 0;
   20/133
```

十五,序列求和

```
问题描述
求 1+2+3+...+n 的值。
输入格式
输入包括一个整数 n。
输出格式
输出一行,包括一个整数,表示 1+2+3+...+n 的值。
样例输入
4
样例输出
10
样例输入
100
说明:有一些试题会给出多组样例输入输出以帮助你更好的做题。
一般在提交之前所有这些样例都需要测试通过才行,但这不代表这几组样例数据都正确
你的程序就是完全正确的,潜在的错误可能仍然导致你的得分较低。
样例输出
5050
数据规模与约定
1 <= n <= 1,000,000,000<sub>o</sub>
说明:请注意这里的数据规模。
本题直接的想法是直接使用一个循环来累加,然而,当数据规模很大时,这种"暴力"
的方法往往会导致超时。
此时你需要想想其他方法。你可以试一试,如果使用 100000000 作为你的程序的输入,
你的程序是不是能在规定的上面规定的时限内运行出来。
本题另一个要值得注意的地方是答案的大小不在你的语言默认的整型(int)范围内,如果
使用整型来保存结果,
会导致结果错误。
如果你使用 C++或 C 语言而且准备使用 printf 输出结果,则你的格式字符串应该写成
%I64d 以输出 long long 类型的整数。
*/
#include <stdio.h>
int main(void)
{
  long long sum;
  long n;
  scanf("%d",&n);
  sum=(1+n)*n/2; //利用等差数列求和公式
  printf("%I64d",sum);
```

```
return 0;
```

十六,图像显示

```
/*
问题描述
    编写一个程序,首先输入一个整数,例如5,然后在屏幕上显示如下的图形(5表
示行数):
*/
#include<stdio.h>
int main(void)
{
   int n;
   int i,j;
   printf("Enter n:");
   scanf("%d",&n);
   for(i=0;i<5;i++)
       for(j=n-i;j>0;j--)
           printf("* ");
       printf("\n");
   }
   return 0;
}
```

十七,星期几

```
/*

1949年的国庆节(10月1日)是星期六。
今年(2012)的国庆节是星期一。
那么,从建国到现在,有几次国庆节正好是星期日呢?
只要答案,不限手段!
可以用 windows 日历,windows 计算器,Excel 公式,。。。。。。当然,也可以编程!
不要求写出具体是哪些年,只要一个数目!
```

```
输出: 9
*/
# include <stdio.h>
int yday(int year);
int main(void)
{
    int year;
              //是星期日的天数
    int n;
               //星期几
    int w;
    n = 0;
    w = 6;
for(year = 1950; year < 2013; year++) {
         w = (yday(year) \% 7 + w) \% 7;
         if(w == 0) {
                        n++;
                  }
              }
    printf("%d", n);
}
int yday(int year)
{
    if((year % 4 == 0) && (year % 100 != 0) || (year % 400 == 0))
    return 366;
    }
          else {
    return 365;
    }
}
```

十八,16进制转10进制

```
/*
输入:
Enter 16 进制的数: 14B
输出:
331
*/
#include "stdio.h"
#define N 50
long fun(char *s)
{
    int i,t;
23/133
```

```
long sum=0;
       for(i=0;s[i];i++)
           if(s[i]>='0'&&s[i]<='9')//当字符是'0'--'9'时, *- '0'就行了
               t=s[i]-'0';
           if(s[i] > = 'a' \& \& s[i] < = 'z')
               t=s[i]-'a'+10;//当字符是 abcdef 时,*-'a'+10 就行了
           if(s[i] > = 'A' \& \& s[i] < = 'Z')
               t=s[i]-'A'+10;//当字符是 ABCDEF 时, *- 'A' +10 就行了
           sum=sum*16+t;
       }
       return sum;
   }
                //本题先定义一个数组来保存输入的数,数采用以字符串的方式输
   int main(void)
入,
                    //利用字符串与字符数组相结合的方式
   {
       int m;
       char s[N];
       printf("Enter 16 进制的数:");
       scanf("%s",s);
       m=fun(s);
       printf("%ld",m);
   }
```

十九,10进制转16进制

```
/*
输入:
Enter n:666
输出:
29A
*/
#include"stdio.h"
#include"stdlib.h"
int main(void)
```

```
int n,a1,count=0,j;//count 用于角标的计数,j 控制 for 循环
int a[100];
printf("Enter n:");
scanf("%d",&n);
if(n==0)
printf("%d",n);

24 / 133
```

```
while(n!=0)
   {
       a1=n;
       n=n/16;
       a[count]=a1%16;
       count++;
   }
   for(j=count-1;j>=0;j--)
   {
       if(a[j]>9&&a[j]<16)
            printf("%c",(a[j]-10+'A'));
       else
            printf("%d",a[j]);
   }
   printf("\n");
   return 0;
}
//另一简单算法
#include <stdio.h>
int main(void)
{
     int n;
     scanf("%d", &n);
     printf("%X", n);
     return 0;
}
```

二十,16进制转8进制

```
/*
输入:
Enter 16 进制的数: 16A
输出:
552
*/
#include "stdio.h"
#define N 50
int fact(int n) //利用了 16 进制转为 10 进制,再将 10 进制转为 8 进制解决的
{
if(n<8)
```

```
return n;
        else
         return fact(n/8)*10+n%8;
   }
   long fun(char *s)
   {
        int i,t;
        long sum=0;
        for(i=0;s[i];i++)
        {
            If(s[i]>='0'&&s[i]<='9')//当字符是'0'--'9'时, *- '0'就行了
                t=s[i]-'0';
            if(s[i] > = 'a' \& \& s[i] < = 'z')
                t=s[i]-'a'+10;//当字符是 abcdef 时,*- 'a'+10 就行了
            if(s[i] > = 'A' \& \& s[i] < = 'Z')
                t=s[i]-'A'+10;//当字符是 ABCDEF 时, *- 'A' +10 就行了
            sum=sum*16+t;
        }
         return fact(sum);
   int main(void) //本题先定义一个数组来保存输入的数,数采用以字符串的方式输
λ,
                     //利用字符串与字符数组相结合的方式
   {
        int m;
        char s[N];
        printf("Enter 16 进制的数: ");
        scanf("%s",s);
        m=fun(s);
        printf("%ld",m);
        return 0;
   }
```

二十一,判断是否是回文

```
/*
输入:
请输入数时用空格隔开
输入你要判断是否是回文的<-6->个数: 159951
输出:
您输入的--是回文
------
输入:
请输入数时用空格隔开
输入你要判断是否是回文的<-6->个数: 152364
26/133
```

```
输出:
您输入--bu 是回文
*/
#include"stdio.h"
#define N 6 //通过更改 N 的值,可以改变输入的数的个数
//本题思想用数组的第一位与最后一位相比,然后依次第二与倒数第二。。。。
//但每输入一个数要用空格隔开,只需要改 N 的值就行了
int main(void)
{
    int a[N],flag=0,i;
    printf("请输入数时用空格隔开\n");
    printf("输入你要判断是否是回文的<-%d->个数: ",N);
    for(i=0;i<N;i++)
   {
       scanf("%d",&a[i]);
    for(i=0;i<N;i++)
    {
       if(a[i]==a[N-i-1])
       {
          flag=1;
          continue;
       }
       else
          break;
   }
    if(flag==1)
       printf("您输入的--是回文\n");
    else
       printf("您输入--bu 是回文\n");
    return 0;
}
```

二十二,闰年的判断

```
/*
输入:
Enter you input year:2014
输出:
您输入的 2014-->bu 是闰年
------
输入:
Enter you input year:2008
输出:
27/133
```

```
您输入的 2008-->是闰年
  */
  #include"stdio.h"
  #include"stdlib.h"
  //主要是判断的算法,因为每一年的精确天是 365.2422, 0.2422*4=0.9688
  //故 4 年算一个闰年,但还有 0.0312 的误差,每 100 个 4 年又少了 3.12 天,因此整
百年要被 400 整除
  int main(void)
  {
       int year;
       printf("Enter you input year:");
       scanf("%d",&year);
       if(year<=0)
       {
          printf("亲,你输入错误\n");
          exit(0);
       }
       else if((year%4==0&&year%100!=0)||year%400==0)
          printf("您输入的%d-->是闰年\n",year);
       else
          printf("您输入的%d-->bu 是闰年\n",year);
       return 0;
```

二十三,输出C字母图形

```
#include "stdio.h"
#include "math.h"
int main(void)
{
    double y;
    int x,m;
    for(y=10;y>=-10;y--)
    {
         m=2.5*sqrt(100-y*y);
         for(x=1;x<30-m;x++)
         printf("");
         printf("*\n");
    }
    return 0;
}</pre>
```

```
1
                                               2
                                                  3
                                 1
                                     5
                                          10
                                                 10
                                 6
                                       15
                                              20
                                                     15
                          1
                                                             6
                                   21
                                          35
                                                 35
                                                        21
                                                                7
                   1
                                28
                                       56
                                              70
                                                     56
                                                            28
                                                                           1
                1
                            36
                                   84
                                         126
                                                126
                                                        84
                                                               36
*/
#include <stdio.h>
#define N 12 //会输出 13 行
                              更改会输出相应的 N+1 行
long combi(int n, int r)
     int i;
     long p = 1;
     for(i = 1; i <= r; i++)
         p = p * (n-i+1) / i;//用于计算每一行的数
     return p;
}
void paint() {
int n, r, t;
for(n = 0; n \le N; n++) {
     for(r = 0; r <= n; r++) {
         int i;
         /* 排版设定开始 */
         if(r == 0) {
             for(i = 0; i \le (N-n); i++)
                  printf(" ");//3 个空格的距离
         }else {
             printf("
                        ");
         }//if-else
         /* 排版设定结束 */
         printf("%3d", combi(n,r));
     }//for2
     printf("\n");
}//for1
29/133
```

```
}
int main(void)
{
  paint();
  return 0;
}
```

二十五,三色旗

/*说明

三色旗的问题最早由 E.W.Dijkstra 所提出,他所使用的用语为 Dutch Nation Flag(Dijkstra 为荷兰

人), 而多数的作者则使用 Three-Color Flag 来称之。

假设有一条绳子,上面有红、白、蓝三种颜色的旗子,起初绳子上的旗子颜色并没 有顺序,您

希望将之分类,并排列为蓝、白、红的顺序,要如何移动次数才会最少,注意您只 能在绳子上

进行这个动作, 而且一次只能调换两个旗子。

解法

在一条绳子上移动,在程式中也就意味只能使用一个阵列,而不使用其它的阵列来 作辅助,问

题的解法很简单,您可以自己想像一下在移动旗子,从绳子开头进行,遇到蓝色往 前移,遇到

白色留在中间,遇到红色往后移,如下所示:

只是要让移动次数最少的话,就要有些技巧:

如果图中 W 所在的位置为白色,则 W+1,表示未处理的部份移至至白色群组。

如果 W 部份为蓝色,则 B 与 W 的元素对调,而 B 与 W 必须各+1,表示两个群组都 多了一个元素 。

如果 W 所在的位置是红色,则将 W 与 R 交换,但 R 要减 1,表示未处理的部份减 1。 注意 B、W、R 并不是三色旗的个数,它们只是一个移动的指标;什么时候移动结束 呢?一开始

时未处理的 R 指标会是等于旗子的总数,当 R 的索引数减至少于 W 的索引数时,表示接下来的旗

子就都是红色了,此时就可以结束移动,如下所示:

*/

//本程序在 D-evc 下运行正常,在 vc++6.0 需要处理一下

//对数组不同的初始化,结果也会不同

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define BLUE 'b' //宏定义字符 b 为蓝色

```
#define WHITE 'w' //宏定义字符 w 为白色
   #define RED 'r'
                   //宏定义字符 r 为红色
   void swap(char color[],int x,int y);
   int main(void) {
   char color[] = {'r', 'w', 'b', 'w', 'w',
       'b', 'r', 'b', 'w', 'r', '\0'};//初始化 color 数组
   int wFlag = 0;//用于记录白色旗的颜色
   int bFlag = 0;//用于记录蓝色旗的颜色
   int rFlag = strlen(color) - 1;//用于记录红色旗的颜色 =9
   printf("color 数组的长度: %d\n",strlen(color));//color 数组的长度
   int i;//控制循环的变量 i
   for(i = 0; i<strlen(color); i++)</pre>
       printf("%c ", color[i]);//将数组先输出一遍
   printf("\n");
   //以下程序段完成字符的交换
                         //循环条件: Wflag<Rflag
   while(wFlag <= rFlag) {
       if(color[wFlag] == WHITE)
           wFlag++;
       else if(color[wFlag] == BLUE) {
           swap(color,bFlag,wFlag);
           bFlag++; wFlag++;//这一点有一点不好理解,因为 Wflag 从 0 开始遍历,当
Wflag 遍历的结果是蓝色时,Bflag 加 1,
                             //Wflag 要到下一个位置,所以也要加1
       }
       else {
           //注意这里是循环,并且注意循环条件
           while(wFlag < rFlag && color[rFlag] == RED)//在 Wflag<Rflag 时,循环执行
               rFlag--;
           swap(color,rFlag,wFlag);//每执行一次交换,将红色旗交换到后面,所以最
后几个肯定是红色旗
           rFlag--;
       }
   }
   for(i = 0; i < strlen(color); i++)//将结果输出
       printf("%c ", color[i]);
   printf("\n");
   return 0;
   }
   void swap(char color[],int x,int y)//交换函数
    {
       char temp;
       temp = color[x];
       color[x] = color[y];
       color[y] = temp;
    31/133
```

二十六,回文数

```
/*
  问题描述
      1221 是一个非常特殊的数,它从左边读和从右边读是一样的,编程求所有这样
的四位十进制数。
  输出格式
      按从小到大的顺序输出满足条件的四位十进制数。
  */
  #include"stdio.h"
  int main(void)
  {
   int i,a,b,c,d;
   for(i=1000;i<10000;i++)
   {
      a=i/1000;
      b=(i%1000)/100;
      c=(i%100)/10;
      d=i%10;
      if(a==d\&\&b==c)
          printf("%d%d%d%d\n",a,b,c,d);
   }
   return 0;
  }
```

二十七,特殊回文数(普通算法)

```
/*
123321 是一个非常特殊的数,它从左边读和从右边读是一样的。
输入一个正整数 n, 编程求所有这样的五位和六位十进制数,满足各位数字之和等于 n 。
输入格式
输入一行,包含一个正整数 n。
输出格式
按从小到大的顺序输出满足条件的整数,每个整数占一行。
样例输入
52
```

```
样例输出
899998
989989
998899
数据规模和约定
1<=n<=54。
*/
#include"stdio.h"
int main(void)
{
int n;
int a[6];
printf("Enter n:");
scanf("%d",&n);
if(n>=1&&n<=54)
     for(a[0]=0;a[0]<=9;a[0]++)
     {
         for(a[1]=0;a[1]<=9;a[1]++)
         {
              for(a[2]=0;a[2]<=9;a[2]++)
              {
                   for(a[3]=0;a[3]<=9;a[3]++)
                       for(a[4]=0;a[4]<=9;a[4]++)
                           if(a[0]==a[4]\&\&a[1]==a[3]\&\&a[0]+a[1]+a[2]+a[3]+a[4]==n)
                                    printf("%d%d%d%d%d\n",a[0],a[1],a[2],a[3],a[4]);
                       }
                   }
              }
         }
     }
     for(a[0]=0;a[0]<=9;a[0]++)
         for(a[1]=0;a[1]<=9;a[1]++)
              for(a[2]=0;a[2]<=9;a[2]++)
                   for(a[3]=0;a[3]<=9;a[3]++)
                   {
                       for(a[4]=0;a[4]<=9;a[4]++)
                           for(a[5]=0;a[5]<=9;a[5]++)
```

```
{
```

二十八,特殊回文数(经典算法)

```
#include <stdio.h>
int main()
{
     int i, n;
     scanf("%d", &n);
     for (i = 10000; i < 100000; i++)
          int a = i / 10000;
          int b = (i \% 10000) / 1000;
          int c = (i \% 1000) / 100;
          int d = (i % 100) / 10;
          int e = i \% 10;
          if (a == e && b == d && a+b+c+d+e == n)
               printf("%d\n", i);
     }
     for (i = 100000; i < 1000000; i++)
          int a = i / 100000;
          int b = (i % 100000) / 10000;
          int c = (i \% 10000) / 1000;
          int d = (i % 1000) / 100;
          int e = (i % 100) / 10;
          int f = i \% 10;
```

二十九,特殊的数字

```
/*问题描述
      153 是一个非常特殊的数,它等于它的每位数字的立方和,即
153=1*1*1+5*5*5+3*3*3。编程求所有满足这种条件的三位十进制数。
  #include"stdio.h"
  int main(void)
   int n;
   int a,b,c;//a 表示数字左起第 1 位, b 表示左起第 2 位, c 表示第 3 位
   for(n=100;n<=999;n++)
   {
      a=n/100;
      b=n/10%10;
      c=n%10;
      if(a*a*a+b*b*b+c*c*c==n)
          printf("%d%d%d\n",a,b,c);
   }
   return 0;
  }
```

三十,查找整数

```
/*
如果 a 在数列中出现了,输出它第一次出现的位置(位置从 1 开始编号),否则输出-1。
样例输入
6
194839
9
样例输出
2
35/133
```

```
数据规模与约定
      1 <= n <= 1000 °
*/
#include"stdio.h"
#include"stdlib.h"
int main(void)
{
int a[1000];
int n,i,x,flag=0;//利用 flag 标记功能,在后面判断如何输出时起作用
printf("Enter n:");
scanf("%d",&n);
printf("input %d number:",n);
for(i=0;i<n;i++)
{
    scanf("%d",&a[i]);
}
printf("Enter x:");
scanf("%d",&x);
for(i=0;i<n;i++)
{
     if(a[i]==x){
         flag=1;
         break;//如不退出,会一直找到最后
    }
}
if(flag==1)
     printf("%d",i+1);
else
     printf("%d",-1);
return 0;
}
```

三十一,操作格子

```
/*
问题描述
有 n 个格子,从左到右放成一排,编号为 1-n。
共有 m 次操作,有 3 种操作类型:
1.修改一个格子的权值,
36/133
```

```
2.求连续一段格子权值和,
```

3.求连续一段格子的最大值。

对于每个2、3操作输出你所求出的结果。

输入格式

第一行 2 个整数 n, m。

接下来一行n个整数表示n个格子的初始权值。

接下来 m 行,每行 3 个整数 p,x,y,p 表示操作类型,p=1 时表示修改格子 x 的权值为 y,p=2 时表示求区间[x,y]内格子权值和,p=3 时表示求区间[x,y]内格子最大的权值。

输出格式

有若干行,行数等于 p=2 或 3 的操作总数。

每行1个整数,对应了每个p=2或3操作的结果。

```
样例输入
43
1234
213
143
314
样例输出
6
3
*/
#include"stdio.h"
void fuzhi(int a[],int n,int x,int y)//用于赋值的函数
{
int i;
for(i=0;i<n;i++)
{
     if(a[i]==x){
         a[i]=y;
         break;
     }
}
void sum(int a[],int n,int x,int y)//用于求和的函数
int c,b,i;
int sum=0;
for(i=0;i<=n;i++)
{
     if(a[i]==x)
         c=i;
     if(a[i]==y)
```

```
b=i;
    }
    for(i=c;i<=b;i++)
         sum=sum+a[i];
    //printf("%d%d\n",c,b);
    printf("%d\n",sum);
   void bijiao(int a[],int n,int x,int y)//用于比较的函数
    int max=-1,i;
    for(i=0;i<n;i++)
    if(max<a[i])
              max=a[i];
    printf("%d\n",max);
}
int main(void)
    int n,m,i;
    int a[100];//保存初始的权值
    int p[100],x[100],y[100];//方便一次输入 m 行
    printf("Enter n and m:");
    scanf("%d%d",&n,&m);
    printf("Enter %d number:\n",n);
    for(i=0;i<n;i++)
    {
         scanf("%d",&a[i]);
    }
    printf("Enter %d hang number:\n",m);
    for(i=0;i<m;i++)
    {
         scanf("%d%d%d",&p[i],&x[i],&y[i]);
    }
    for(i=0;i<m;i++)
    {
         if(p[i]==1)
             fuzhi(a,n,x[i],y[i]);
         if(p[i]==2)
              sum(a,n,x[i],y[i]);
         if(p[i]==3)
              bijiao(a,n,x[i],y[i]);
```

```
}
```

三十二,n阶高精度算法<1--10000>的阶乘

/*

输入:

Enter n:500

输出:

```
*/
#include "stdio.h"
#include "String.h"
#define MAX 10000
int f[MAX];
void Arr_reset(int a[],int m,int n)
{
    int i;
    for(i=m;i<=m;i++)
    {
        a[i]=0;
    }
}
int main(void)</pre>
```

```
{
    int i,j,n;
    printf("Enter n:");
    scanf("%d",&n);
    Arr_reset(f,0,(sizeof(f)/sizeof(int)));//对数组进行初始化
    f[0]=1;
    for(i=2;i<=n;i++)
        //乘以 i
        int c=0;
        for(j=0;j<MAX;j++)//最不易理解的
            int s=f[j]*i+c;
            f[j]=s%10;
            c=s/10;
            //算出的 s 是单位数时, 会连续覆盖 f[0]
            //否则一个多位数会倒过来存储,如 123,f[0]存 3,f[1]存 2,f[3]存 1
            //因此上式先求余, 在求模
        }
    }
    for(j=MAX-1;j>=0;j--)
        if(f[j])
            break;//忽略前导 0
    for(i=j;i>=0;i--)
        printf("%d",f[i]);
    printf("\n");
    return 0;
}
```

三十三,老鼠走迷官

/*说明 老鼠走迷宫是递回求解的基本题型,我们在二维阵列中使用 2 表示迷宫墙壁,使用 1 来表

示老鼠的行走路径,试以程式求出由入口至出口的路径。

解法 老鼠的走法有上、左、下、右四个方向,在每前进一格之后就选一个方向前进,无法前

进时退回选择下一个可前进方向,如此在阵列中依序测试四个方向,直到走到出口 为止,这是

递回的基本题。

*/

//在数组的初始化中可以画你的地图

#include <stdio.h>
#include <stdlib.h>

```
int visit(int, int);
int maze[7][7] = {{2, 2, 2, 2, 2, 2, 2},
               \{2, 0, 0, 0, 0, 0, 2\},\
               \{2, 0, 2, 0, 2, 0, 2\},\
               \{2, 0, 0, 2, 0, 2, 2\},\
               \{2, 2, 0, 2, 0, 2, 2\},\
               \{2, 0, 0, 0, 0, 0, 2\},\
               {2, 2, 2, 2, 2, 2, 2}
               };
int startI = 1, startJ = 1; // 指定入口
int endI = 5, endJ = 5; // 指定出口
int success = 0;//用来标记 visit 函数的结果
int main(void) {
int i, j;
printf("显示迷宫: \n");
for(i = 0; i < 7; i++) {
     for(j = 0; j < 7; j++)
          if(maze[i][j] == 2)
               printf(""");
          else
               printf("
                         ");
          printf("\n");
}
if(visit(startI, startJ) == 0)
     printf("\n 没有找到出口! \n");
else {
     printf("\n 显示路径: \n");
     for(i = 0; i < 7; i++) {
          for(j = 0; j < 7; j++) {
               if(maze[i][j] == 2)
                    printf(""");
               else if(maze[i][j] == 1)
                    printf("\diamondsuit");
               else
                    printf(" ");
          printf("\n");
     }
}
return 0;
}
int visit(int i, int j) {//递归算出老鼠该走的道路,上下方向判断
maze[i][j] = 1;
if(i == endl && j == endJ)//说明到达了指定的出发点
```

三十四, 逆序对

/* Alice 是一个让人非常愉跃的人!他总是去学习一些他不懂的问题,然后再想出许多稀奇古怪的题目。这几天, Alice 又沉浸在逆序对的快乐当中,他已近学会了如何求逆序对对数,动态维护逆序对对数等等题目,他认为把这些题让你做简直是太没追求了,于是,经过一天的思考和完善, Alice 终于拿出了一道他认为差不多的题目:

有一颗 2n-1 个节点的二叉树,它有恰好 n 个叶子节点,每个节点上写了一个整数。如果将这棵树的所有叶子节点上的数从左到右写下来,便得到一个序列 a[1]… a[n]。现在想让这个序列中的逆序对数量最少,但唯一的操作就是选树上一个非叶子节点,将它的左右两颗子树交换。他可以做任意多次这个操作。求在最优方案下,该序列的逆序对数最少有多少。

Alice 自己已近想出了题目的正解,他打算拿来和你分享,他要求你在最短的时间内完成。

输入格式

第一行一个整数 n。

下面每行,一个数 x。

如果 x=0,表示这个节点非叶子节点,递归地向下读入其左孩子和右孩子的信息,如果 $x\neq 0$,表示这个节点是叶子节点,权值为 x。

输出格式

输出一个整数,表示最少有多少逆序对。

```
样例输入
3
0
0
3
1
2
样例输出
1
*/
```

```
//逆序对
#include "stdio.h"
#include "stdlib.h"
#define MAX 20000
typedef struct node
{
int data;
struct node *lchild,*rchild;
}Bttree;
int count=0;//用于叶子节点计数
int temp=0;//记录数组的元素个数
int CountNi(int a[],int n)
{
int i,j;
int nicount=0;//逆序对计数
printf("数组的输出是:");
for(i=0;i<n;i++)
    printf("%d",a[i]);
printf("\n");
for(i=0;i<n;i++)
    for(j=0;j<i;j++)
    {
        if(a[j]>a[i])
            nicount++;
    return nicount;
}
void ChangeArr(Bttree *bt,int a[])//将叶子节点按从左到右保存在数组中
{
if(bt->data==0)
{
    ChangeArr(bt->lchild,a);
    ChangeArr(bt->rchild,a);
}
else
{
    a[temp]=bt->data;//将 0.....n-1 的数据保存到 a 数组中
    temp++;
}
//printf("temp=%d",temp);
void Swap(Bttree *&bt,int a[])//交换二叉树的左右子树(根据条件)
Bttree *p;//定义一个临时变量用于交换
43/133
```

```
int Old, New;
if(bt->data!=0)
     return;
else
{
     if(bt->lchild->data>bt->rchild->data)
     {
          p=bt->lchild;
          bt->lchild=bt->rchild;
          bt->rchild=p;
     }
         temp=0;
         ChangeArr(bt,a);
         int b=temp;
          Old=CountNi(a,b);
          temp=0;
          int c=temp;
          New=CountNi(a,c);
          if(Old>New)
         {
              p=bt->lchild;
              bt->lchild=bt->rchild;
              bt->rchild=p;
         }
}
Swap(bt->lchild,a);
Swap(bt->rchild,a);
void Creat(Bttree *&bt,int n)//创建一颗二叉树
{
int data;
//printf("Enter data:");
scanf("%d",&data);
if(data==0)
{
     bt=(Bttree *)malloc(sizeof(Bttree));
     if(bt==NULL)
     {
          printf("bt 创建失败!\n");
          exit(0);
     }
     bt->data=data;
     Creat(bt->lchild,n);
```

```
Creat(bt->rchild,n);
}
else
{
    bt=(Bttree *)malloc(sizeof(Bttree));
    if(bt==NULL)
    {
        printf("bt 创建失败!\n");
        exit(0);
    }
    bt->data=data;
    bt->lchild=NULL;
    bt->rchild=NULL;
    count++;
    if(count==n)
        return;
}
//printf("count=%d\n",count);
int main(void)
Bttree * bt;
int a[MAX]; //用于保存叶子节点
int n,result;
printf("Enter n:");
scanf("%d",&n);
//printf("%d\n",Creat(bt,n));
Creat(bt,n);//创建一颗二叉树
//printf("yes");
Swap(bt,a);
                    //交换二叉树的左右孩子(根据条件)
//printf("yes\n");
ChangeArr(bt,a); //将叶子节点按从左到右保存在数组中
//printf("yes111\n");
                          //将逆序对的个数输出
result=CountNi(a,n);
printf("逆序对=%d\n",result);
    return 0;
}
```

```
/*
问题描述
   给定一个长度为 n 的数列,将这个数列按从小到大的顺序排列。1<=n<=200
输入格式
   第一行为一个整数 n。
   第二行包含 n 个整数,为待排序的数,每个整数的绝对值小于 10000。
输出格式
   输出一行,按从小到大的顺序输出排序后的数列。
样例输入
5
83649
样例输出
34689
*/
//快速排序算法
#include "stdio.h"
#include "stdlib.h"
void Print(int data[],int n)//输出函数
{
int i;
for(i=0;i<n;i++)
{
    printf("%2d",data[i]);
}
int Splid(int *data,int pre,int rear)//取 mid
int temp=data[pre];
if(pre<rear)
{
    while(data[rear]>=temp&&pre<rear)
           rear--;
        data[pre]=data[rear];
    while(data[pre]<temp&&pre<rear)
           pre++;
        data[rear]=data[pre];
}
data[pre]=temp;
return pre;
void Quicksort(int *data,int pre,int rear)//递归排序
{
int mid;
if(pre<rear)
```

```
{
     mid=Splid(data,pre,rear);
     Quicksort(data,pre,mid-1);
     Quicksort(data,mid+1,rear);
}
int main(void)
{
int n,i;
int *data;
data=(int *)malloc(sizeof(int)*n);//动态分配
printf("Enter n:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
     scanf("%d",&data[i]);
}
Quicksort(data,0,n-1);//排序
Print(data,n);//输出
return 0;
}
```

三十六,数列排序另一经典算法

```
#include <stdio.h>
void swap(int* a, int *b)
{
    int c = *a;
    *a = *b;
    *b = c;
    printf("%d %d\n",*a,*b);
}
void sort(int* arr, int len)
{
    if (len <= 1) return;
    int x = arr[len-1];
    int i = 0, j = 0;
    while (j < len - 1)
    {
        if (arr[j] < x){
        47 / 133</pre>
```

```
swap(&arr[i], &arr[j]);
                i++;
           }
          j++;
     }
     swap(&arr[i], &arr[len-1]);
     sort(arr, i);
     sort(arr + i + 1, len - i - 1);
}
int main(void)
{
     int i, n, a[200];
     scanf("%d", &n);
     for (i = 0; i < n; i++)
     scanf("%d", &a[i]);
     sort(a, n);
     for (i = 0; i < n; i++)
       printf("%d ", a[i]);
     printf("\n");
     return 0;
}
```

三十七,第 39 台阶

```
/*
如果我每一步迈上1个或两个台阶,先迈左脚,然后左右交换,最后一步迈右脚,
也就是一共要走偶数步,那么,上完39级台阶,有多少种不同的算法
*/
/*思路:
不管走的是左脚还是右脚,只要走的步数是偶数即可,递归的思想
                 39,0
           38,1
                       37,1
             36,2
       37,2
                   36,2
                           35,2
                            ... ...
                请等待几秒
输出: Total=51167078
*/
//方法1
#include "stdio.h"
```

```
int total=0;//计数
int Sum(int num,int step)
if(num<0)
    return 0;
if(num==0&&step%2==0)//不管走的是左脚还是右脚,只要走的步数是偶数即可
    total++;
    return 0;
}
Sum(num-1,step+1);
Sum(num-2,step+1);
}
int main(void)
{
int n;
printf("Enter n:");
scanf("%d",&n);
                //从顶部算
Sum(n,0);
printf("Total=%d\n",total);
return 0;
}
//方法 2
#include "stdio.h"
int total=0;//计数
int Sum(int num,int step)
if(num>39)
    return 0;
if(num==39&&step%2==0)//不管走的是左脚还是右脚,只要走的步数是偶数即可
    total++;
    return 0;
}
Sum(num+1,step+1);
Sum(num+2,step+1);
}
int main(void)
Sum(0,0);
                //从底部算
49/133
```

```
printf("Total=%d\n",total);
return 0;
}
*/
```

三十八,第三十九台阶(非递归)

```
#include "stdio.h"
   long fact(int m,int n)
   int i;
   double a,b,c;//因为阶乘数 m, n 稍大, 阶乘结果变大
   a=b=c=1;
    printf("a=%.0f,b=%.0f,c=%.0f\n",a,b,c);
    for(i=m+n;i>=1;i--)//求 m+n 的阶乘
       a=a*i;
   for(i=m;i>=1;i--)//求 m 的阶乘
        b=b*i;
   for(i=n;i>=1;i--)//求 n 的阶乘
       c=c*i;
   return a/(b*c);
   int main(void)
   int i,j;
   long sum=0;
   for(i=1;i<=37;i++)
       for(j=1;j<=19;j++)
            if((i+j)%2==0&&(i+2*j)==39)//满足两只脚的走的总的步数为偶数且走的阶
梯数总和为39
           {
               sum+=fact(i,j);//求总的组合数
               printf("%d %d\n",i,j);//输出两只脚的种类数
           }
       }
        printf("%ld\n",sum);
        return 0;
   }
   */
```

三十九,最短路径 ——Dijkstar 算法

```
数据已在数组中初始化
      算出从给定的点到最最终点的一条最短距离
   输出:
       最短路径为16
   */
#include "stdio.h"
#include "stdlib.h"
#define MAXVEX 9
#define M 65535
                //用于存储最短路径下标的数组
int P[MAXVEX];
int D[MAXVEX]; //用于存储到各点最短路径下的权值和
typedef struct {
   int numVertexes;
   int arc[MAXVEX][MAXVEX];
   }MGraph;
void ShortestPath_Dijkstar(MGraph G,int v0)
{
   int v,w,k,min;
   int final[MAXVEX];//final[w]==1 表示已经求得顶点 VO 到 Vw 的最短路径
   //初始化数据
   for(v=0;v<G.numVertexes;v++)
   {
       final[v]=0;
                        //全部顶点初始化为未找到最短路径
       D[v]=G.arc[v0][v]; //将于 V0 点有连线的顶点加上权值
                    //初始化路径数组 P 为 0
       P[v]=0;
   }
             //v0 到 v0 的路径为 0
   D[v0]=0;
   final[v0]=1; //v0 到 v0 不需要求路径
   //开始主循环,每次求得 v0 到某个 v 顶点的最短路径
   for(v=1;v<G.numVertexes;v++)
   {
       min=M;
       for(w=0;w<G.numVertexes;w++)</pre>
          if(!final[w]&&D[w]<min)</pre>
```

```
{
                k=w;//k=8 就不再读了
               min=D[w];
           }
       }
       //printf("k=%d",k);
       final[k]=1; //将目前找到的最近的顶点置为1
        //修正当前最短路径及距离
       for(w=0;w<G.numVertexes;w++)
       {
            //如果经过 v 顶点的路径比现在这条路径的长度短的话, 更新!
           if(!final[w]&&(min+G.arc[k][w]<D[w]))
           {
                D[w]=min+G.arc[k][w]; //修正当前路径长度
                                    //存放前驱节点
               P[w]=k;
               //printf(" -%d",D[w]);
           }
       }
   }
}
int main(void)
    MGraph G;
    int i,j;
    G.numVertexes=MAXVEX;
    int a[MAXVEX][MAXVEX]={
      \{0,1,5,M,M,M,M,M,M,M\}
      {1,0,3,7,5,M,M,M,M},
      {5,0,3,M,1,7,M,M,M},
      {M,7,M,0,2,M,3,M,M},
      {M,5,1,2,0,3,6,9,M},
      {M,M,7,M,3,0,M,5,M},
      {M,M,M,3,6,M,0,2,7},
      {M,M,M,M,9,5,2,0,4},
      {M,M,M,M,M,7,4,0},
      };
   for(i=0;i<MAXVEX;i++){</pre>
        for(j=0;j<MAXVEX;j++)</pre>
               G.arc[i][j]=a[i][j];
    }
    /*
    52/133
```

```
for(i=0;i<MAXVEX;i++){//验证数组中的值
        for(j=0;j<MAXVEX;j++)
             printf("%d\t",a[i][j]);
        printf("\n");
    }
    */
    ShortestPath_Dijkstar(G,0);//迪杰斯特拉算最短路径
    printf("P 的过程为: "); //处理过程
    for(i=0;i<MAXVEX;i++)</pre>
    {
        if(i==MAXVEX-1)
             printf("%d\n",P[i]);
        else
             printf("%d->",P[i]);
    }
    */
    //printf("D 的过程为: ");
    for(i=0;i<MAXVEX;i++)
        if(i==MAXVEX-1)
             printf("最短路径为%d\n",D[i]);
        //else
            //printf("%d->",D[i]);
    }
    return 0;
}
```

四十,最短路径 ——弗洛伊德算法

```
//最短路径 ——Floid 算法
/*
  数据已在数组中初始化
  算出从每个点到最最终点的最短距离
输出:
0-->8 的最短路径为 16
1-->8 的最短路径为 15
2-->8 的最短路径为 12
3-->8 的最短路径为 9
4-->8 的最短路径为 9
6-->8 的最短路径为 9
6-->8 的最短路径为 6
7-->8 的最短路径为 4
53/133
```

```
8-->8 的最短路径为 0
   */
   #include "stdio.h"
   #include "stdlib.h"
   #define MAXVEX 9
   #define M 65535
   int P[MAXVEX][MAXVEX]; //用于存储最短路径下标的数组
   int D[MAXVEX][MAXVEX]; //用于存储到各点最短路径下的权值和
   typedef struct {
    int numVertexes;
    int matrix[MAXVEX][MAXVEX];
    }MGraph;
void ShortestPath_Floyd(MGraph G)
    int v,w,k;
    //初始化 D 和 P;
    for(v=0;v<G.numVertexes;v++)
    {
        for(w=0;w<G.numVertexes;w++)</pre>
        {
            D[v][w]=G.matrix[v][w];
            P[v][w]=w;
        }
    }
    //优美的弗洛伊德算法
    for(k=0;k<G.numVertexes;k++)
        for(v=0;v<G.numVertexes;v++)</pre>
            for(w=0;w<G.numVertexes;w++)</pre>
            {
                if(D[v][w]>(D[v][k]+D[k][w]))
                {
                    D[v][w]=D[v][k]+D[k][w];
                    P[v][w]=P[v][k];
                }
            }
}
int main(void)
{
    MGraph G;
    int i,j;
    G.numVertexes=MAXVEX;
```

```
int a[MAXVEX][MAXVEX]={
  {0,1,5,M,M,M,M,M,M},
  {1,0,3,7,5,M,M,M,M},
  {5,0,3,M,1,7,M,M,M},
  \{M,7,M,0,2,M,3,M,M\},
  {M,5,1,2,0,3,6,9,M},
  {M,M,7,M,3,0,M,5,M},
  {M,M,M,3,6,M,0,2,7},
  {M,M,M,M,9,5,2,0,4},
  {M,M,M,M,M,7,4,0},
  };
for(i=0;i<MAXVEX;i++){</pre>
                          //对二维数组进行赋值
     for(j=0;j<MAXVEX;j++)
             G.matrix[i][j]=a[i][j];
}
/*
for(i=0;i<MAXVEX;i++){</pre>
                                   //测试数据
    for(j=0;j<MAXVEX;j++)
         printf("%d\t",a[i][j]);
     printf("\n");
}
printf("-----\n");
for(i=0;i<MAXVEX;i++){</pre>
     for(j=0;j<MAXVEX;j++)
         printf("%d\t",G.matrix[i][j]);
     printf("\n");
}
*/
  ShortestPath_Floyd(G);//弗洛伊德算最短路径
for(i=0;i<G.numVertexes;i++)//输出数据
{
     for(j=0;j<G.numVertexes;j++)</pre>
          if(j==G.numVertexes-1)
             printf("%d-->%d 的最短路径为%d\n",i,j,D[i][j]);
}
return 0;
}
```

四十一, 区间 k 大数查询

```
给定一个序列,每次询问序列中第 I 个数到第 r 个数中第 K 大的数是哪个。
  输入格式
        第一行包含一个数 n,表示序列长度。
        第二行包含 n 个正整数,表示给定的序列。
        第三个包含一个正整数 m,表示询问个数。
        接下来 m 行,每行三个数 l,r,K,表示询问序列从左往右第 l 个数到第 r 个数
中,从大往小第 K 大的数是哪个。序列元素从 1 开始标号。
  输出格式
        总共输出 m 行,每行一个数,表示询问的答案。
  样例输入
  5
  12345
  2
  152
  232
  样例输出
  2
  */
  #include<stdio.h>
  #include<stdlib.h>
  int count=0;
                      //记录每行的值
  //返回 mid
  int Splid(int *temp,int pre,int rear)
{
                  //临时存储第一个数据
   int tem=temp[pre];
   if(pre<rear)
   {
      while(temp[rear]>=tem&&pre<rear)//后面的数据比 tem 小,rear--
         rear--;
      temp[pre]=temp[rear];
      while(temp[pre]<tem&&pre<rear)//前面的数据比 tem 大, pre++
          pre++;
      temp[rear]=temp[pre];
      temp[pre]=tem;
   }
   return pre;
  }
  //快速排序
  void QuickSort(int *temp,int pre,int rear,int k,int *a)
  {
   if(pre<rear)
   {
      int mid=Splid(temp,pre,rear);
```

```
if(mid==k)
            a[count++]=temp[mid];
        if(mid>k)
            QuickSort(temp,pre,mid-1,k,a);
        if(mid<k)
            QuickSort(temp,mid+1,rear,k,a);
   }
   }
   int main(void)
{
                    //n 控制第一次输入的数的个数, m 控制输入几行数
    int n,m,i;
    int *data;
                        //动态分配
    int *a;
                        //保存每行的值
    printf("Enter n:");
    scanf("%d",&n);
    data=(int *)malloc(sizeof(int)*n);
    a=(int *)malloc(sizeof(int)*n);
    for(i=0;i<n;i++) //输入n个数据
    {
        scanf("%d",&data[i]);
    }
    printf("Enter m:");
    scanf("%d",&m);
    printf("Enter pre,rear,k:\n");
    while(m) //输入 m 行数据
    {
        int pre,rear,k;//需要输入的 3 个数据
                      //一个临时的动态数组
        int *temp;
        temp=(int *)malloc(sizeof(int)*n);
        scanf("%d%d%d",&pre,&rear,&k);
        for(i=0;i<n;i++)
            temp[i]=data[i];
        QuickSort(temp,pre-1,rear-1,rear-k,a); //rear-k 就是要查找的数据角标次数
        m--;
               //控制次数
    }
    for(i=0;i<count;i++)</pre>
                            //输出结果
        printf("%d\n",a[i]);
   }
   }
```

```
/*
```

西洋棋中的皇后可以直线前进,吃掉遇到的所有棋子,如果棋盘上有八个皇后,则 这八

个皇后如何相安无事的放置在棋盘上

解法 关于棋盘的问题, 都可以用递回求解, 然而如何减少递回的次数? 在八个皇后的问题中,

不必要所有的格子都检查过,例如若某列检查过,该该列的其它格子就不用再检查 了,这个方

法称为分支修剪。

```
输出:
0 0 0 0 0 0
0 0 0 0 0 0
第91种
0000001
00100000
10000000
00000100
01000000
00001000
0000010
00010000
第92种
0000001
00010000
1000000
00100000
00000100
01000000
0000010
00001000
*/
#include<stdio.h>
int count=0; //用于计数
int notDanger(int row,int line,int (*chess)[8])
int i,k,flag1=0,flag2=0,flag3=0,flag4=0,flag5=0;
//判断列方向
for(i=0;i<8;i++)
    if(chess[i][line]!=0)
    {
       flag1=1;
       break;
    }
```

```
//判断左上方
for(i=row,k=line;i>=0&&k>=0;i--,k--)
     if(chess[i][k]!=0)
     {
         flag2=1;
         break;
     }
}
//判断右下方
for(i=row,k=line;i<8\&\&k<8;i++,k++)
     if(chess[i][k]!=0)
         flag3=1;
         break;
     }
}
//判断左上方
for(i=row,k=line;i<8\&\&k>=0;i++,k--)
     if(chess[i][k]!=0)
     {
         flag4=1;
         break;
     }
}
//判断右下方
for(i=row,k=line;i>=0\&\&k<8;i--,k++)
     if(chess[i][k]!=0)
     {
         flag5=1;
         break;
     }
}
if(flag1==1||flag2==1||flag3==1||flag4==1||flag5==1)
     return 0;
}
else
     return 1;
}
```

```
EightQueen(int row,int n,int (*chess)[8])
{
//参数 row: 表示起始行
//参数 n:表示列数
//参数(*chess)[8]:传递棋盘的一个指针
int chess2[8][8],j,i;//chess2 是一个临时的二维数组
for(i=0;i<8;i++)
    for(j=0;j<8;j++)
        chess2[i][j]=chess[i][j];
                         //如果行到达 8,输出
if(row==8)
{
    printf("第%d 种\n",count+1);
    for(i=0;i<8;i++)
            for(j=0;j<n;j++)
                 printf("%d ",chess2[i][j]);
            }
            printf("\n");
    }
    count++;
}
                    //否则插入到合适的位置
else
    {
        for(i=0;i<n;i++)
            if(notDanger(row,i,chess2)!=0) //判断是否危险
            {
                 for(j=0;j<n;j++)
                     chess2[row][j]=0; //行赋值为 0, 每行只有一个
                 chess2[row][i]=1;
                 EightQueen(row+1,n,chess2);
            }
        }
    }
}
int main(void)
int chess[8][8],i,j;
               //初始化
for(i=0;i<8;i++)
    for(j=0;j<8;j++)
        chess[i][j]=0;
```

```
EightQueen(0,8,chess);
return 0;
}
```

四十三,八皇后(回溯算法)

```
//类同三十一
```

```
#include <stdio.h>
#include <stdlib.h>
#define N 8
int column[N+1]; // 同栏是否有皇后, 1表示有
int rup[2*N+1]; // 右上至左下是否有皇后
int lup[2*N+1]; // 左上至右下是否有皇后
int queen[N+1] = \{0\};
int num=0; // 解答编号
void backtrack(int); // 递回求解
int main(void) {
int i;
for(i = 1; i <= N; i++)
     column[i] = 1;
for(i = 1; i <= 2*N; i++)
     rup[i] = lup[i] = 1;
backtrack(1);//从第一个开始
return 0;
void showAnswer() {
int x, y;
printf("\n 解答 %d\n",++num);
for(y = 1; y \le N; y++) {
     for(x = 1; x \le N; x++) {
         if(queen[y] == x) {
              printf(" 1");
         }
         else {
             printf(" 0");
         }
     printf("\n");
}
}
void backtrack(int i) {
int j;
if(i > N) {
     showAnswer();
61/133
```

```
| else {
| for(j = 1; j <= N; j++) {
| if(column[j] == 1 && | lup[i-j+N] == 1) {
| queen[i] = j; | // 设定为占用 |
| column[j] = rup[i+j] = lup[i-j+N] = 0; |
| backtrack(i+1); |
| column[j] = rup[i+j] = lup[i-j+N] = 1; |
| }
| }
| }
| }
```

//有些算法在你不明白时,最好在稿纸上走一遍,这样可以更好地理解算法

四十四,八皇后问题(回溯算法)(2)

```
//类同三十一
//八皇后问题(回溯算法)
#include<stdio.h>
#include<stdlib.h>
#define max 8
int queen[max],sum=0;/*max 为棋盘最大的坐标*/
void show()//输出所有皇后的坐标
{
 int i;
 for(i=0;i<max;i++)</pre>
  printf("%d ",queen[i]);
 printf("\n");
 sum++;
int check(int n)
{
 int i;
 for(i=0;i<n;i++)
    if(queen[i]==queen[n]||abs(queen[i]-queen[n])==(n-i))
       return 1;
 }
 return 0;
}
void put(int n)
62/133
```

```
{
  int i;
  for(i=0;i<max;i++)
    queen[n]=i;//将皇后摆到当前循环的位置
     if(!check(n))
       if(n==max-1)
       show();//如果全部摆好,则输出所有皇后的坐标
       else
       {
       put(n+1);//否则继续摆放后一个皇后
       }
     }
 }
}
int main()
{
    put(0)://从横坐标为 0 开始一次尝试
    printf("%d",sum);
    system("pause");
    return 0;
}
```

四十五, 2n 皇后问题

/*

问题描述

给定一个 n*n 的棋盘,棋盘中有一些位置不能放皇后。现在要向棋盘中放入 n 个黑皇后和 n 个白皇后,使任意的两个黑皇后都不在同一行、同一列或同一条对角线上,任意的两个白皇后都不在同一行、同一列或同一条对角线上。问总共有多少种放法? n 小于等于 8。

输入格式

输入的第一行为一个整数 n,表示棋盘的大小。

接下来 n 行,每行 n 个 0 或 1 的整数,如果一个整数为 1,表示对应的位置可以放皇后,如果一个整数为 0,表示对应的位置不可以放皇后。

输出格式

输出一个整数,表示总共有多少种放法。

样例输入

1

```
1111
1111
1111
1111
样例输出
样例输入
4
1011
1111
1111
1111
样例输出
0
*/
#include <stdio.h>
int go(int* a, int* b, int* c, int m, int n)
{
     if (m == n)
          return 1;
     int i, j, k, count = 0;
     for (i = 0; i < n; i++)
     {
          if (a[m * n + i] == 0)
               continue;
          for (j = 0; j < m; j++)
               if (b[j] == i \mid \mid b[j] - i == j - m \mid \mid b[j] + j == i + m)
                     break;
          if (j == m)
               b[m] = i;
               for (k = 0; k < n; k++)
               {
                     if (a[m * n + k] == 0 | | k == i)
                          continue;
                     for (j = 0; j < m; j++)
                          if (c[j] == k \mid \mid c[j] - k == j - m \mid \mid c[j] + j == k + m)
                                break;
                     if (j == m)
                          c[m] = k;
                          count += go(a, b, c, m + 1, n);
                     }
               }
```

```
四十六, 前缀表达式
/*
问题描述
  编写一个程序,以字符串方式输入一个前缀表达式,然后计算它的值。
输入格式为: "运算符 对象 1 对象 2", 其中, 运算符为"+"(加法)、"-"(减
法)、"*"(乘法)
或"/"(除法),运算对象为不超过10的整数,它们之间用一个空格隔开。
要求:对于加、减、乘、除这四种运算,分别设计相应的函数来实现。
  输入格式:输入只有一行,即一个前缀表达式字符串。
  输出格式:输出相应的计算结果(如果是除法,直接采用c语言的"/"运算符,
结果为整数)。
  输入输出样例
样例输入
+52
样例输出
7
*/
#include<stdio.h>
int main(void)
{
  char fh;
  int a,b;
  scanf("%c%d%d",&fh,&a,&b);
  if(fh=='+')
     printf("%d\n",a+b);
  else if(fh=='-')
     printf("%d\n",a-b);
  65 / 133
```

四十七,最大最小公倍数(公约数不一定相同)

```
问题描述
      已知一个正整数 N, 问从 1~N 中任选出三个数, 他们的最小公倍数最大可以
为多少。
输入格式
     输入一个正整数 N。
输出格式
     输出一个整数,表示你找到的最小公倍数。
样例输入
样例输出
504
*/
#include"stdio.h"
int Zhishu(int n)
                 //质数判断
{
    int i,temp;
    if(n==1)
       return 0;
    if(n==2)
       return 1;
    temp=n/2;
    for(i=2;i<temp;i++)
       if(n%i==0)
           return 0;
    return 1;
}
int GongBei(int i,int j,int k,int n)//计算最小公倍数
int v,temp1,temp2,temp3;
temp1=Zhishu(i);
temp2=Zhishu(j);
 66/133
```

```
temp3=Zhishu(k);
    if(!temp1&&!temp2&&temp3)//i,j 不是质数, k 是质数
        for(v=n;v>0;v--)
        {
             if(i\%v==0\&\&j\%v==0)
                 break;
        }
        return ((i*j/v)*k);
    }
    if(!temp1&&temp2&&!temp3)//i,k 不是质数, j 是质数
        for(v=n;v>0;v--)
        {
             if(i\%v==0\&\&k\%v==0)
                 break;
        return ((i*k/v)*j);
    }
   if(temp1&&!temp2&&!temp3)//j,k 不是质数,i 是质数
        for(v=n;v>0;v--)
        {
             if(j\%v==0\&\&k\%v==0)
                  break;
        return ((j*k/v)*i);
    }
   //i,j,k 是质数或!i,j,k 是质数或 i,!j,k 是质数或 i,j,!k 是质数
    if(temp1&&temp2&&temp3||!temp1&&temp2&&temp3||temp1&&!temp2&&temp3
||temp1&&temp2&&!temp3)
        return (i*j*k);
}
int Result(int n)
{
    int i,j,k;
    int max=-1;
                  //
    for(i=1;i<=n;i++)
        for(j=i+1;j<=n;j++)
            for(k=j+1;k<=n;k++)
            {
                 if(i!=j\&\&i!=k\&\&j!=k)
                 {
    67/133
```

```
//printf("%d %d %d\n",i,j,k);<mark>//测试结果</mark>
                        if(max< GongBei(i,j,k,n))</pre>
                             max=GongBei(i,j,k,n);
                        }
                   }
    return max;
}
int main(void)
{
     int n,result;
     printf("Enter n:");
    scanf("%d",&n);
     result=Result(n);//算最大的最小公倍数
     printf("Result=%d",result);
    return 0;
}
```

四十八,2的次幂问题

```
问题描述
  任何一个正整数都可以用 2 进制表示,例如: 137 的 2 进制表示为 10001001。
  将这种 2 进制表示写成 2 的次幂的和的形式,令次幂高的排在前面,可得到如下表
达式: 137=2^7+2^3+2^0
  现在约定幂次用括号来表示,即 a^b 表示为 a(b)
  此时,137可表示为:2(7)+2(3)+2(0)
  进一步: 7=2^2+2+20 (2^1 用 2 表示)
  3=2+2^0
  所以最后 137 可表示为: 2(2(2)+2+2(0))+2(2+2(0))+2(0)
  又如: 1315=2^10+2^8+2^5+2+1
  所以 1315 最后可表示为:
  2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)
输入格式
  正整数(1<=n<=20000)
输出格式
  符合约定的 n 的 0, 2 表示 (在表示中不能有空格)
样例输入
137
样例输出
2(2(2)+2+2(0))+2(2+2(0))+2(0)
```

```
样例输入
1315
样例输出
2(2(2+2(0))+2)+2(2(2+2(0)))+2(2(2)+2(0))+2+2(0)
提示
用递归实现会比较简单,可以一边递归一边输出
*/
#include<stdio.h>
#include<math.h>
void fun(int x)
{
     int y,z;
     y=(int)(log(x)/log(2)+0.1);
     z=x-pow(2,y);
     if(y==0)
      printf("2(0)");
      return;
     }
     else if(y==1)
      printf("2");
     }
     else
      printf("2(");
      fun(y);
      printf(")");
     }
     if(z)
      printf("+");
      fun(z);
     }
}
int main(void)
{
     int n;
     scanf("%d",&n);
     fun(n);
     return 0;
}
```

```
/*
将 n 个数字 1, 2, ···n 的所有排列按字典顺序枚举出猴
输入: n
输出:
123
132
231
213
312
321
*/
//关于全排列你可以理解为深搜加回溯
第一种
#include<stdio.h>
#define MAX 10
int used[MAX];//用来标记数字是否已经在前面使用过
int result[MAX];//存放结果
int N;
void Print()//输出结果
{
   int i;
   for(i=0;i<N;i++)
      printf("%d",result[i]);
   printf("\n");
}
void proc(int step)//step 用来表示摆放好了几个数
{
   int i;
   if(step==N) //如果已经摆放好了 N 个数,那么结果就产生了,就输出结果
      Print();
   else
   {
      for(i=0;i<N;i++)//枚举 1——N, 找到没有使用过的最小的数
      {
         if(!used[i])//如果没使用过
         {
            used[i]=1;//标记 i 已经使用过
            result[step]=i+1;//记录结果
            proc(step+1);
            used[i]=0;/*这里就是所谓的回溯算法,也许比较难以理解,你可以人
工走一遍加深理解。其实回溯的主要想法就是"还原现场"。当执行到这一步时,i+1
```

```
这个数放在 step 个位置的情况已经解决了,我们就要拿出 i+1 这个数,把它标记为未使
用 */
             }
        }
    }
}
 int main(void)
    {
        printf("Enter N:");
        scanf("%d",&N);
        proc(0);
        return 0;
    }
第二种全排列算法, 也用到了回溯算法
#include<stdio.h>
#include<String.h>
void f(char data[],int k)
{
    if(k==sizeof(data)/sizeof(data[0]))
                                     //sizeof(data)/sizeof(data[0])用于计算数组的长
度
    {
             for(int i=0;i<sizeof(data)/sizeof(data[0]);i++)</pre>
             {
                 printf("%c ",data[i]);
             printf("\n");
    }
    for(int i=k;i<sizeof(data)/sizeof(data[0]);i++)</pre>
    {
        {
             char t=data[k];data[k]=data[i];data[i]=t;
        }
        f(data,k+1);
        {
             char t=data[k];data[k]=data[i];data[i]=t;
        }
    }
int main(void)
    char data[]={'A','B','C'};
    f(data,0);
    return 0;
```

```
}
```

五十,猴子吃桃

```
/*
```

小猴子第一天摘下若干桃子,当即吃掉一半,又多吃一个.第二天早上又将剩下的桃子吃一半,

又多吃一个.以后每天早上吃前一天剩下的一半另一个.到第 10 天早上猴子想再吃时发现

只剩下一个桃子了.问第一天猴子共摘多少个桃子? 输出:

```
1534
```

```
*/
#include "stdio.h"
int fruit(int begin,int times)//从反面考虑,用递归解决
{
    if(times==10) return begin;//注意递归出口
    return fruit((begin+1)*2,times+1);
}
int main(void)
{
    printf("%d\n",fruit(1,1));
    return 0;
```

五十一,角谷定理(递归解法)

/*

}

角谷定理。输入一个自然数,若为偶数,则把它除以 2,若为奇数,则把它乘以 3 加 1。 经过如此有限次运算后,总可以得到自然数值 1。求经过多少次可得到自然数 1。 如:输入 22,

```
输出 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1 STEP=16 */
```

```
#include "stdio.h"
int count=1;//用于记录 step
int f(int n)
{
    if(n==1)
```

```
{
         printf("%d ",n);
         return count;
    }
     else if(n%2==0)
     {
         printf("%d ",n);
         f(n/2);
         count++;
    }
    else
     {
         printf("%d ",n);
         f(3*n+1);
         count++;
    }
}
int main(void)
{
     int n,step;
     printf("Enter n:");
    scanf("%d",&n);
     step=f(n);
     printf("\n");
     printf("step=%d\n",step);
     return 0;
}
```

五十二, 高斯日记

/*

高斯日记

大数学家高斯有个好习惯, 无论如何都要记日记。

他的日记有个与众不同的地方,他从不注明年月日,而是用一个整数代替。比如, 4210 后来人们知道那个整数就是日期,它表示那一天是高斯出生后的第几天。这或许 也是个好习惯,它时时刻刻提醒着主人,日子又过去一天。还有多少时光可以用于浪费 呢况

高斯出生于 1777 年 4 月 30 日。

在高斯发现的一个重要定理的日记上标注着 5343。因此可算出那天是 1791 年 12 月 15 日。

高斯获得博士学位的那天日记上标着 8113 请你算出高斯获得博士学位的年月日。 提交答案的格式是 yyyy-mm-dd, 例如 1980-03-21

```
输出: 1799-7-16
   */
#include "stdio.h"
int leapYear(int year) //判断是否是闰年
   if((year%4==0&&year%100!=0)||year%400==0)
       return 1;
   else
       return 0;
}
int main(void)
{
   int year=1791,month=12,day=15;
   int t=0,i,n;//t 用于保存天数,i 是控制循环的角标,n 用于计算距离高斯发现重要定
理那天
             //用于操作天数
   int d=day;
   int arr[]={31,28,31,30,31,30,31,30,31,30,31};//定义好一年中每个月的天数
   n=8113-5343==2770; //距离高斯出生 2770 天
   for(i=11;t<=2770;i++) //循环条件 t 小于等于 2770
       if(i>=12) //角标对应 0——11, 所以到 12 要从新为 0, 年+1
       {
           i=0;
           year++;
       if(leapYear(year))
           arr[1]=29;
       else
           arr[1]=28;
       d=2770-t;
       t=t+arr[i];
       month=i+1; //角标对应 0——11, 加 1 就对应 1——12 了
   printf("%d-%d-%d\n",year,month,d+day);//天数还要加上原先的 15 天
   return 0;
}
```

五十三,马虎的算式

```
/*
马虎的算式
小明是个急性子,上小学的时候经常把老师写在黑板上的题目抄错了。
有一次,老师出的题目是 36 x 495 = ?
```

```
他却给抄成了,396 x 45 =?
但结果却很戏剧性,他的答案竟然是对的
因为 36 * 495 = 396 * 45 = 17820
类似这样的巧合情况可能还有很多,比如,27*594=297*54
假设 abcde 代表 1~9 不同的 5个数字,注意是各不相同的数字,且不含 0
能满足形如, ab * cde = adb * ce 这样的算式一共有多少种呢况
 输出: 142 种
*/
#include<stdio.h>
int main(void)
{
   int a,b,c,d,e;
   int sum=0; //用于记录个数
   for(a=1;a<10;a++)
       for(b=1;b<10;b++)
          for(c=1;c<10;c++)
              for(d=1;d<10;d++)
                  for(e=1;e<10;e++)
                     if((10*a+b)*(100*c+10*d+e)==(a*100+d*10+b)*(c*10+e)
           \&\&a!=b\&\&a!=c\&\&a!=d\&\&a!=e\&\&b!=c\&\&b!=d\&\&b!=e\&\&c!=d\&\&c!=e\&\&d!=e)
                     {//注意是各不相同的数字
                         sum++;
                         //printf("%3d,%3d,%3d,%3d,%3d\n",a,b,c,d,e);
                     }
                     printf("%4d 种\n",sum);
                     return 0;
}
```

五十四,黄金连分数

/*

黄金连分数

黄金分割数 0.61803...是个无理数,这个常数十分重要,在许多工程问题中会出现。有时需要把这个数字求得很精确。对于某些精密工程,常数的精度很重要。也许你听说过哈勃太空望远镜

,它首次升空后,就发现了一处人工加工错误,对那样一个庞然大物,其实只是镜面加工时 有比头发丝还细许

多倍的一处错误而已,却使它成了"近视眼"!!

言归正传,我们如何求得黄金分割数的尽可能精确的值呢,有许多方法。

比较简单的一种是用连分数

1 黄金数 =-----

```
1
1 + ------
1
1 + ------
1 + -----
1 + ...
```

这个连分数计算的"层数"越多,它的值越接近黄金分割数。

请你利用这一特性,求出黄金分割数的足够精确值,要求四舍五入到小数点后 100 位。

小数点后 3 位的值为:0.618

小数点后 4 位的值为:0.6180

小数点后 5 位的值为:0.61803

小数点后 7 位的值为:0.6180340

注意尾部的 0 不能忽略

你的任务是:写出精确到小数点后 100 位精度的黄金分割值。

注意:尾数的四舍五入尾数是0也要保留

输出:

 $0.618033988749894848204586834365638111089200288055402654977955065422553\\430723954$

```
4920647070512445164786
 //结果后面的精度根据 N 的值不同,结果尾部的精度有点偏差
*/
#include "stdio.h"
#define F85 //更改 F的值越大,精度越大,但不能太大,大了会溢出
//通过推算可知
//求这个的过程相当于斐波那契相邻的数相除 fib[n-1]/fib[n-2]
int main(void)
{
   long long int fib[1000];
   int f=0,i;
   long long int x,y;
   int a[101];
   int count=0;
   fib[0]=1;
   fib[1]=1;
   for(i=2;fib[i]<1e18;i++)//1e18 就是 1*10^18
       fib[i]=fib[i-1]+fib[i-2];
       f++;
   }
   //printf("%d \n",f);
   x=fib[F-2];
```

76/133

y=fib[F-1];

```
五十五,前缀判断
/*
前缀判断
如下的代码判断 needle_start 指向的串是否为 haystack_start 指向的串的前缀,如不是,
则返回 NULL。
比如: "abcd1234" 就包含了 "abc" 为前缀
输出: 是
*/
#include "stdio.h"
char *prefix(char *needle_start,char *haystack_start)
{
    char *haystak=haystack_start;
    char *needle=needle_start;
    while(*haystak&&*needle)
    {
        if(*(needle++)!=*(haystak++))
            return NULL;
    }
    if(!needle)
        return NULL;
    return haystak;
}
int main(void)
{
    char needle_start[100]={"abc"};
    char haystack_start[100]={"abc123"};
    if(prefix(needle_start,haystack_start))
        printf("是\n");
```

```
else
printf("否\n");
return 0;
}
```

五十六,三部排序

```
/*
```

三部排序

一般的排序有许多经典算法,如快速排序、希尔排序等。

但实际应用时,经常会或多或少有一些特殊的要求。我们没必要套用那些经典算法,可以根据实际情况建立更好的解法。

比如,对一个整型数组中的数字进行分类排序:

使得负数都靠左端,正数都靠右端,0在中部。注意问题的特点是:

负数区域和正数区域内并不要求有序。可以利用这个特点通过 1 次线性扫描就结束战斗!!

以下的程序实现了该目标。

其中 x 指向待排序的整型数组, len 是数组的长度。

```
输入:
        Enter 10 num:1 0 1 0 -2 -8 3 0 5 -10
输出: -10-2-80003511
*/
#include"stdio.h"
#define Len 10
void Sort3p(int *x,int len)
{
    int p=0,temp;//temp 用于保存要交换的变量
    int left=0;
    int right=len-1;
    while(p<=right)
    {
        if(x[p]<0) //p++根据 left
            temp=x[left];
            x[left]=x[p];
            x[p]=temp;
             p++;
            left++;
        }
        else if(x[p]>0)//p 不变
        {
             temp=x[right];
             x[right]=x[p];
```

```
x[p]=temp;
              right--;
         }
         else
              p++;
    }
}
int main(void)
{
    int x[Len];
    int i;
    printf("Enter %d num:",Len);
    for(i=0;i<Len;i++)
    {
         scanf("%d",&x[i]);
    }
    Sort3p(x,Len); //三部排序
    for(i=0;i<Len;i++)
    {
         printf("%d ",x[i]);
    }
}
```

五十七,翻硬币

/*

翻硬币

小明正在玩一个"翻硬币"的游戏。

桌上放着排成一排的若干硬币。我们用 * 表示正面,用 o 表示反面(是小写字母,不是零)。

比如,可能情形是: **oo***oooo

如果同时翻转左边的两个硬币,则变为: 0000***0000

现在小明的问题是:如果已知了初始状态和要达到的目标状态,每次只能同时翻转相邻的两个硬币.

那么对特定的局面,最少要翻动多少次呢?

我们约定:把翻动相邻的两个硬币叫做一步操作,那么要求:

程序输入:

两行等长的字符串,分别表示初始状态和要达到的目标状态。每行的长度**<1000**程序输出:

一个整数,表示最小操作步数

例如:

用户输入:

```
0****0****
程序应该输出:
再例如:
用户输入:
*0**0***0***
*0***0**0***
程序应该输出:
*/
#include<string.h>
#include<stdio.h>
void turn(char *c,int k) //只会改变自己和相邻
{
    int j;
      for(j=k;j<=k+1;j++)
        {
           if(c[j]=='*')
                 c[j]='o';
                 continue;
                }
          if(c[j]=='o')
               {
                 c[j]='*';
                 continue;
               }
        }
}
void handle(char *a,char *b)
{
    int i,count=0;
    char c[100];
    strcpy(c,a); //c 用来暂时代替 a,以防 a 以后有用。
    for(i=0;c[i]!='\0';i++)
        if(c[i]!=b[i])
        {
            turn(c,i);
            count++;
    printf("%d \n",count);
}
```

```
int main(void)
               //方法一
{
    char m[100]={0},n[100]={0};
    gets(m); //起始状态
    gets(n); //目标状态
    handle(m,n);
    return 0;
}
                //方法二 方法二必须要有 getchar();
int main(void)
    char a[100]={0},b[100]={0};
    scanf("%s",a);
    getchar();
    scanf("%s",b);
    handle(a,b);
    return 0;
}
*/
```

五十八,李白打酒(递归)

/*

话说大诗人李白,一生好饮。幸好他从不开车。

一天,他提着酒壶,从家里出来,酒壶中有酒2斗。他边走边唱:

无事街上走, 提壶去打酒。

逢店加一倍, 遇花喝一斗。

这一路上,他一共遇到店 5 次,遇到花 10 次,已知最后一次遇到的是花,他正好把酒喝光了。

请你计算李白遇到店和花的次序,可以把遇店记为 a, 遇花记为 b。

则: babaabbabbabbbb 就是合理的次序。像这样的答案一共有多少呢?请你计算出所有可能方案的个数(包含题目给出的)。

输出:

bababaababbbbbb

babaabbabbabbbb

babaababbbbbabb

baabbbaabbabbbb

baabbabbbaabbbb

baabbabbabb

baababbbbbababb

abbbabaabbabbbb

abbbaabbbaabbbb

abbbaabbabbabb

```
abbabbbabaabbbb
abbabbbaabbbabb
abbabbabbbababb
ababbbbbbabababb
14
*/
#include <stdio.h>
int count=0;//用于计数
char order[16];//一共 15 次,最后一个方'\0'
void fun(int num, int flower, int store, int jiu)
{
   //num: 次数,jiu:就是酒,flower: 就是花,store: 商店
   if(jiu<0 || store>5 || flower>10)
   {
        return;
    }
   else if(num==13 && store==5 && jiu==2)
    {
        count++;
        printf("%s\n",order);
        return;
    }
    else if(num<13)
                  //先遇花,先遇店结果都是一样,顺序相反
    {
        order[num]='b';//遇到花
        fun(num+1,flower+1,store,jiu-1);
        order[num]='a'; //遇到店
        fun(num+1,flower,store+1,jiu*2);
    }
}
int main(void)
{
   //根据已经判断,最后两次遇到的必定是花;第13次之前酒的数为2斗。
   order[15]='\0';
   order[14]='b';
    order[13]='b';
    fun(0,0,0,2);
    printf("%d\n",count);
    return 0;
}
```

```
/*
#include <stdio.h>
int main(void)
{
   char order[16]; //遇到花的次序
   int n; //用二进制表示可能遇到的花与店的情况
   int i,num;
   int jiu; //酒的斗数
   int counta; //遇到店的次数
   int countb; //遇到花的次数
   int count=0; //满足条件的个数
   order[15]='\0';
   for(n=31; n<=7936; n++)
   {
       //根据已经判断,最后两次遇到的必定是花;第13次之前酒的数为2斗。
       order[14]='b';
       order[13]='b';
       jiu=2;
       counta=0;
       countb=0;
       num=n;
       for(i=0; i<=12; i++)
           if(num%2==0) //遇到的是花
           {
               jiu=jiu-1;
               order[i]='b';
               counta++;
               if(counta>8)
               {
                   break;
               }
           }
           else //遇到的是店
           {
               jiu=jiu*2;
               order[i]='a';
               countb++;
               if(countb>5)
               {
                   break;
               }
```

```
    num=num/2;
}

if(countb==5 && jiu==2)
{
    count++;
    printf("%s\n",order);
}

printf("%d\n",count);
return 0;
}
*/
```

六十, 普利姆算法(最小生成树)

/*

从连通网 $N=\{V,E\}$ 中的某一个节点 UO 出发,选择与它关联的具有最小权值的边 (UO,v),

将其顶点加入到生成树的顶点集合 U 中。以后每一步从一个顶点在 U 中,而另一个顶点不在

U中的各条边中选择权值最小的边(u,v),把它的顶点加入到集合中。如此,继续下去,

直到网中的所有顶点都加入到生成树顶点集合U中为止。

```
输出:
        3->5:2
        2->5:4
        0->2:1
        1->2:5
        4->1:3
*/
#include "stdio.h"
#define max 32767
#define n 6
void prim(int matrix[n][n],int v0)
    int i,j,k,min,lowcost[n],closet[n]/*记录每次数组中列的值*/;
    for(i=0; i<=n-1;i++)//初始化数组,是一个临时放权值的地方
        if (i!=v0)
       {
            lowcost[i]=matrix[v0][i];
            closet[i]=v0;//closet[j]=k:表示下一次到角标为 k 的行去(也是 k 节点)。
```

```
}
        else
            lowcost[i]=0;
        for(i=1;i<=n-1;i++)
        {
            for(min=max,j=0;j<=n-1;j++)
            //从 lowcost 数组中找出最小的权值出来,并记录它所在的列。每次的 min
值都是 32767。
                if (min>lowcost[j]&&lowcost[j]!=0)
                     min=lowcost[j];
                     k=j;
                }
            printf("%d->%d:%d\n",k,closet[k],min);//找到一次输出一次,
            lowcost[k]=0;
            for(j=0;j<=n-1;j++)//k 的值将会在此段发生变化。
                 if(matrix[k][j]<lowcost[j] && lowcost[j]!=0)</pre>
                {
                     lowcost[j]=matrix[k][j];
                     closet[j]=k;
                }
        }
}
int main(void)
    int matrix[n][n]={//初始化
        {max,6,1,5,max,max},
        {6,max,5,max,3,max},
        {1,5,max,max,6,4},
        {5,max,max,max,max,2},
        {max,3,6,max,max,6},
        {max,max,4,2,6,max}
    };
    prim(matrix,5);
    return 0;
}
```

六十一,深度优先遍历

//深度优先遍历在算法竞赛以及解决一些问题时经常会用到 /*

深度优先遍历是从某个顶点出发,首先访问这个节点,然后找出刚访问这个顶点的

第一个未被访问的邻节点,然后再一此节点为顶点,继续找它的下一个新的顶点进 行 访问,重复此步骤,直到所有的节点都被访问完为止。 输入: Enter 1:01 Enter 2:0 2 Enter 3:03 Enter 4:2 3 Enter 5:1 4 Enter 6:2 4 输出: 0 3 2 4 1 */ #include"stdio.h" #include"stdlib.h" #define m 100 #define e 6 #define n 6 int visited[m];//标记遍历过得节点 typedef struct node1 { int adjvertex;//当前节点的值 struct node1 *nextarc;//指向下一个节点 }glinklistnode; typedef struct node2 { int vertexinfo;//当前节点的信息 glinklistnode *firstart;//每个节点的指针开始 }glinkheadnode; void createadjlist(glinkheadnode g[])//用链表创建数据 { int i,j,k; glinklistnode *p; for(i=0;i<n;i++)//节点初始化 { g[i].vertexinfo=i; g[i].firstart=0; } 无向图 for(k=0;k<e;k++)//创建 { printf("Enter %d:",k+1); scanf("%d%d",&i,&j); p=(glinklistnode *)malloc(sizeof(glinklistnode));

```
p->adjvertex=j;
         p->nextarc=g[i].firstart;
         g[i].firstart=p;
         p=(glinklistnode *)malloc(sizeof(glinklistnode));
         p->adjvertex=i;
         p->nextarc=g[j].firstart;
         g[j].firstart=p;
    }
}
void dfs(glinkheadnode g[],int v0)//深度遍历
    glinklistnode *p;
    printf("%d\t",v0);
    visited[v0]=1;
    p=g[v0].firstart;
    while(p)
    {
         if(visited[p->adjvertex]==0)//visited[i]=0;表示未遍历过 =1,则已遍历过
             dfs(g,p->adjvertex);
         p=p->nextarc;
    }
}
int main(void)
    glinkheadnode g[m];
    createadjlist(g);//创建图地邻阶表
    dfs(g,0);//深度遍历
    return 0;
}
```

六十二,广度优先遍历

/*

广度优先遍历从某个顶点出发,首先访问这个顶点,然后找出这个结点的所有未被访问 的邻接点

访问完成后再访问这些结点中第一个邻接点的所有结点,重复此方法,直到所有结点都被访问完为止

一层一层的访问

```
输入:
```

Enter 1:0 1

Enter 2:1 2

Enter 3:13

Enter 4:2 4

Enter 5:2 5

```
Enter 6:3 6
输出:
                     2
    0
                              3
                                       4
                                                5
             1
*/
#include<stdio.h>
#define m 100
#define n 6
#define e 6
int visited[m];//访问标记数组
typedef struct
    int vertex[m]; //每个节点
    int edge[m][m]; //保存节点之间的关系
}gadjmatrix;
void createadjmatrix(gadjmatrix &g)//创建广度遍历的数据
{
    int i,j,k;
    for(i=0;i<n;i++)
    {
        g.vertex[i]=i;
    }
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            g.edge[i][j]=0;
        }
    }
    for(k=0;k<e;k++)//i 和 j 相关联, j 和 i 也相关联
        printf("Enter %d:",k+1);
        scanf("%d%d",&i,&j);
        g.edge[i][j]=g.edge[j][i]=1;
    }
void bfs(gadjmatrix g,int v0) //广度遍历 : 跟层序遍历很相似
    struct{
        int front, rear;
        int q[m];
    }queue;
    int i,j;
    printf("%d\t",v0);
    visited[v0]=1;
```

```
queue.front=queue.rear=0;
    queue.rear=(queue.rear+1)%m;
    queue.q[queue.rear]=v0;
    while(queue.front!=queue.rear)
    {
        queue.front=(queue.front+1)%m;
        i=queue.q[queue.front];
        for(j=0;j<n;j++)
        if(g.edge[i][j]==1\&\&visited[j]==0)
        {
            printf("%d\t",j);
            visited[j]=1;
            queue.rear=(queue.rear+1)%m;
            queue.q[queue.rear]=j;
        }
    }
}
int main(void)
{
    gadjmatrix g;
    //创建时必须有0节点,因为广度遍历从0开始
    createadjmatrix(g);//创建广度遍历的数据
    bfs(g,0);//广度遍历
    return 0;
}
```

六十三,两个物种

```
/*
```

假设有两种微生物 X 和 Y

X 出生后每隔 3 分钟分裂一次(数目加倍),Y 出生后每隔 2 分钟分裂一次(数目加倍)。一个新出生的 X,半分钟之后吃掉 1 个 Y,并且,从此开始,每隔 1 分钟吃 1 个 Y。现在已知有新出生的 X=10, Y=89,求 60 分钟后 Y 的数目。

如果 X=10, Y=90 呢?

本题的要求就是写出这两种初始条件下, 60 分钟后 Y 的数目。

```
输出: 1073741825
*/
#include "stdio.h"
int main(void)
{
    int X=10,Y=89;
    89 / 133
```

```
int i;
for(i=0;i<120;i++)//半分钟一个单位
{
    if(i%2==1)/*
```

因为X出生半分钟后就要吃Y,尔后每1分钟要吃Y,所以永远都是奇数个半分钟的时候吃Y,又因为此时X不会增长(题目为了减小讨论的复杂度),所以直接减X数量即可。

```
*/
    Y -= X;
    if(i%4==0)//每 2 分钟翻倍
        Y*=Y;
    if(i%6==0)//每 3 分钟翻倍, X 和 Y 的翻倍是相互独立的, 不需要另作讨论
        X*=X;
}
printf("%d\n",Y);
}
```

六十四, 选手答题

/*

选手答题

某电视台举办了低碳生活大奖赛。题目的计分规则相当奇怪:

每位选手需要回答 10 个问题(其编号为 1 到 10),越后面越有难度。

答对的,当前分数翻倍;答错了则扣掉与题号相同的分数(选手必须回答问题,不回答按错误处理)。

每位选手都有一个起步的分数为10分。

某获胜选手最终得分刚好是100分,如果不让你看比赛过程,

你能推断出他(她)哪个题目答对了,哪个题目答错了吗?

如果把答对的记为 1,答错的记为 0,则 10 个题目的回答情况可以用仅含有 1 和 0 的串来表示。

例如: 0010110011 就是可能的情况。

你的任务是算出所有可能情况。每个答案占一行。

输出:

```
1011010000

0111010000

0010110011

*/

#include "stdio.h"

char a[12];

void fact(int n,int store)

{

90 / 133
```

```
if(n==11&&store==100)
    {
        printf("%s\n",a);
        return;
   }
   if(n<11)
    {
        a[n]='1';
        fact(n+1,store*2); //答对的, 当前分数翻倍
        a[n]='0';
        fact(n+1,store-n);// 答错了则扣掉与题号相同的分数
   }
}
int main(void)
    int n, store;
   a[10]='\0';
   a[0]=' ';//a[0]必须赋一个字符的值,可将其设置成空格
    n=1,store=10;
   fact(n,store);
    return 0;
}
```

六十五, 比酒量

/*

有一群海盗(不多于 **20** 人),在船上比拼酒量。过程如下:打开一瓶酒,所有在场的人平分喝下,

有几个人倒下了。再打开一瓶酒平分,又有倒下的,再次重复...... 直到开了第 4 瓶酒,坐着的已经所剩无几,海盗船长也在其中。当第 4 瓶酒平分喝下后,大家都倒下了。 等船长醒来,发现海盗船搁浅了。他在航海日志中写到: "......昨天,

我正好喝了一瓶……奉劝大家,开船不喝酒,喝酒别开船……" 请你根据这些信息,推断开始有多少人,每一轮喝下来还剩多少人。

如果有多个可能的答案,请列出所有答案,每个答案占一行。

格式是:人数,人数,...

例如,有一种可能是: 20,5,4,2,0

输出:

20542

18932

15 10 3 2

12642

六十六, 盒子取球方法一(写到文件)

/*

今盒子里有n个小球,A、B两人轮流从盒中取球,每个人都可以看到另一个人取了多少个,

也可以看到盒中还剩下多少个,并且两人都很聪明,不会做出错误的判断。 我们约定:

每个人从盒子中取出的球的数目必须是: 1,3,7或者8个。

轮到某一方取球时不能弃权!

A 先取球,然后双方交替取球,直到取完。被迫拿到最后一个球的一方为负方(输方)请编程确定出在双方都不判断失误的情况下,对于特定的初始球数,A 是否能赢?程序运行时,从标准输入获得数据,其格式如下:

先是一个整数 n(n<100),表示接下来有 n 个整数。然后是 n 个整数,每个占一行(整数 <10000),

表示初始球数。 程序则输出 n 行,表示 A 的输赢情况(输为 0, 赢为 1)。

输入:

4

1

2

10 8

输出:

0:1

1:0

2:1

3:0

4:1

5:0

```
6:1
7:0
8:1
9:1
10:1
0
1
1
1
*/
#include <stdio.h>
#include <string.h>
int a[10001];
int b[4] = \{1, 3, 7, 8\};
int main(void)
{
   int i,n,j;
   freopen("2.out", "w", stdout);//写到当前目录下的 2.out 文件中
   memset(a,0, sizeof a);//初始化判定数组为 0 在 String.h 头文件中,可以直接调用
   a[0]=1;//说明对方取完了最后一堆,那么自己胜利
   for(i=1;i<=10000;i++)
       for(j=0; j<4; j++)
           if(i-b[j]>=0)
           {
               if(a[i-b[j]]==0)//有取法能致使对方输,那么自己就要赢
                   a[i]=1;
                   break;
               }
           }
   for(int k=0; k<=10; k++)//k 的值可以根据打出的需要,打印出来
       printf("%d:%d\n", k, a[k]);
   scanf("%d",&n);
   while(n--)
   {
       scanf("%d",&i);
       printf("%d\n", a[i]);//直接输出即可,不需要存储下来统一输出,PS:ACM 里都是
这样处理的。
   }
   return 0;
}
```

```
/*
```

盒子取球方法二

今盒子里有n个小球,A、B两人轮流从盒中取球,每个人都可以看到另一个人取了多少个,

也可以看到盒中还剩下多少个,并且两人都很聪明,不会做出错误的判断。 我们约定:

每个人从盒子中取出的球的数目必须是: 1, 3, 7或者8个。

轮到某一方取球时不能弃权!

A 先取球,然后双方交替取球,直到取完。被迫拿到最后一个球的一方为负方(输方)请编程确定出在双方都不判断失误的情况下,对于特定的初始球数,A 是否能赢?程序运行时,从标准输入获得数据,其格式如下:

先是一个整数 n(n<100),表示接下来有 n 个整数。然后是 n 个整数,每个占一行(整数 <10000),

表示初始球数。 程序则输出 n 行,表示 A 的输赢情况(输为 0,赢为 1)。输入:

Enter n:4 1 2 10 8

输出:

0 1 1

1 */

. 分析:

当 n=1 的时候,A 一定会输;当 n=1+[1,3,7,8]的时候,A 一定会赢,因为 A 可以取 1,3,7,8 对应的那个数,然后把 1 留给 B;也就是说,当 n 的值是一个必输的情况加 1,3,7,8 的时候,则 A 可以取到对应的值,则把必输的情况交给 B。我们根据这条规律,把 n 的所有取值 1-10000 都进行标为 1(必胜)或 0(必输),然后输出对应的标记即可。

```
*/
#include <stdio.h>
#include <string.h>
void fun(int *a)
{
    int b[4]={1,3,7,8};
    for(int i=1;i+8<10000;i++)
    {
        if(a[i]==0)//必败局势
        {
            //printf("%d ",i); //可以查看输出情况
            for(int j=0;j<4;j++)
        {
                 a[i+b[j]]=1;
```

```
}
         }
//
         else
              printf(" ");
//
    }
}
int main(void)
    int n,m;
    int a[10001]={0};
    fun(a);
     printf("\n");
     printf("Enter n:");
    scanf("%d",&n);
    for(int i=0;i<n;i++)
         scanf("%d",&m);
         printf("%d\n",a[m]);
    }
     return 0;
}
```

六十八, 大数相乘

```
//输入两个数,输出两个数的相乘结果
/*
    输出:
         1082152022374638
*/
void bigmul(int x, int y, int r[])
{
    int base = 10000;
    int x2 = x / base;
    int x1 = x \% base;
    int y2 = y / base;
    int y1 = y \% base;
    int n1 = x1 * y1;
    int n2 = x1 * y2;
    int n3 = x2 * y1;
    int n4 = x2 * y2;
    r[3] = n1 \% base;
    r[2] = n1 / base + n2 % base + n3 % base;
    95/133
```

```
r[1] = n3/base+n2/base+n4%base;
r[0] = n4 / base;

r[1] += r[2]/base;
r[2] = r[2] % base;
r[0] += r[1] / base;
r[1] = r[1] % base;
}

int main(void)
{

    int x[] = {0,0,0,0};
    bigmul(87654321, 12345678, x);
    printf("%d%d%d%d\n",x[0],x[1],x[2],x[3]);
    return 0;
}

// 有些算法在你不明白时,最好在稿纸上走一遍,这样可以更好地理解算法
```

六十九,字母转换为6位数字(密码)

/*

//字母转换为6位数字(密码)

这个程序的任务就是把一串拼音字母转换为6位数字(密码)。

我们可以使用任何好记的拼音串(比如名字,王喜明,

就写: wangximing)作为输入,程序输出 6 位数字。变换的过程如下:

第一步. 把字符串 6个一组折叠起来,比如 wangximing 则变为:

wangxi

ming

第二步. 把所有垂直在同一个位置的字符的 ascii 码值相加,得出 6 个数字,如上面的例子,则得出:

228 202 220 206 120 105

第三步. 再把每个数字"缩位"处理: 就是把每个位的数字相加,得出的数字如果不是一位数字,

就再缩位,直到变成一位数字为止。例如: 228 => 2+2+8=12 => 1+2=3

上面的数字缩位后变为: 344836, 这就是程序最终的输出结果!

要求程序从标准输入接收数据,在标准输出上输出结果。

输入格式为:第一行是一个整数 n (<100),表示下边有多少输入行,接下来是 n 行字符串,

就是等待变换的字符串。

输出格式为: n 行变换后的 6 位密码。

如:

```
输入:
   1
   xuyisu
   输出:
                           6
                                   7
                                           9
   3
           9
                   4
*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
//全局变量 int 类型在定义时,会自动初始化为 0,也可不设为 0
int str[6],istr; // istr 用于 str 数组计数
int res[102][6],ires=0; //ires 用于记录 ires 组数据,一组 6 个
void getstr() //从键盘上获取数据
{
   char ch;
   while((ch=getchar())!='n')
   {
        str[istr%6]+=ch; //把所有垂直在同一个位置的字符的 ascii 码值相加
        istr++;
   }
}
void cal()
           //计算结果
{
   int i,t;
   for(i=0;i<6;i++)
   {
        while(str[i]>9) //这一段可以用一个数据加以理解:如 228
           t=str[i];
           str[i]=0;
           while(t)
           {
               str[i]+=t%10;
               t/=10;
           }
        }
   }
}
          //将结果保存入数组
void save()
{
   int i;
   for(i=0;i<6;i++)
```

```
res[ires][i]=str[i];
     ires++;
}
int main(void)
{
     int n,i,j;
     scanf("%d",&n);
     getchar();
     while(n--)
     {
          memset(str,0,sizeof(str));
          istr=0;
          getstr();
          cal();
          save();
     }
     for(i=0;i<ires;i++) //输出结果
     {
          for(j=0;j<6;j++)
              printf("%d\t",res[i][j]);
          printf("\n");
     }
     return 0;
}
```

七十, 打印图形

```
ran=6
#include "stdio.h"
```

```
#define N 70
void f(char a[][N], int rank, int row, int col)
     if(rank==1){
          a[row][col] = '*';
          return;
     }
     int w = 1;
     int i;
     for(i=0; i<rank-1; i++)
          w *= 2;
     f(a,rank-1,row,col+w/2);
     f(a,rank-1,row+w/2,col);
     f(a,rank-1,row+w/2,col+w);
}
int main(void)
{
     char a[N][N];
     int i,j;
     int n;
     for(i=0;i<N;i++)
          for(j=0;j<N;j++)
               a[i][j] = ' ';
     printf("Enter n:");
     scanf("%d",&n);
     f(a,n,0,0);
     for(i=0; i<N; i++){
          for(j=0;j<N;j++)
               printf("%c",a[i][j]);
          printf("\n");
     }
     return 0;
}
```

七十一,奇怪的分式

/*

人上小学的时候,小明经常自己发明新算法。一次,老师出的题目是: 1/4 乘以 8/5 小明居然把分子拼接在一起,分母拼接在一起,答案是: 18/45 老师刚想批评他,转念一想,这个答案凑巧也对啊,真是见鬼!

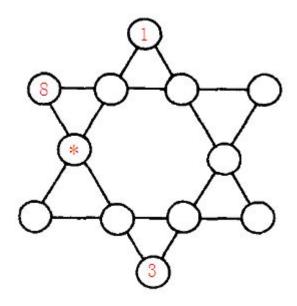
对于分子、分母都是 1~9 中的一位数的情况,还有哪些算式可以这样计算呢?请写出所有不同算式的个数(包括题中举例的)。

显然,交换分子分母后,例如: 4/1 乘以 5/8 是满足要求的,这算做不同的算式。但对于分子分母相同的情况, 2/2 乘以 3/3 这样的类型太多了,不在计数之列!注意:答案是个整数(考虑对称性,肯定是偶数)。请通过浏览器提交。不要书写多余的内容。

```
输出:
1/2*5/4
1/4*8/5
1/6*4/3
1/6*6/4
1/9*9/5
2/1*4/5
2/6*6/5
4/1*5/8
4/9*9/8
6/1*3/4
6/1*4/6
6/2*5/6
9/1*5/9
9/4*8/9
14
*/
#include <stdio.h>
#include <string.h>
int main(void)
{
    int count=0;
    int fz1,fz2,fm1,fm2;
    for(fz1=1; fz1<=9; fz1++)
    {
        for(fm1=1; fm1<=9; fm1++)
             if(fz1 != fm1)
                 for(fz2=1; fz2<=9; fz2++)
                      for(fm2=1; fm2<=9; fm2++)
                      {
                          if(fz2 != fm2)
```

七十二,六角填数

/*如图所示六角形中,填入 1~12 的数字。 使得每条直线上的数字之和都相同。



图中,已经替你填好了3个数字,请你计算星号位置所代表的数字是多少?

输出: 10 */

```
#include <stdio.h>
#define N 13
short int vis[N];
short int a[N];
void tianchong(int x);//函数申明
int main(void)
{
    //设置已经有的数
    vis[1]=1;
    a[1]=1;
    vis[8]=1;
    a[2]=8;
    vis[3]=1;
    a[12]=3;
    tianchong(1);
    return 0;
}
void tianchong(int x)
{
    if(x==1 | | x==2 | | x==12)
    {
         tianchong(x+1);
    }
    if(x>12) //a[]数组都填充了各不相同的数据
    {
         int t[6];
         t[0]=a[1]+a[3]+a[6]+a[8];
         t[1]=a[1]+a[4]+a[7]+a[11];
         t[2]=a[2]+a[3]+a[4]+a[5];
         t[3]=a[2]+a[6]+a[9]+a[12];
         t[4]=a[8]+a[9]+a[10]+a[11];
         t[5]=a[12]+a[10]+a[7]+a[5];
         for(int i=0; i<6; i++)
         {
             if(t[i] != 26) return;
         }
         printf("%d",a[6]);
         return;
    }
    else
```

```
{
    for(int i=1; i<=12; i++)
    {
        if(vis[i] ==0 && i!=1 && i!=3 && i!=8)
        {
            vis[i]=1;
            a[x]=i;
            tianchong(x+1);
            vis[i]=0; //如果填充不到最后一个数据,则返回,重新赋值,将 vis[i]
重置为 0
        }
    }
}
```

七十三, 蚂蚁感冒

/*蚂蚁感冒

长 100 厘米的细长直杆子上有 n 只蚂蚁。它们的头有的朝左,有的朝右。

每只蚂蚁都只能沿着杆子向前爬,速度是1厘米/秒。

当两只蚂蚁碰面时,它们会同时掉头往相反的方向爬行。

这些蚂蚁中,有1只蚂蚁感冒了。并且在和其它蚂蚁碰面时,会把感冒传染给碰到的蚂蚁。

(只数增加前题条件:一只感冒,一只没有感冒)

请你计算, 当所有蚂蚁都爬离杆子时, 有多少只蚂蚁患上了感冒。

变量:

n: 个数

表示蚂蚁离开杆子左边端点的距离和方向: ant[50]

感冒状态 gm[50] 1 为感冒 0 为非感冒

条件:

爬离杆子: if (ant[n]==0 || ant[n]==100) ant[n]==101, n--

所有蚂蚁都爬离杆子: num 杆子上的蚂蚁数, num=n, num==0

蚂蚁碰面: ant[i]==-ant[j]

只数增加: if(gm[i]+gm[j]==1) {gm[i]=1;gm[j]=1}

第一行输入一个整数 n (1 < n < 50), 表示蚂蚁的总数。

接着的一行是 n 个用空格分开的整数 Xi (-100 < Xi < 100), Xi 的绝对值,

表示蚂蚁离开杆子左边端点的距离。正值表示头朝右,负值表示头朝左,

数据中不会出现 0 值,也不会出现两只蚂蚁占用同一位置。其中,第一个数据代表的蚂蚁感冒了。

要求输出1个整数,表示最后感冒蚂蚁的数目。

```
例如,输入:
3
5 -2 8
程序应输出:
1
再例如,输入:
-10 8 -20 12 25
程序应输出:
3
*/
#include "stdio.h"
#include <math.h>
#define N 50
void mygm(int x[N][2],int n,int numAnt,int numGM);//函数申明
int main(void)
{
    int x[N][2];
    int n; //蚂蚁总数
    int numAnt; //杆子上的蚂蚁总数
    int numGM; //杆子上感冒的蚂蚁总数
    int i;
    printf("Enter n:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&x[i][0]);
        x[i][1]=0;
   }
    x[0][1]=1; //第一只蚂蚁感冒
    mygm(x,n,n,1);
    return 0;
}
void mygm(int x[N][2], int n,int numAnt,int numGM)
{
    if(numAnt==0)//如果杆子上的蚂蚁总数为 0
        printf("%d\n",numGM);
        return;
    for(int i=0;i<n; i++)
    105 / 133
```

```
{
         if(fabs(x[i][0])==100 || (x[i][0]==0) )//爬离杆子
         {
             numAnt--;
             x[i][0]=101; //表示已经不在杆子上
         }
         for(int j=0; j<n; j++)
             if(i!=j)
             {
                  //printf("numGm=%d\n",numGM);
                  if(x[i][0] == -x[j][0] && x[i][0]!=101 && x[j][0]!=101)
                  {
                       //printf("\n %d %d\n",x[i][0],x[j][0]);
                       x[i][0]=-x[i][0]; //转向
                       x[j][0]=-x[j][0];
                       if(x[i][1]+x[j][1]==1) //如果一方感冒
                       {
                           x[i][1]=1;
                           x[j][1]=1; //都感冒了
                           numGM++;
                       }
                  }
             }
         if(x[i][0]!=101 \&\& x[i][0]!=0 \&\& x[i][0]!=100)
         {
             x[i][0]++;
         //printf("%d ", x[i][0]);
    }
    //printf("\n");
    mygm(x,n,numAnt,numGM);
}
```

七十四, 地宫取宝

/*

X 国王有一个地宫宝库。是 nxm 个格子的矩阵。每个格子放一件宝贝。每个宝贝贴着价值标签。

地宫的入口在左上角, 出口在右下角。

小明被带到地宫的入口, 国王要求他只能向右或向下行走。

走过某个格子时,如果那个格子中的宝贝价值比小明手中任意宝贝价值都大,

小明就可以拿起它(当然,也可以不拿)。

当小明走到出口时,如果他手中的宝贝恰好是 k 件,则这些宝贝就可以送给小明。请你帮小明算一算,在给定的局面下,他有多少种不同的行动方案能获得这 k 件宝贝。

输入一行 3 个整数,用空格分开: n m k (1<=n,m<=50, 1<=k<=12)

接下来有 n 行数据,每行有 m 个整数 Ci (0<=Ci<=12)代表这个格子上的宝物的价值 要求输出一个整数,表示正好取 k 个宝贝的行动方案数。该数字可能很大,输出它对 1000000007 取模的结果。

```
例如,输入:
234
123
425
程序应该输出:
5
再例如,输入:
232
123
215
程序应该输出:
14
*/
#include <stdio.h>
#define N 50
#define M 50
void xunbao(int digong[N][M], int i, int j,int max);//函数申明
long long sum=0;
int count=0;
int max=-1;
int n,m,k;
int main(void)
    int digong[N][M]; //地宫
    int i,j;
    scanf("%d%d%d",&n,&m,&k);
    for(i=0; i<n; i++)
        for(j=0; j<m; j++)
        {
            scanf("%d",&digong[i][j]); //初始输入
        }
    }
```

```
xunbao(digong,0,0,max);//从数组的左上角进入
    printf("%-64d\n",sum%1000000007);
    return 0;
}
void xunbao(int digong[N][M],int i,int j,int max)
    if(max<digong[i][j])</pre>
    {
         //printf("max=%d, digong[]=%d\n",max,digong[i][j]);
         max=digong[i][j];
         count++;
    }
    if(i==n-1 && j==m-1)
    {
         if(count>=k)
             long long s1=1;
             long long s2=1;
             for(int j=1; j<=k; j++)
                  s1=s1*(count-j-1);
                  s2=s2*j;
             }
              sum = sum + s1/s2;
         }
         return;
    }
    else
    {
         if(j<m-1 && i<n-1)
         {
             xunbao(digong,i,j+1,max);
             xunbao(digong,i+1,j,max);
         }
         if(i==n-1 && j<m-1)
             xunbao(digong,i,j+1,max);
         if(i<n-1 && j==m-1)
         {
              xunbao(digong,i+1,j,max);
         }
```

```
}
}
```

七十五,高精度加法

/*

问题描述

输入两个整数 a 和 b,输出这两个整数的和。a 和 b 都不超过 100 位。 算法描述

由于a和b都比较大,所以不能直接使用语言中的标准数据类型来存储。对于这种 问题,一般使用数组来处理。

定义一个数组 A, A[0]用于存储 a 的个位, A[1]用于存储 a 的十位, 依此类推。同样 可以用一个数组 B 来存储 b。

计算 c = a + b 的时候, 首先将 A[0]与 B[0]相加, 如果有进位产生,则把进位(即和 的十位数) 存入 r, 把和的个位数存入 C[0], 即 C[0]等于(A[0]+B[0])%10。然后计算 A[1] 与 B[1]相加,这时还应将低位进上来的值 r 也加起来,即 C[1]应该是 A[1]、B[1]和 r 三个 数的和. 如果又有进位产生,则仍可将新的进位存入到 r 中,和的个位存到 C[1]中。依 此类推,即可求出C的所有位。

最后将 C 输出即可。

输入格式

输入包括两行,第一行为一个非负整数 a,第二行为一个非负整数 b。两个整数都 不超过100位,两数的最高位都不是0。

输出格式

```
输出一行,表示a+b的值。
```

样例输入

```
20100122201001221234567890
```

2010012220100122

样例输出

```
20100122203011233454668012
*/
#include<stdio.h>
#include<String.h>
int main(void)
{
    int i:
    char a[101],b[101],c[102]={0};
    scanf("%s%s", a, b);
    int m = strlen(a);
    int n = strlen(b);
    for (i=0;i<m||i<n;i++)
    {
         if (i < m)
```

```
c[i]+= a[m-i-1]-'0';
          if (i < n)
                c[i]+=b[n-i-1]-'0';
          if (c[i] >= 10)
          {
                c[i+1] = c[i]/10;
                c[i]%=10;
          }
     }
      if(m<n)
          m=n;
      if(c[m]>0)
          printf("%d",c[m]);
      for(i=m-1;i>=0;i--)
          printf("%d",c[i]);
     return 0;
}
```

七十六, Huffman 树

/*

问题描述

Huffman 树在编码中有着广泛的应用。在这里,我们只关心 Huffman 树的构造过程。 给出一列数{pi}={p0, p1, ···, pn-1},用这列数构造 Huffman 树的过程如下:

- 1. 找到 $\{pi\}$ 中最小的两个数,设为 pa 和 pb,将 pa 和 pb 从 $\{pi\}$ 中删除掉,然后将它们的和加入到 $\{pi\}$ 中。这个过程的费用记为 pa + pb。
 - 2. 重复步骤 1, 直到{pi}中只剩下一个数。

在上面的操作过程中,把所有的费用相加,就得到了构造 Huffman 树的总费用。 本题任务: 对于给定的一个数列,现在请你求出用该数列构造 Huffman 树的总费用。

例如,对于数列{pi}={5, 3, 8, 2, 9}, Huffman 树的构造过程如下:

- 1. 找到{5, 3, 8, 2, 9}中最小的两个数,分别是 2 和 3,从{pi}中删除它们并将和 5 加入,得到{5, 8, 9, 5},费用为 5。
- 2. 找到 $\{5, 8, 9, 5\}$ 中最小的两个数,分别是 5 和 5,从 $\{pi\}$ 中删除它们并将和 10 加入,得到 $\{8, 9, 10\}$,费用为 10。
- 3. 找到{8, 9, 10}中最小的两个数,分别是 8 和 9,从{pi}中删除它们并将和 17 加入,得到{10, 17},费用为 17。
- 4. 找到{10, 17}中最小的两个数,分别是 10 和 17,从{pi}中删除它们并将和 27 加入,得到{27},费用为 27。
- 5. 现在,数列中只剩下一个数 27,构造过程结束,总费用为 5+10+17+27=59。 输入格式

输入的第一行包含一个正整数 n(n<=100)。

接下来是 n 个正整数, 表示 p0, p1, …, pn-1, 每个数不超过 1000。 输出格式

```
输出用这些数构造 Huffman 树的总费用。
样例输入
5
53829
样例输出
59
*/
#include <stdio.h>
int main(void)
{
     int i, j, k, l, n;
     int a[100];
     int count = 0;
     scanf("%d", &n);
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);
     for (i = n; i > 1; i--)//n 次
     {
          k = 0, l = 1;
          if (a[0] < a[1])
               k = 1, l = 0;
          for (j = 2; j < i; j++)
               if (a[j] \le a[k])
               {
                    if (a[j] < a[l])
                         k = I, I = j;
                    else
                         k = j;
               }
          }
          a[l] += a[k];
          count += a[l];
          for (j = k; j < i - 1; j++)
               a[j] = a[j + 1];
     }
     printf("%d\n", count);
     return 0;
}
```

七十七,报时助手

```
问题描述
   给定当前的时间,请用英文的读法将它读出来。
   时间用时 h 和分 m 表示, 在英文的读法中, 读一个时间的方法是:
   如果 m 为 0,则将时读出来,然后加上"o'clock",如 3:00 读作"three o'clock"。
   如果 m 不为 0,则将时读出来,然后将分读出来,如 5:30 读作"five thirty"。
   时和分的读法使用的是英文数字的读法,其中 0~20 读作:
   0:zero, 1: one, 2:two, 3:three, 4:four, 5:five, 6:six, 7:seven, 8:eight, 9:nine, 10:ten,
11:eleven, 12:twelve, 13:thirteen, 14:fourteen, 15:fifteen, 16:sixteen, 17:seventeen,
18:eighteen, 19:nineteen, 20:twenty.
   30 读作 thirty, 40 读作 forty, 50 读作 fifty。
   对于大于 20 小于 60 的数字, 首先读整十的数, 然后再加上个位数。如 31 首先读
30 再加 1 的读法,读作"thirty one"。
   按上面的规则 21:54 读作"twenty one fifty four", 9:07 读作"nine seven", 0:15
读作"zero fifteen"。
输入格式
   输入包含两个非负整数 h 和 m,表示时间的时和分。非零的数字前没有前导 0。h
小于 24, m 小于 60。
输出格式
   输出时间时刻的英文。
样例输入
0 15
样例输出
zero fifteen
*/
#include <stdio.h>
int main()
{
   int h, m;
   char* en0to19[] = {
       "zero", "one", "two", "three", "four", "five",
       "six", "seven", "eight", "nine", "ten", "eleven",
       "twelve", "thirteen", "fourteen", "fifteen",
       "sixteen", "seventeen", "eighteen", "nineteen"
   };
   char* en20to50[] = {
       "twenty", "thirty", "forty", "fifty"
   };
   char* oclick = "o'clock";
   scanf("%d%d", &h, &m);
   if (h >= 20)
   {
       printf("%s", en20to50[0]);
       if (h \% 20 > 0)
```

/*

```
printf(" %s", en0to19[h % 20]);
     }
     else
     {
          printf("%s", en0to19[h]);
     }
     if (m == 0)
     {
          printf(" %s", oclick);
     }
     else if (m \ge 20)
     {
          printf(" %s", en20to50[m / 10 - 2]);
          if (m \% 10 > 0)
               printf(" %s", en0to19[m % 10]);
     }
     else
     {
          printf(" %s", en0to19[m]);
     }
     return 0;
}
```

七十八,回形取数

/*

问题描述

回形取数就是沿矩阵的边取数,若当前方向上无数可取或已经取过,则左转 90 度。 一开始位于矩阵左上角,方向向下。

输入格式

输入第一行是两个不超过 200 的正整数 m, n, 表示矩阵的行和列。接下来 m 行每 行 n 个整数,表示这个矩阵。

输出格式

输出只有一行,共 mn 个数,为输入矩阵回形取数得到的结果。数之间用一个空格分隔,行末不要有多余的空格。

样例输入

3 3

123

456

789

样例输出

147896325

样例输入

```
32
12
3 4
56
样例输出
135642
*/
#include <stdio.h>
int main()
{
     int i, j, m, n;
     int a[200][200];
     scanf("%d%d", &m, &n);
     for (i = 0; i < m; i++)
           for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);
     for (i = 0; i < (n + 1) / 2 & i < (m + 1) / 2; i++)
     {
           for (j = i; j < m - i; j++)
                printf("%d ", a[j][i]);
           for (j = i + 1; j < n - i; j++)
                printf("%d ", a[m - i - 1][j]);
           if (n - i - 1 > i)
                for (j = m - i - 2; j >= i; j--)
                      printf("%d ", a[j][n - i - 1]);
           if (m - i - 1 > i)
                for (j = n - i - 2; j > i; j--)
                      printf("%d ", a[i][j]);
     }
     return 0;
}
```

七十九,龟兔赛跑预测

/*

问题描述

话说这个世界上有各种各样的兔子和乌龟,但是研究发现,所有的兔子和乌龟都有一个共同的特点——喜欢赛跑。于是世界上各个角落都不断在发生着乌龟和兔子的比赛,小华对此很感兴趣,于是决定研究不同兔子和乌龟的赛跑。他发现,兔子虽然跑比乌龟快,但它们有众所周知的毛病——骄傲且懒惰,于是在与乌龟的比赛中,一旦任一

秒结束后兔子发现自己领先 t 米或以上, 它们就会停下来休息 s 秒。对于不同的兔子, t, s 的数值是不同的,但是所有的乌龟却是一致——它们不到终点决不停止。

然而有些比赛相当漫长,全程观看会耗费大量时间,而小华发现只要在每场比赛开 始后记录下兔子和乌龟的数据——兔子的速度 v1 (表示每秒兔子能跑 v1 米),乌龟的 速度 v2,以及兔子对应的 t, s 值,以及赛道的长度 I——就能预测出比赛的结果。但是 小华很懒,不想通过手工计算推测出比赛的结果,于是他找到了你——清华大学计算机 系的高才生——请求帮助,请你写一个程序,对于输入的一场比赛的数据 v1, v2, t, s, I, 预测该场比赛的结果。

输入格式

输入只有一行,包含用空格隔开的五个正整数 v1, v2, t, s, I, 其中 (v1,v2<=100;t<=300;s<=10;l<=10000 且为 v1,v2 的公倍数) 输出格式

输出包含两行,第一行输出比赛结果——一个大写字母 "T"或 "R"或 "D",分 别表示乌龟获胜,兔子获胜,或者两者同时到达终点。

第二行输出一个正整数,表示获胜者(或者双方同时)到达终点所耗费的时间(秒 数)。

```
样例输入
1055220
样例输出
D
4
样例输入
1055120
样例输出
R
3
样例输入
1055320
样例输出
Т
4
*/
#include <stdio.h>
int main(void)
{
   int v1, v2, t, s, l;//v1 兔子的速度, v2 乌龟的速度, 兔子领先 t 米, 就会停下来休息 s
秒,赛道的长度1
   scanf("%d%d%d%d%d", &v1, &v2, &t, &s, &I);
   int i, t1 = 0, t2 = I / v2; //t1 兔子的时间, t2 乌龟的时间
   int s1 = 0, s2 = 0;//兔子和乌龟各跑的路程
   for (i = 1; i <= t2; i++)
       s2 += v2;//乌龟每秒跑的距离
       if (t1 < i)
```

```
{
             t1 = i;
             s1 += v1;//兔子每秒跑的距离
         }
         if (s1 == I)//如果兔子跑到终点,程序结束
             break;
         if (t1 == i \&\& s1 - s2 >= t)
             t1 += s;//s 是兔子等的时间
    }
    if (s1 == 1 \&\& s2 == 1)
         printf("D\n%d", t2);
    else if (s1 == 1)
         printf("R\n%d", t1);
    else
         printf("T\n%d", t2);
    return 0;
}
```

八十,芯片测试

/*

问题描述

有 n (2≤n≤20) 块芯片,有好有坏,已知好芯片比坏芯片多。

每个芯片都能用来测试其他芯片。用好芯片测试其他芯片时,能正确给出被测试芯片是好还是坏。而用坏芯片测试其他芯片时,会随机给出好或是坏的测试结果(即此结果与被测试芯片实际的好坏无关)。

给出所有芯片的测试结果,问哪些芯片是好芯片。

输入格式

输入数据第一行为一个整数 n, 表示芯片个数。

第二行到第 n+1 行为 n*n 的一张表,每行 n 个数据。表中的每个数据为 0 或 1,在 这 n 行中的第 i 行第 j 列($1 \le i$, $j \le n$)的数据表示用第 i 块芯片测试第 j 块芯片时得到的测试结果,1 表示好,0 表示坏,i=j 时一律为 1(并不表示该芯片对本身的测试结果。芯片不能对本身进行测试)。

输出格式

按从小到大的顺序输出所有好芯片的编号

```
样例输入
```

```
#include <stdio.h>
int main(void)
     int i, j, k, l, n, a[20][20], b[20] = \{0\}, c[20] = \{0\};
     scanf("%d", &n);
     for (i = 0; i < n; i++)
           for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);
     l = 1;
     for (i = 0; i < n; i++)
           if (b[i] != 0)
                continue;
           b[i] = l++;//如果 b[i]==0,l++
           for (j = i + 1; j < n; j++)
                if (b[j]!=0)//说明 b[j]==0
                      continue;
                for (k = 0; k < n; k++)
                      if (a[i][k] != a[j][k])
                           break;
                if (k == n)
                      b[j] = b[i];
           }
     }
     int max = 0;
     for (i = 0; i < n; i++)
           c[b[i]]++;
     for (i = 0; i < l; i++)
           if (c[max] < c[i])
                max = i;
     for (i = 0; i < n; i++)
           if (max == b[i])
                printf("%d ", i + 1);
     return 0;
}
```

八十一,FJ 的字符串

```
/*
问题描述
FJ 在沙盘上写了这样一些字符串:
117/133
```

```
A1 = "A"
   A2 = "ABA"
   A3 = "ABACABA"
   A4 = "ABACABADABACABA"
   你能找出其中的规律并写所有的数列 AN 吗?
输入格式
   仅有一个数: N ≤ 26。
输出格式
   请输出相应的字符串 AN,以一个换行符结束。输出中不得含有多余的空格或换行、
回车符。
样例输入
样例输出
ABACABA
*/
#include <stdio.h>
#include <string.h>
int main(void)
{
   int i, j, n;
   char str[1024] = "A", str2[1024];
   scanf("%d", &n);
   for (i = 1; i < n; i++)
   {
       sprintf(str2, "%s%c%s", str, 'A' + i, str);//将字符打印到 str2 中
       strcpy(str, str2);//复制 str2 到 str1 中
   }
   printf("%s", str);
   return 0;
}
```

//有些算法在你不明白时,最好在稿纸上走一遍,这样可以更好地理解算法

八十二,Sine 之舞

问题描述

最近 FJ 为他的奶牛们开设了数学分析课, FJ 知道若要学好这门课, 必须有一个好 的三角函数基本功。所以他准备和奶牛们做一个"Sine之舞"的游戏,寓教于乐,提高 奶牛们的计算能力。

```
不妨设
```

```
An=sin(1 - sin(2+sin(3 - sin(4+...sin(n))...)
Sn=(...(A1+n)A2+n-1)A3+...+2)An+1
118 / 133
```

```
FJ 想让奶牛们计算 Sn 的值,请你帮助 FJ 打印出 Sn 的完整表达式,以方便奶牛们
做题。
输入格式
    仅有一个数: N<201。
输出格式
    请输出相应的表达式Sn,以一个换行符结束。输出中不得含有多余的空格或换行、
回车符。
样例输入
3
样例输出
((\sin(1)+3)\sin(1-\sin(2))+2)\sin(1-\sin(2+\sin(3)))+1
*/
#include <stdio.h>
#include<String.h>
#define N 200000
//sprintf 与 printf 功能一样,将字符打印到一个字符串数组中去而已
int main()
{
    char* sinA = "sin(%d%s%s)";
    char* Sa = "%s+%d";
    char* S = "(%s)%s";
    char An[N] = "sin(1%s%s)", An2[N], An3[N], Sn[N] = "sin(1)+%d", Sn2[N];
    int i, j, n;
    scanf("%d", &n);
    sprintf(Sn2, Sn, n);
    strcpy(Sn, Sn2);
    for (i = 2; i <= n; i++)
    {
        sprintf(An2, An, i % 2 == 0 ? "-" : "+", sinA);
        strcpy(An, An2);
        sprintf(An2, An, i, "%s", "%s");
        strcpy(An, An2);
        sprintf(An2, Sa, An, n - i + 1);
        sprintf(An3, An2, "", "");
        sprintf(Sn2, S, Sn, An3);
        strcpy(Sn, Sn2);
        sprintf(Sn2, Sn, n - i+ 1);
        strcpy(Sn, Sn2);
    printf("%s", Sn);
    return 0;
```

}

/*

问题描述

Tom 教授正在给研究生讲授一门关于基因的课程,有一件事情让他颇为头疼:一条染色体上有成千上万个碱基对,它们从 0 开始编号,到几百万,几千万,甚至上亿。

比如说,在对学生讲解第 **1234567009** 号位置上的碱基时,光看着数字是很难准确的念出来的。

所以,他迫切地需要一个系统,然后当他输入 12 3456 7009 时,会给出相应的念法: 十二亿三千四百五十六万七千零九

用汉语拼音表示为

shi er yi san qian si bai wu shi liu wan qi qian ling jiu

这样他只需要照着念就可以了。

你的任务是帮他设计这样一个系统:给定一个阿拉伯数字串,你帮他按照中文读写的规范转为汉语拼音字串,相邻的两个音节用一个空格符格开。

注意必须严格按照规范,比如说"10010"读作"yi wan ling yi shi"而不是"yi wan ling shi","100000"读作"shi wan"而不是"yi shi wan","2000"读作"er qian"而不是"liang qian"。

输入格式

有一个数字串,数值大小不超过 2,000,000,000。

输出格式

是一个由小写英文字母,逗号和空格组成的字符串,表示该数的英文读法。

if (i >= 4 && !![i / 4 - 1]) I[i / 4 - 1] = 1, str[j++] = wei[i / 4 + 2];

样例输入

```
1234567009
```

样例输出

```
shi er yi san qian si bai wu shi liu wan qi qian ling jiu
*/
#include <stdio.h>
int main(void)
{
    char* num[] = {"ling","yi","er","san","si","wu","liu","qi","ba","jiu"};
    char* wei[] = {"shi","bai","qian","wan","yi"};
    char* str[20];
    int n;
    scanf("%d", &n);
    int i = 0, j = 0, k, l[2] = {0};
    while (n > 0)
    {
        k = n % 10;
    }
}
```

if (i % 4 != 0) str[j++] = wei[i % 4 - 1];

n /= 10; if (k > 0) {

{

if (i > 0)

```
}
                 str[j++] = num[k];
           ext{le | } else if (j > 0 && str[j - 1] != num[0]) {
                 str[j++] = num[0];
           }
           i++;
     if (!(str[j-1] == num[1] \&\& j > 1 \&\& str[j-2] == wei[0]))
           printf("%s ", str[j - 1]);
     for (i = j - 2; i >= 0; i--)
           printf("%s ", str[i]);
     return 0;
}
```

八十四,完美的代价

```
/*
```

回文串,是一种特殊的字符串,它从左往右读和从右往左读是一样的。小龙龙认为 回文串才是完美的。现在给你一个串,它不一定是回文的,请你计算最少的交换次数使

```
问题描述
得该串变成一个完美的回文串。
   交换的定义是:交换两个相邻的字符
  例如 mamad
  第一次交换 ad: mamda
  第二次交换 md: madma
  第三次交换 ma: madam (回文! 完美!)
输入格式
  第一行是一个整数 N,表示接下来的字符串的长度(N <= 8000)
  第二行是一个字符串,长度为 N. 只包含小写字母
输出格式
   如果可能,输出最少的交换次数。
   否则输出 Impossible
样例输入
5
mamad
样例输出
3
*/
#include <stdio.h>
void swap(char* a, char* b)
{
  char c = *a;
```

```
*a = *b;
     *b = c;
int find(char* p, char* q)//左边查找
{
     char* o = q - 1;
     while (o > p \&\& *o != *p) o--;
     return o - p;
}
int rfind(char*p, char* q)//右边查找
     char* o = q + 1;
     while (o 
     return p - o;
}
int main(void)
{
     int i, j, k, len, count[26] = {0};
     char str[8000];
     scanf("%d", &len);
     scanf("%s", str);
     for (i = 0; i < len; i++)
          count[str[i]-'a']++;
     int odd = 0;
     for (i = 0; i < 26; i++)
          if (count[i] % 2 == 1)
               odd++;
     if (odd <= 1)
     {
          int change = 0;
          for (i = 0; i < len / 2; i++)
          {
               if (count[str[i]-'a'] == 1)
                    break;
               count[str[i]-'a'] -= 2;
               k = find(str + i, str + len - i);
               for (j = i + k; j < len - i - 1; j++)
               {
                    swap(str + j, str + j + 1);
                    change++;
               }
          }
          if (i != len / 2)
          {
```

```
for (i = len - i - 1; i > len / 2; i--)
                 {
                      k = rfind(str + i, str + len - i - 1);
                      for (j = i - k; j > len - i - 1; j--)
                            swap(str + j, str + j - 1);
                            change++;
                      }
                 }
           }
           printf("%d", change);
     }
     else
     {
           printf("Impossible");
     }
     return 0;
}
```

八十五, 矩形的面积交

/*

问题描述

平面上有两个矩形,它们的边平行于直角坐标系的 X 轴或 Y 轴。对于每个矩形,我们给出它的一对相对顶点的坐标,请你编程算出两个矩形的交的面积。

输入格式

输入仅包含两行,每行描述一个矩形。

在每行中,给出矩形的一对相对顶点的坐标,每个点的坐标都用两个绝对值不超过 10^7 的实数表示。

输出格式

输出仅包含一个实数,为交的面积,保留到小数后两位。

样例输入

```
1133
2244
样例输出
1.00
*/
#include <stdio.h>
double dabs(double a)
{
    return a > 0 ? a : -a;
}
int in(double x, double y, double* a)
```

```
{
     return (((x \ge a[0] \&\& x \le a[2] \mid | x \le a[0] \&\& x \ge a[2]) \&\&
          (y \ge a[1] \&\& y \le a[3] \mid | y \le a[1] \&\& y \ge a[3])) ? 1 : 0);
double part(double a, double b, double c, double d)
{
     return ((a > c && a < d || a < c && a > d) ? a : b);
}
int count(double* a, double* b, int* c)
{
     c[0] = in(a[0], a[1], b);
     c[1] = in(a[0], a[3], b);
     c[2] = in(a[2], a[3], b);
     c[3] = in(a[2], a[1], b);
     return c[0] + c[1] + c[2] + c[3];
}
int one(int* a)
{
     return (a[0]?0:a[1]?1:a[2]?2:3);
int rone(int* a)
{
     return (a[3]?3:a[2]?2:a[1]?1:0);
}
double mixed1(int* a, int* b, double* c, double* d)
     return dabs((c[one(a) < 2 ? 0 : 2] - d[one(b) < 2 ? 0 : 2]) *
          (c[one(a) \% 3 == 0 ? 1 : 3] - d[one(b) \% 3 == 0 ? 1 : 3]));
double mixed2(int* a, double* b, double* c)
     return dabs((one(a) / 2 == rone(a) / 2)
          ? (b[1] - b[3]) * (b[one(a) / 2 * 2] - part(c[0], c[2], b[0], b[2]))
          (b[0] - b[2]) * (b[one(a) % 3 == 0 ? 1 : 3] - part(c[1], c[3], b[1], b[3])));
double mixed4(double* a)
     return dabs((a[0] - a[2]) * (a[1] - a[3]));
int main(void)
     int i, j;
     double coords[2][4];
     for (i = 0; i < 2; i++)
```

```
for(j = 0; j < 4; j++)
               scanf("%lf", &coords[i][j]);
     int in[2][4];
     int a = count(coords[0], coords[1], in[0]);
     int b = count(coords[1], coords[0], in[1]);
     double mixed = 0;
     if (a == 1) {
          mixed = mixed1(in[0], in[1], coords[0], coords[1]);
     else if (a == 2) {
          mixed = mixed2(in[0], coords[0], coords[1]);
     } else if (a == 4) {
          mixed = mixed4(coords[0]);
     } else if (b == 2) {
          mixed = mixed2(in[1], coords[1], coords[0]);
     } else if (b == 4) {
          mixed = mixed4(coords[1]);
     }
     printf("%.2lf", mixed);
     return 0;
}
```

八十六,矩阵乘法

```
/*
问题描述
    给定一个 N 阶矩阵 A, 输出 A 的 M 次幂 (M 是非负整数)
例如:
    A =
    12
    34
    A 的 2 次幂
    7 10
    15 22
输入格式
```

第一行是一个正整数 N、M(1<=N<=30, 0<=M<=5),表示矩阵 A 的阶数和要求的 幂数

接下来 N 行,每行 N 个绝对值不超过 10 的非负整数,描述矩阵 A 的值输出格式

输出共N行,每行N个整数,表示A的M次幂所对应的矩阵。相邻的数之间用一个空格隔开

```
样例输入
22
12
3 4
样例输出
7 10
15 22
*/
#include <stdio.h>
int main(void)
{
     int a[30][30], b[30][30] = {0}, c[30][30];
     int i, j, k, l, m, n;
     scanf("%d%d", &n, &m);
     for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
                scanf("%d", &a[i][j]);
     for (i = 0; i < n; i++)
          b[i][i] = 1;
     for (i = 0; i < m; i++)
          for (j = 0; j < n; j++)
                for (k = 0; k < n; k++)
                {
                     c[j][k] = 0;
                     for (I = 0; I < n; I++)
                           c[j][k] += a[j][l] * b[l][k];
                }
          for (j = 0; j < n; j++)
                for (k = 0; k < n; k++)
                     b[j][k] = c[j][k];
     }
     for (i = 0; i < n; i++)
     {
          for (j = 0; j < n; j++)
                printf("%d ", b[i][j]);
          printf("\n");
     }
     return 0;
}
```

```
问题描述
    求出区间[a,b]中所有整数的质因数分解。
输入格式
    输入两个整数 a, b。
输出格式
    每行输出一个数的分解, 形如 k=a1*a2*a3...(a1<=a2<=a3..., k 也是从小到大的)(具体
可看样例)
样例输入
3 10
样例输出
3=3
4=2*2
5=5
6=2*3
7=7
8=2*2*2
9=3*3
10=2*5
提示
    先筛出所有素数, 然后再分解。
数据规模和约定
    2<=a<=b<=10000
*/
#include <stdio.h>
#include <math.h>
int factor(n)
{
    int i, j = (int)sqrt(n);
    if (n \% 2 == 0) return 2;
    for (i = 3; i <= j; i++)
        if (n % i == 0) return i;
    return n;
}
int main(void)
{
    int i, j, k, m, n;
    scanf("%d%d", &m, &n);
    for (i = m; i \le n; i++)
        j = factor(i);
        k = i / j;
        printf("%d=%d", i, j);
    127 / 133
```

八十八,字符串对比

/*

问题描述

给定两个仅由大写字母或小写字母组成的字符串(长度介于 1 到 10 之间),它们之间的关系是以下 4 中情况之一:

- 1: 两个字符串长度不等。比如 Beijing 和 Hebei
- 2: 两个字符串不仅长度相等,而且相应位置上的字符完全一致(区分大小写),比如 Beijing 和 Beijing
- 3: 两个字符串长度相等,相应位置上的字符仅在不区分大小写的前提下才能达到 完全一致(也就是说,它并不满足情况 2)。比如 beijing 和 BEljing
- 4: 两个字符串长度相等,但是即使是不区分大小写也不能使这两个字符串一致。 比如 Beijing 和 Nanjing

编程判断输入的两个字符串之间的关系属于这四类中的哪一类,给出所属的类的编号。

输入格式

包括两行,每行都是一个字符串

输出格式

仅有一个数字, 表明这两个字符串的关系编号

```
样例输入
```

BEIjing

beiJing

样例输出

```
3
*/
```

#include <stdio.h>
#include <string.h>

int main(void)

```
{
```

char a[11], b[11];

scanf("%s%s", a, b);

if (strlen(a) != strlen(b)) printf("1");

```
else if (strcmp(a, b) == 0) printf("2");
else if (strcasecmp(a, b) == 0) printf("3");
else printf("4");
return 0;
}
```

八十九,时间转换

/*

问题描述

给定一个以秒为单位的时间 t,要求用 "<H>:<M>:<S>"的格式来表示这个时间。<H>表示时间,<M>表示分钟,而<S>表示秒,它们都是整数且没有前导的"0"。例如,若t=0,则应输出是"0:0:0";若t=3661,则输出"1:1:1"。

输入格式

输入只有一行,是一个整数 t(0<=t<=86399)。

输出格式

输出只有一行,是以 "<H>:<M>:<S>"的格式所表示的时间,不包括引号。

样例输入

0

样例输出

0:0:0

样例输入

5436

样例输出

1:30:36

*/

#include <stdio.h>

int main(void)
{

int n; scanf("%d", &n);

printf("%d:%d:%d", n / 3600, n % 3600 / 60, n % 60); return 0;

九十, 出现次数最多的整数

/*

}

问题描述

编写一个程序, 读入一组整数, 这组整数是按照从小到大的顺序排列的,

它们的个数 N 也是由用户输入的,最多不会超过 20。然后程序将对这个数组进行统计,把出现次数最多的那个数组元素值打印出来。如果有两个元素值出现的次数相同,即并列第一,

那么只打印比较小的那个值。

输入格式:第一行是一个整数 N, N 小于 20;接下来有 N 行,每一行表示一个整数,并且按照从小到大

的顺序排列。

```
输出格式:输出只有一行,即出现次数最多的那个元素值。
输入输出样例
```

```
样例输入
5
100
150
150
200
250
样例输出
150
*/
#include<stdio.h>
int main(void)
{
    int count[20];//用于计数
    int a[20];
    int n,i,j,max,temp;//max 求最大, temp 临时变量
    scanf("%d",&n);
    for(i=0;i<n;i++)
        count[i]=0;
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    }
    temp=a[0];
    for(i=0;i<n;i++)//主要方法区
    {
        if(temp!=a[i])
            temp=a[i];
        for(j=i+1;j<n;j++)
             if(temp==a[j])
                 count[i]++;
        }
    max=-32767;
```

```
for(i=0;i<n;i++)
{
        if(max<count[i]){
            max=count[i];
        }
     }
    for(i=0;i<n;i++)
     {
        if(max=count[i])
            printf("%d\n",a[i]);
     }
    return 0;
}</pre>
```

九十一, 捉鬼大师

/*

世界上有鬼吗?如果有,那就一定有捉鬼大师,但是捉鬼大师的能力有限,他们只能消灭位于他们东南面的鬼,也就是说,如果把捉鬼大师的位置视为原点,那么他们只能消灭第四象限的鬼怪(位于 X 轴的正半轴, Y 的负半轴)。

现在给出一些鬼的坐标和一些捉鬼大师的坐标,请问有多少鬼不能消灭?

1.输入:

第一行:给出两个数字 g,b 对应于鬼的个数和捉鬼大师的个数。接下来的 g 行,每行两个整数,是每个鬼的坐标(之间用空格隔开),在接下来的 b 行,每行两个整数,是每个捉鬼大师的坐标(之间用空格个隔开),不会有重复的鬼和捉鬼大师出现。

2.输出:

无法被消灭的鬼的坐标 x,y 之间用一个空格隔开,注意这些坐标输出顺序必须按照 y 值的降序排列,当 y 相等时,则按照 x 的降序排列

/ / 范例 输入:

5 2

4 2

2 2

11

1 -2

0 -1

.. .

输出:

11

22

42

```
*/
#include"stdio.h"
#include"stdlib.h"
int g,b;
void result(int m[],int n[],int a[],int c[])
{
    int i,j;
    for(i=0;i<b;i++){//将为被捉鬼大师干掉的鬼保留
         for(j=0;j< g;j++)
         {
              if((m[j]-a[i]>0\&&n[j]-c[i]<0)\&\&j<g-1)
              {
                   m[j]=m[j+1];
                   n[j]=n[j+1];
                  g--;
              }
              if((m[j]-a[i]>0\&&n[j]-c[i]<0)\&\&(j==g-1))
                  g--;
         }
    }
         for(i=0;i<g-1;i++)//排序
         {
              for(j=0;j< g-1-i;j++)
                   if(n[j]>n[j+1])
                   {
                        int temp=n[j];
                        n[j]=n[j+1];
                        n[j+1]=temp;
                        int temp1=m[j];
                        m[j]=m[j+1];
                        m[j+1]=temp1;
                   if(n[j]==n[j+1]\&\&m[j]>m[j+1])
                   {
                        int temp=n[j];
                        n[j]=n[j+1];
                        n[j+1]=temp;
                        int temp1=m[j];
                        m[j]=m[j+1];
                        m[j+1]=temp1;
                  }
              }
         }
```

```
for(i=0;i<g;i++)//输出
         {
             printf("%d %d\n",m[i],n[i]);
         }
}
int main(void)
{
    int i,*m,*n,*a,*c;//m,n 是鬼,a,c 是大师
    scanf("%d%d",&g,&b);
    m=(int *)malloc(g*sizeof(int));//动态数组
    n=(int *)malloc(g*sizeof(int));
    a=(int *)malloc(b*sizeof(int));
    c=(int *)malloc(b*sizeof(int));
    for(i=0;i<g;i++)
    {
         scanf("%d%d",&m[i],&n[i]);
    }
    for(i=0;i<g;i++)//判断鬼是否重复
    {
         if(m[i]==m[i+1]\&\&n[i]==n[i+1]){
             printf("鬼重复\n");
             exit(0);
         if(i+1==g-1)
             break;
    }
    for(i=0;i<b;i++)
    {
         scanf("%d%d",&a[i],&c[i]);
    for(i=0;i<b;i++)//判断捉鬼大师是否重复
    {
         if((a[i]==a[i+1])&&(c[i]==c[i+1])){
             printf("捉鬼大师重复\n");
             exit(0);
         }
         if(i+1==b-1)
             break;
    }
    result(m,n,a,c);
    return 0;
}
```