07.7.23—07.10.9 程序资料整理

算法名称

题目名字

题目描述

在这里输入题目的内容，包括描述，输入输出

算法描述

对这个题目的算法描述，最好写详细点

代码

就是代码啦。如果可以尽量写好注释

算法名称：模拟

假币问题

题目描述

赛利有 12 枚银币。其中有 11 枚真币和 1 枚假币。假币看起来和真币没有区别，但是重量不同。但赛利不知道假币比真币轻还是重。于是他向朋友借了一架天平。朋友希望赛利称三次就能找出假币并且确定假币是轻是重。例如:如果赛利用天平称两枚硬币，发现天平平衡，说明两枚都是真的。如果赛利用一枚真币与另一枚银币比较，发现它比真币轻或重，说明它是假币。经过精心安排每次的称量，赛利保证在称三次后确定假币。

Input

输入有三行，每行表示一次称量的结果。赛利事先将银币标号为 A-L。每次称量的结果用三个以空格隔开的字符串表示：天平左边放置的硬币 天平右边放置的硬币 平衡状态。其中平衡状态用``up", ``down", 或 ``even"表示, 分别为右端高、右端低和平衡。天平左右的硬币数总是相等的。

Output

输出哪一个标号的银币是假币，并说明它比真币轻还是重(heavy or light)。

Sample Input

1

ABCD EFGH even

ABCI EFJK up
ABIJ EFGH even
Sample Output
K is the counterfeit coin and it is light.

算法描述

代码

```
#include<stdio.h>
int main ()
{
	char l[3][5], r[3][5], judge[3][5], x;
	int i, j, y, n, m, t, w, d;
	scanf ( "%d", &d );//测试数据的组数
	while(d--)
	{
		for ( i=0; i<3; i++ )
		{
			scanf ( "%s %s %s", l[i], r[i], judge[i] );
		}
		for ( x='A';x<='L';x++ )
		{
			n=0;m=0;w=1;//3 个标记符
			for ( y=0; y<2; y++ )
			{//0 表示轻，1 表示重
				for ( i=0; i<3; i++ )
				{
					t=0;
					for ( j=0; j<4; j++ )
					{
						if ( l[i][j] == x )
						{
							t=1;//表示在左边
						}
						if ( r[i][j] == x )
						{
							t=2;//表示在右边
						}
					}
					//假设 A 是轻币
					if ( y == 0)
```

```
{
    if ( t == 1 )
        if ( judge[i][0] == 'e' || judge[i][0] == 'u' )
            break;
        else
        {
            n++;//记正确的数据组数
            continue;
        }
        else if ( t == 2 )
            if ( judge[i][0] == 'e' || judge[i][0] == 'd' )
                break;
            else
            {
                n++;
                continue;
            }
            else if ( t == 0 )
            {
                n++;
                continue;
            }
}
//假设 A 是重币
else
{
    if ( t == 1 )
        if ( judge[i][0] == 'e' || judge[i][0] == 'd' )
        {
            break;
        }
        else
        {
            m++;//记正确的数据组数
            continue;
        }
        else if ( t == 2 )
        {
            if ( judge[i][0] == 'e' || judge[i][0] == 'u' )
                break;
            else
            {
                m++;
                continue;
```

```
                                        }
                                    }
                                    else if ( t == 0 )
                                    {
                                        m++;
                                        continue;
                                    }
                                }
                            }
                        }
                        if ( n == 3 )
                        {
                            printf ( "%c is the counterfeit coin and it is light.\n", x );
                            break;
                        }
                        if ( m == 3 )
                        {
                            printf ( "%c is the counterfeit coin and it is heavy.\n", x );
                            break;
                        }
                    }
                }
                return 0;
}
```

跳绳问题

题目描述

小朋友玩跳绳比赛,要计算在一分钟内跳了多少下.假设每秒钟跳一下,如果中途失败了,则要花三秒钟后才能开始重跳.一般小朋友跳绳一分钟要跳坏好几次.现在给出小朋友每次跳坏时已经跳的总数,求小朋友在一分钟内跳了多少下.(请注意分析示例数据.)

Input

第一行为 n 个小朋友

其余各行,每行第一个整数是跳坏的次数 m,其余 m 个整数是跳坏时累计跳了多少下.

Output

输出相应小朋友头一分钟内跳了多少下.

Sample Input

6

0

3 12 23 45

1 17
4 10 20 30 40
5 10 20 30 40 58
6 10 20 30 40 47 60
Sample Output
60
51
57
48
48
47
Hint
提示,在跳绳比赛时,你可能已经超时了,但自己还在计数,但裁判已经停止计时并得到成绩了.这里相当与自己计数.因此,并非跳坏的时候都是在前一分钟以内.请注意分析示例数据.

算法描述

代码：

```
#include<stdio.h>
#define N 100
void main ()
{
    int k, n, t, s, i, j, a[N];
    scanf ( "%d", &n );
    for ( i=0; i<n; i++ )
    {
        t = 0;                  //记住要给 t 归 0
        scanf ( "%d", &k );
        if ( k == 0 )
        {
            t = 60;
            printf ( "%d\n",    t);
            continue;
        }
        for ( j=0; j<k; j++ )
            scanf ( "%d", &a[j] );
        for ( j=0; j<k && t<=60; j++ )
            t = a[j] + 3 * ( j + 1 );
        s = a[j-1];
        if ( t < 60 )
            s = s + 60 - t;
```

```
            if ( t > 63 )
                s = s - t + 63;
            printf ( "%d\n", s );
        }
}
```

生日相同

题目描述

在一个有 180 人的大班级中，存在两个人生日相同的概率非常大，现给出每个学生的学号，出生月日。试找出所有生日相同的学生。

Input

第一行为整数 n，表示有 n 个学生，n<100。
此后每行包含一个字符串和两个整数，分别表示学生的学号（字符串长度小于 10）和出生月(1<=m<=12)日(1<=d<=31)。
学号、月、日之间用一个空格分隔。

Output

对每组生日相同的学生，输出一行，
其中前两个数字表示月和日，后面跟着所有在当天出生的学生的学号，数字、学号之间都用一个空格分隔。
对所有的输出，要求按日期从前到后的顺序输出。
对生日相同的学号，按输入的顺序输出。

Sample Input

5
00508192 3 2
00508153 4 5
00508172 3 2
00508023 4 5
00509122 4 5

Sample Output

3 2 00508192 00508172
4 5 00508153 00508023 00509122

算法描述

代码

```
#include<stdio.h>
#define N 100
int main ()
{
    int a[N][2], n, m=0, h=0, i, j, k;
    char s[N][12];
    scanf ( "%d", &n );
    for ( i=0; i<n; i++ )
    {
        scanf ( "%s %d %d", s[i], &a[i][0],    &a[i][1] );
    }
    for ( i=1; i<=12; i++ )
    {
        m=0;
        for ( j=0; j<n; j++ )
            if ( a[j][0] == i )
                m++;
        if ( m>1 )
        {
            for ( k=1; k<32; k++ )
            {
                h=0;
                for ( j=0; j<n; j++ )
                    if ( a[j][0] == i && a[j][1] == k )//要保证 k 在同一月份 i 下
                        h++;
                if ( h>1 )
                {
                    printf ( "%d %d ", i, k );
                    for ( j=0; j<n; j++ )
                        if ( a[j][0] == i && a[j][1] == k )
                            printf ( "%s ", s[j] );
                    printf ( "\n" );
                }
            }
        }
    }
    return 0;
}
```

约瑟夫问题

题目描述

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

Input

每行是用空格分开的两个整数，第一个是 n, 第二个是 m (0 < m,n < 300)。最后一行是：

0 0

Output

对于每行输入数据（最后一行除外)，输出数据也是一行，即最后猴王的编号

Sample Input

6 2
12 4
8 3
0 0

Sample Output

5
1
7

算法描述：

模拟

代码

```
#include<stdio.h>
#define N 300
int main ()
{
    int a[N], j, h, k=1, m, n;
    while(scanf ( "%d %d", &n, &m ) != EOF)
    {
```

```
        if ( n==0 && m==0)
            break;
        for ( j=1; j<=n; j++)
            a[j]=1;
        h=n;
        j=1;
        k=1;
        while (h-1)
        {
            if ( a[j] == 0)//遇到标记为 0 的就跳过
            {
                j++;
                if ( j==n+1 )
                    j=1;
                continue;
            }
            if ( k==m )
            {
                a[j] = 0;
                h--;
                k=0;
            }
            k++;
            j++;
            if ( j==n+1 )
                j=1;
        }
        for ( j=1; j<=n; j++ )
            if ( a[j]!=0 )
            {
                printf( "%d\n", j );
                break;
            }
    }
    return 0;
}
```

算法描述：

数学方法

代码

```
#include <stdio.h>
int main()
{
	int n, m, s, i;
	while ( scanf( "%d %d", &n, &m ) != EOF )
	{
		if ( n==0 && m==0 )//如果输入 0 0 就跳出
			break;
		for ( s=0, i=2; i<=n; i++ )
			s = (s+m) % i;
		printf ( "%d\n", s+1 );
	}
	return 0;
}
```

Jugs

题目描述：

In the movie "Die Hard 3", Bruce Willis and Samuel L. Jackson were confronted with the following puzzle. They were given a 3-gallon jug and a 5-gallon jug and were asked to fill the 5-gallon jug with exactly 4 gallons. This problem generalizes that puzzle.

You have two jugs, A and B, and an infinite supply of water. There are three types of actions that you can use: (1) you can fill a jug, (2) you can empty a jug, and (3) you can pour from one jug to the other. Pouring from one jug to the other stops when the first jug is empty or the second jug is full, whichever comes first. For example, if A has 5 gallons and B has 6 gallons and a capacity of 8, then pouring from A to B leaves B full and 3 gallons in A.

A problem is given by a triple (Ca,Cb,N), where Ca and Cb are the capacities of the jugs A and B, respectively, and N is the goal. A solution is a sequence of steps that leaves exactly N gallons in jug B. The possible steps are

fill A
fill B
empty A
empty B
pour A B
pour B A

success

where "pour A B" means "pour the contents of jug A into jug B", and "success" means that the goal has been accomplished.

You may assume that the input you are given does have a solution.

Input

Input to your program consists of a series of input lines each defining one puzzle. Input for each puzzle is a single line of three positive integers: Ca, Cb, and N. Ca and Cb are the capacities of jugs A and B, and N is the goal. You can assume 0 < Ca <= Cb and N <= Cb <=1000 and that A and B are relatively prime to one another.

Output

Output from your program will consist of a series of instructions from the list of the potential output lines which will result in either of the jugs containing exactly N gallons of water. The last line of output for each puzzle should be the line "success". Output lines start in column 1 and there should be no empty lines nor any trailing spaces.

Sample Input

3 5 4
5 7 3

Sample Output

fill B
pour B A
empty A
pour B A
fill B
pour B A
success
fill A
pour A B
fill A
pour A B
empty B
pour A B
success

算法描述

模拟

代码

```
#include<stdio.h>
#define N 100
int main ()
{
	int ca, cb, va, vb, n;
	while ( scanf ( "%d%d%d", &ca, &cb, &n) != EOF )
	{
		if ( cb == n )
		{
			printf( "fill B\n");
			printf( "success\n");
			continue;
		}
		if ( ca == n )
		{
			printf( "fill A\n");
			printf( "pour A B\n");
			printf( "success\n");
			continue;
		}
		va = 0;
		vb = 0;
		do
		{
			printf( "fill A\n");
			va += ca;
			printf( "pour A B\n");
			vb += va;
			va = 0;
			if ( vb > cb)
			{
				va += vb - cb;
				vb = cb;
				printf( "empty B\n");
				vb = 0;
				printf ( "pour A B\n");
				vb += va;
				va = 0;
			}
		}while ( vb != n );
		printf( "success\n");
	}
```

```
    return 0;
}
```

算法名称：数学

判断闰年

代码：

```
#include<stdio.h>
int main ()
{
    int a;
    while ( scanf ( "%d", &a ) != EOF )
    {
        if ( a % 4 == 0 )
        {
            if ( a % 100 == 0    )
            {
                if ( a % 3200 == 0 )
                    printf ( "N\n");
                else if ( a % 400 != 0 )
                    printf ( "N\n" );
                else
                printf ( "Y\n" );
            }
            else
                printf ( "Y\n" );
        }
        else
            printf ( "N\n" );
    }
    return 0;
}
```

最长上升子序列

题目描述

一个数的序列 b_i，当 $b_1 < b_2 < ... < b_S$的时候，我们称这个序列是上升的。对于给定的一个序列($a_1, a_2, ..., a_N$)，我们可以得到一些上升的子序列($a_{i1}, a_{i2}, ..., a_{iK}$)，这里 1 <= $i_1 < i_2 < ... < i_K$ <= N。比如，对于序列(1, 7, 3, 5, 9, 4, 8)，有它的一些上升子序列，如(1, 7), (3, 4, 8)等等。这些子序列中最长的长度是 4，比如子序列(1, 3, 5, 8).
你的任务，就是对于给定的序列，求出最长上升子序列的长度。

Input

输入的第一行是序列的长度 N (1 <= N <= 1000)。第二行给出序列中的 N 个整数，这些整数的取值范围都在 0 到 10000。

Output

最长上升子序列的长度。

Sample Input

7
1 7 3 5 9 4 8

Sample Output

4

算法描述

代码

```
#include<stdio.h>
#define N 1000
int main()
{
    int a[N], i, j, n, f[N], len=1;
    scanf ( "%d", &n );
    for ( i=0; i<n; i++ )
        scanf ( "%d", &a[i] );
    f[0] =1;
    for ( i=1; i<n; i++ )//前 i 个数最大上升序列长度为前 j 个数中最大上升序列长度加 1
    {
        f[i] = 1;
        for ( j=0; j<i; j++ )
```

```
            {
                if ( a[i] > a[j] && f[i] < f[j]+1 )
                    f[i] = f[j]+1;
                if ( f[i] > len )
                    len = f[i];
            }
        }
        printf ( "%d", len );
        return 0;
}
```

Warcraft III 守望者的烦恼

题目描述：

守望者-warden，长期在暗夜精灵的的首都艾萨琳内担任视察监狱的任务，监狱是成长条行的，守望者 warden 拥有一个技能名叫“闪烁”，这个技能可以把她传送到后面的监狱内查看，她比较懒，一般不查看完所有的监狱，只是从入口进入，然后再从出口出来就算完成任务了。
描述 Description
头脑并不发达的 warden 最近在思考一个问题，她的闪烁技能是可以升级的，k 级的闪烁技能最多可以向前移动 k 个监狱，一共有 n 个监狱要视察，她从入口进去，一路上有 n 个监狱，而且不会往回走，当然她并不用每个监狱都视察，但是她最后一定要到第 n 个监狱里去，因为监狱的出口在那里，但是她并不一定要到第 1 个监狱。
守望者 warden 现在想知道，她在拥有 k 级闪烁技能时视察 n 个监狱一共有多少种方案？

Input

第一行是闪烁技能的等级 k(1<=k<=10)
第二行是监狱的个数 n(1<=n<=2^31-1)

Output

由于方案个数会很多，所以输出它 mod 7777777 后的结果就行了

Sample Input

2
4

Sample Output

5

Hint

把监狱编号 1 2 3 4,闪烁技能为 2 级，
一共有 5 种方案

→1→2→3→4
→2→3→4
→2→4
→1→3→4
→1→2→4

小提示：建议用 int64，否则可能会溢出

算法描述

N 阶矩阵相乘：
递推关系 An=x1A(n-1)+x2(An-2)+...+xk(An-k)(其中 xi 为常数)叫做 k 阶常系数线性递推关系,求满足此递推关系的数列的第 n 项的最正点的方法是矩阵乘法(简称矩乘),
其复杂度为 O(k^3*logn).当然,这类数列通常增长都比较快,因此往往求的是第 n 项除以某个数的余数.
先来讲一下矩阵乘法及矩阵幂的快速计算.两个矩阵可以相乘,当且仅当第一个矩阵的列数等于第二个矩阵的行数.
由此亦可看出,矩乘没有交换律.现有两个矩阵 A 和 B,A 的大小为 m*n,B 的大小为 n*p,则 A*B 的结果 C 是一个 m*p 的矩阵.
它的每个元素满足.矩乘的编程实现是十分简单的.计算矩阵幂,当然可以一次一次地乘.
但是,由于矩乘满足结合律(A*B)*C=A*(B*C),故可以用二分法计算矩阵幂.
例如计算 An 时,若 n 为偶数,则 An=An/2*An/2;若 n 为奇数,则 An=A(n-1)/2*A(n-1)/2*A.
那么矩乘与矩阵幂在求满足 k 阶常系数线性递推关系的数列通项时有什么用呢?
我们想办法构造一个矩阵 A,使得[a1 a2 ... ak]*A=[a2 a3 ... ak+1].容易得出矩阵 A 有如下特征:
它的大小为 k*k;
它的第 k 列为 x1 至 k,即递推关系中的系数;
第 i(1<=i<k)列中除第 i+1 行元素为 1 外,其余元素均为 0.
这样,每乘一次 A,就可以多求出数列的一项,乘以 A 的 n-k 次幂,结果矩阵的最右一个元素就是要求的 an 了

代码

```
#include <stdio.h>
#include <string.h>
void bs(__int64 a[10][10],__int64 b[10][10],long k)
{
	long i,j,h;
	__int64 c[10][10]={0};
	for(i=0;i<k;i++)
	{
		for(j=0;j<k;j++)
		{
```

```
                for(h=0;h<k;h++)
                {
                    c[i][j]=(c[i][j]+(a[i][h]*b[h][j])%7777777)%7777777;
                }
            }
        }
        for(i=0;i<k;i++)
        {
            for(j=0;j<k;j++)
                a[i][j]=c[i][j];
        }
}
int main()
{
    long i,n,k,j;
    __int64 a[10][10],s[10][10];
    scanf("%ld %ld",&k,&n);
    if(k==1)
        printf("1\n");
    else
    {
        memset(a,0,sizeof(a));
        memset(s,0,sizeof(s));
        for(i=0;i<k;i++)
        {
            a[i][0]=1;
            s[i][i]=1;
            if(i+1<k)
                a[i][i+1]=1;
        }
        while(n)
        {
            if(n%2)
                bs(s,a,k);
            bs(a,a,k);
            n/=2;
        }
        printf("%I64d\n",s[0][0]%7777777);
    }
    return 0;
}
```

出栈序列统计

题目描述：

栈是常用的一种数据结构，有 n 令元素在栈顶端一侧等待进栈，栈顶端另一侧是出栈序列。你已经知道栈的操作有两·种：push 和 pop，前者是将一个元素进栈，后者是将栈顶元素弹出。现在要使用这两种操作，由一个操作序列可以得到一系列的输出序列。请你编程求出对于给定的 n，计算并输出由操作数序列 1，2，…，n，经过一系列操作可能得到的输出序列总数。

输入格式 Input Format

一个整数 n（1<=n<=15）

输入格式 Input Format

一个整数 n（1<=n<=15）

样例输入 Sample Input

3

样例输出 Sample Output

5

算法描述

Catalan 数：

原理：

令 h(0)＝1，catalan 数满足递归式：

h(n)= h(0)*h(n-1) + h(1)*h(n-2) + ... + h(n-1)h(0) (其中 n>=1)

Catalan 数最典型的三类应用：（实质上却都一样，无非是递归等式的应用，就看你能不能分解问题写出递归式了）

1.括号化问题。

矩阵链乘： P=a1×a2×a3×……×an，依据乘法结合律，不改变其顺序，只用括号表示成对的乘积，试问有几种括号化的方案？(h(n)种)

2.出栈次序问题。

一个栈(无穷大)的进栈序列为 1,2,3,..n,有多少个不同的出栈序列?

类似：有 2n 个人排成一行进入剧场。入场费 5 元。其中只有 n 个人有一张 5 元钞票，另外n 人只有 10 元钞票，剧院无其它钞票，问有多少中方法使得只要有 10 元的人买票，售票处就有 5 元的钞票找零？(将持 5 元者到达视作将 5 元入栈，持 10 元者到达视作使栈中某 5 元出栈)

3.将多边行划分为三角形问题。

将一个凸多边形区域分成三角形区域的方法数?

类似：一位大城市的律师在她住所以北 n 个街区和以东 n 个街区处工作。每天她走 2n 个街区去上班。如果他

从不穿越（但可以碰到）从家到办公室的对角线，那么有多少条可能的道路？

类似：在圆上选择 2n 个点,将这些点成对连接起来使得所得到的 n 条线段不相交的方法数?

代码

```
#include<stdio.h>
int main ()
{
	long int n, i, j, c[20];
	scanf("%ld", &n);
	c[0]=1;
	for(i=1; i<=n; i++)
	{
		for(j=0, c[i]=0; j<i; j++)
		{
			c[i]+=c[j]*c[i-1-j];
		}
	}
	printf("%ld\n", c[n]);
	return 0;
}
```

算法名称：贪心

金银岛

题目描述

某天 KID 利用飞行器飞到了一个金银岛上，上面有许多珍贵的金属，KID 虽然更喜欢各种宝石的艺术品，可是也不拒绝这样珍贵的金属。但是他只带着一个口袋，口袋至多只能装重量为 w 的物品。岛上金属有 s 个种类, 每种金属重量不同，分别为 $n_1, n_2, ... , n_s$，同时每个种类的金属总的价值也不同，分别为 $v_1,v_2, ..., v_s$。KID 想一次带走价值尽可能多的金属，问他最多能带走价值多少的金属。注意到金属是可以被任意分割的，并且金属的价值和其重量成正比。

Input

第 1 行是测试数据的组数 k，后面跟着 k 组输入。

每组测试数据占 3 行，第 1 行是一个正整数 w (1 <= w <= 10000)，表示口袋承重上限。第 2 行是一个正整数 s (1 <= s <=100)，表示金属种类。第 3 行有 2s 个正整数，分别为 $n_1, v_1, n_2, v_2, ... , n_s, v_s$ 分别为第一种，第二种，...，第 s 种金属的总重量和总价值(1 <= n_i <= 10000, 1 <= v_i <= 10000)。

Output

k 行，每行输出对应一个输入。输出应精确到小数点后 2 位。

Sample Input

```
2
50
4
10 100 50 30 7 34 87 100
10000
5
1 43 43 323 35 45 43 54 87 43
```

Sample Output

```
171.93
508.00
```

算法描述：

贪心+快排

代码

```
#include<stdio.h>
#define N 200
void qsort(double *r ,double *w ,double *v, int start, int end)
{
	int i, j;
	double temp1, temp2, temp3;
	if(start<end)
	{
		i=start,j=end;
		temp1=r[i];temp2=w[i];temp3=v[i];
		while(i!=j)
		{
			while(j>i&&r[j]<=temp1) j--;
			if(i<j)
			{
				r[i]=r[j];
				w[i]=w[j];
				v[i]=v[j];
				i++;
			}
			while(i<j&&r[i]>=temp1) i++;
```

```
            if(i<j)
            {
                r[j]=r[i];
                w[j]=w[i];
                v[j]=v[i];
                j--;
            }
        }
        r[i]=temp1;w[i]=temp2;v[i]=temp3;
        qsort(r,w,v,start,i-1);
        qsort(r,w,v,i+1,end);
    }
}

int main()
{
    int i, n, k, s, h;
    double weight, value, p[N], w[N], v[N], r[N];
    scanf("%d",&n);
    while(n--)
    {
        scanf("%lf", &weight);
        scanf("%d", &s);
        for(i=0; i<2*s ;i++)
            scanf("%lf", &p[i]);
        k=0;h=0;
        for(i=0; i<2*s; i++)
        {
            if(i%2)
                v[k++]=p[i];
            else
                w[h++]=p[i];
        }
        for(i=0; i<s; i++)
        {
            r[i]=v[i]/w[i];
        }
        qsort(r,w,v,0,s-1);
        value=0;
        for(i=0;i<s;i++)
        {
            if(weight>=w[i])
            {
                weight=weight-w[i];
```

```
                value+=v[i];
            }
            else
                break;
        }
        if(i<s)
        {
            value+=weight*v[i]/w[i];
        }
        printf("%.2lf\n", value);
    }
    return 0;
}
```

木棍加工////??????

题目描述：

一堆木头棍子共有 n 根，每根棍子的长度和宽度都是已知的。棍子可以被一台机器一个接一个地加工。机器处理一根棍子之前需要准备时间。准备时间是这样定义的：
第一根棍子的准备时间为 1 分钟；
如果刚处理完长度为 L，宽度为 W 的棍子，那么如果下一个棍子长度为 Li，宽度为 Wi，并且满足 L>＝Li，W>＝Wi，这个棍子就不需要准备时间，否则需要 1 分钟的准备时间；
计算处理完 n 根棍子所需要的最短准备时间。比如，你有 5 根棍子，长度和宽度分别为(4, 9)，(5, 2)，(2, 1)，(3, 5)，(1, 4)，最短准备时间为 2（按(4, 9)、(3, 5)、(1, 4)、(5, 2)、(2, 1)的次序进行加工）。

Input

第一行是一个整数 n(n<＝5000)，第 2 行是 2n 个整数，分别是 L1，W1，L2，w2，…，Ln，Wn。L 和 W 的值均不超过 10000，相邻两数之间用空格分开。

Output

仅一行，一个整数，所需要的最短准备时间。

Sample Input

5
4 9 5 2 2 1 3 5 1 4

Sample Output

2

算法描述

贪心

代码

```
#include<stdio.h>
void qsort ( int *l, int *w, int start, int end)
{
	int i, j,temp1, temp2;
	if (start<end)
	{
		i=start;
		j=end;
		temp1=l[i];
		temp2=w[i];
		while (i!=j)
		{
			while ( i<j && l[j] >=temp1) j--;
			if ( i<j )
			{
				l[i]=l[j];
				w[i]=w[j];
				i++;
			}
			while ( i<j && l[i]<=temp1) i++;
			if ( i<j)
			{
				l[j]=l[i];
				w[j]=w[i];
				j--;
			}
		}
		l[i]=temp1;
		w[i]=temp2;
		qsort ( l, w, start, i-1);
		qsort ( l, w, i+1, end);
	}
}
int main ()
{
	int n,l[5000],w[5000], i, j, k, s, cnt, flag[5000];
	scanf ("%d", &n);
```

```
    for ( i=0; i<n; i++)
    {
        scanf ("%d%d", &l[i], &w[i]);
        flag[i]=0;
    }
    qsort( l, w, 0, n-1);
     for ( i=0; i<n; i++)//对 l[i]的相等的项数，再对 w[i]进行排序。
    {
        s=i;
        while ( i<n-1 && l[i]==l[i+1] ) i++;
        qsort(w, l, s, i);
    }
    for ( i=0, cnt=0;i<n;i++)//????????
    {
        if (flag[i]==0)
        {
            cnt++;
            flag[i]=1;
            k=i;
            for ( j=k+1; j<n; j++)
            {
                if ( flag[j]==0 && w[j]>=w[k] )
                {
                    flag[j]=1;
                    k=j;
                }
            }
        }
    }
    printf("%d\n", cnt);
    return 0;
}
```

算法名称：搜索

Sticks PKU_1011

题目描述：

George took sticks of the same length and cut them randomly until all parts became at most 50 units long. Now he wants to return sticks to the original state, but he forgot how many sticks he had originally and how long they were originally. Please help him and design a program which computes the smallest possible original length of those sticks. All lengths expressed in units are integers greater than zero.

Input

The input contains blocks of 2 lines. The first line contains the number of sticks parts after cutting, there are at most 64 sticks. The second line contains the lengths of those parts separated by the space. The last line of the file contains zero.

Output

The output should contains the smallest possible length of original sticks, one per line.

Sample Input

```
9
5 2 1 5 2 1 5 2 1
4
1 2 3 4
0
```

Sample Output

```
6
5
```

算法描述

两个函数相互嵌套，深度搜索

代码

```
#include<stdio.h>
#define N 100
int sticks[N], used[N];
int len, sum, num, n, ok;
void qsort(int *l,int start,int end)
{
	int i,j,temp1;
	if(start<end)
	{
		i=start,j=end;
		temp1=l[i];
		while(i!=j)
		{
			while(j>i&&l[j]<=temp1) j--;
			if(i<j)
			{
				l[i]=l[j];
```

```
                i++;
            }
            while(i<j&&l[i]>=temp1) i++;
            if(i<j)
            {
                l[j]=l[i];
                j--;
            }
        }
        l[i]=temp1;
        qsort(l,start,i-1);
        qsort(l,i+1,end);
    }
}
void reduction( int k);
void dfs ( int k, int now, int next )
{
    int i;
    if (ok) return;
    if ( now == len )
    {
        reduction ( k + 1 );
        return;
    }
    if ( next == n ) return ;
    for ( i = next + 1; i < n; i++)
    {
        if ( !used[i] && now + sticks[i] <=len)
        {
            used[i] = 1;
            dfs ( k, now+sticks[i], i);
            used[i]=0;
            if (ok) return;
            if (sticks[i] == len - now) return;
        }
    }
}
void reduction ( int k )//还原第 k 根木棍
{
    int i;
    if (ok) return;
    if ( k > num )
    {
        ok=1;
```

```
            printf("%d\n", len);
            return ;
        }
        for ( i=0; i<n; i++)//找到第一根没用过的木棍
        {
            if ( !used[i] )
                break;
        }
        used[i]=1;
        dfs( k, sticks[i], i );
        used[i]=0;
}
int main ()
{
        int i;
        while ( scanf ("%d", &n ) !=    EOF)
        {
            if ( n == 0 )
                break;
            for ( i = 0, sum = 0; i < n; i++)
            {
                scanf ("%d", &sticks[i]);
                sum += sticks[i];
            }
            qsort(sticks,0, n-1);//从大到小的排序
            ok=0;
            for ( len = sticks[0]; len <= sum; len++)
            {
                if ( sum % len == 0 && !ok )
                {
                    for ( i=0; i<n; i++)
                    used[i]=0;
                    num = sum / len;
                    reduction(1);
                }
            }
        }
        return 0;
}
```

Frame Stacking PKU_1128

题目描述：

Consider the following 5 picture frames placed on an 9 x 8 array.

```
........ ........ ........ ........ .CCC....

EEEEEE.. ........ ........ ..BBBB.. .C.C....

E....E.. DDDDDD.. ........ ..B..B.. .C.C....

E....E.. D....D.. ........ ..B..B.. .CCC....

E....E.. D....D.. ....AAAA ..B..B.. ........

E....E.. D....D.. ....A..A ..BBBB.. ........

E....E.. DDDDDD.. ....A..A ........ ........

E....E.. ........ ....AAAA ........ ........

EEEEEE.. ........ ........ ........ ........

    1        2        3        4        5
```

Now place them on top of one another starting with 1 at the bottom and ending up with 5 on top. If any part of a frame covers another it hides that part of the frame below.

Viewing the stack of 5 frames we see the following.

```
.CCC....

ECBCBB..

DCBCDB..

DCCC.B..

D.B.ABAA

D.BBBB.A
```

DDDDAD.A

E...AAAA

EEEEEE..

In what order are the frames stacked from bottom to top? The answer is EDABC.

Your problem is to determine the order in which the frames are stacked from bottom to top given a picture of the stacked frames. Here are the rules:

1. The width of the frame is always exactly 1 character and the sides are never shorter than 3 characters.

2. It is possible to see at least one part of each of the four sides of a frame. A corner shows two sides.

3. The frames will be lettered with capital letters, and no two frames will be assigned the same letter.

Input

Each input block contains the height, h (h<=30) on the first line and the width w (w<=30) on the second. A picture of the stacked frames is then given as h strings with w characters each.
Your input may contain multiple blocks of the format described above, without any blank lines in between. All blocks in the input must be processed sequentially.

Output

Write the solution to the standard output. Give the letters of the frames in the order they were stacked from bottom to top. If there are multiple possibilities for an ordering, list all such possibilities in alphabetical order, each one on a separate line. There will always be at least one legal ordering for each input block. List the output for all blocks in the input sequentially, without any blank lines (not even between blocks).

Sample Input

9
8
.CCC....
ECBCBB..
DCBCDB..

```
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
```

Sample Output

```
EDABC
```

算法描述

拓扑排序+深度搜索

代码

```
#include<stdio.h>
#include<string.h>
#define N 100
int    h, w, num, m, a[N][N], p[N], flag[N];
int minx, miny, maxx, maxy;
char map[N][N], q[N][N];
void sort ( int t )//拓扑排序
{
    int i, j;
    if ( t == num )
    {
        for ( i=t-1,j=0; i>=0; i-- )
        {
            q[m][j++]=p[i]+'A'-1;
        }
        q[m][j]='\0';
        m++;
        return ;
    }
    for ( i=26; i>=1; i-- )//dfs
    {
        if ( a[i][0]==0 && flag[i]==0 )
        {
            p[t]=i;
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
char map[50][50];
int adj[30][30];
int path[30];
int flag[30];
struct str
{
	char a[30];
}ans[6000];
int ansnum;
int m, n, s;
int num;
int cmp(const void *c, const void *d)
{
	return strcmp((*(struct str *)c).a, (*(struct str *)d).a);
}
void BuildAdj()
{
	int i, j, k, tag;
	int maxr, maxc, minr, minc;
	memset(adj, -1, sizeof(adj));
	num = 0;
	for(k = 'A';k <= 'Z';k++)
	{
		adj[k - 'A' + 1][0] = 0;
		tag = 0;
		maxr = -1;
		maxc = -1;
		minr = 100;
		minc = 100;
		for(i = 1;i <= m;i++)
		{
			for(j = 1;j <= n;j++)
			{
				if(map[i][j] == k)
				{
					tag = 1;
					if(i > maxr) maxr = i;
					if(j > maxc) maxc = j;
					if(i < minr) minr = i;
					if(j < minc) minc = j;
				}
			}
		}
```

```
            if(tag == 0)
            {
                adj[k - 'A' + 1][0] = -1;
                continue;
            }
            num++;
            for(i = minr;i <= maxr;i++)
            {
                for(j = minc, s = 1;s <= 2;j = maxc, s++)
                {
                    if(map[i][j] != k && adj[k - 'A' + 1][map[i][j] - 'A' + 1] == -1)
                    {
                        adj[k - 'A' + 1][0]++;
                        adj[k - 'A' + 1][map[i][j] - 'A' + 1] = 1;
                    }
                }
            }
            for(j = minc;j <= maxc;j++){
                for(i = minr,s = 1;s <= 2;i = maxr, s++){
                    if(map[i][j] != k && adj[k - 'A' + 1][map[i][j] - 'A' + 1] == -1)
                    {
                        adj[k - 'A' + 1][0]++;
                        adj[k - 'A' + 1][map[i][j] - 'A' + 1] = 1;
                    }
                }
            }
        }
    }



void TopologicalSort(int t)//t 只是个计数器
{
    int i, j;
    if(num == t)
    {
        j = 0;
        for(i = t - 1;i >= 0;i--)
        {
            ans[ansnum].a[j++] =    path[i] + 'A' - 1;
        }
        ans[ansnum].a[j] = 0;
        ansnum++;
        return ;
```

```
        }
        for(i = 26;i >= 1;i--)
        {
            if(adj[i][0] == 0 && flag[i] == 0)
            {
                path[t] = i;
                flag[i] = 1;
                for(j = 1;j <= 26;j++)
                {
                    if(adj[j][i] == 1)
                    {
                        adj[j][0]--;
                        adj[j][i] = 0;
                    }
                }
                TopologicalSort(t + 1);
                flag[i] = 0;
                for(j = 1;j <= 26;j++)
                {
                    if(adj[j][i] == 0)
                    {
                        adj[j][0]++;
                        adj[j][i] = 1;
                    }
                }
            }
        }
}
int main()
{
        int i, j;
        while(scanf("%d%d", &m ,&n) != EOF)
        {
            for(i = 1;i <= m;i++)
            {
                scanf("%s", map[i] + 1);
            }
            BuildAdj();
            for(i = 0;i <= 26;i++) flag[i] = 0;
            ansnum = 0;
            TopologicalSort(0);
            qsort(ans, ansnum , sizeof(ans[0]), cmp);
            for(i = 0;i <= ansnum - 1;i++){
                printf("%s\n", ans[i].a);
```

```
            }
        }
        return 0;
}
                flag[i]=1;
                for ( j=1; j<=26; j++ )
                {
                    if ( a[j][i]== 1 )
                    {
                        a[j][0]--;
                        a[j][i]=0;
                    }
                }
                sort(t+1);
                flag[i] = 0;
                for(j = 1;j <= 26;j++)
                {
                    if(a[j][i] == 0)
                    {
                        a[j][0]++;
                        a[j][i] = 1;
                    }
                }
            }
        }
}
void qsort(char l[][N],int start,int end)
{
        int i,j;
        char temp1[N];
        if(start<end)
        {
            i=start,j=end;
            strcpy(temp1,l[i]);
            while(i!=j)
            {
                while(j>i&&strcmp(l[j],temp1)>=0) j--;
                if(i<j)
                {
                    strcpy(l[i],l[j]);
                    i++;
                }
                while(i<j&&strcmp(l[i],temp1)<=0) i++;
                if(i<j)
```

```
                {
                    strcpy(l[j],l[i]);
                    j--;
                }
            }
            strcpy(l[i],temp1);
            qsort(l,start,i-1);
            qsort(l,i+1,end);
        }
}
int main ()
{
    int i, j, tag;
    int k;
    while ( scanf ( "%d%d", &h, &w ) != EOF )
    {
        getchar();
        for ( i=1; i<=h; i++ )
        {
            gets (&map[i][1]);
        }
        for ( i=0; i<=26; i++ )//初始化数组覆盖情况标记数组 a
        {
            for ( j=0; j<=26; j++ )
            {
                a[i][j]=0;
            }
        }
        for ( k=1, num=0 ; k<=26; k++)//存储覆盖情况
        {
            minx=100;
            maxx=0;
            miny=100;
            maxy=0;
            tag=0;
            for ( i=1; i<=h; i++)
            {
                for ( j=1; j<=w; j++)
                {
                    if ( map[i][j]==k+'A'-1 )
                    {
                        tag = 1;
                        if ( i < miny )     miny=i;
                        if ( i > maxy )     maxy=i;
```

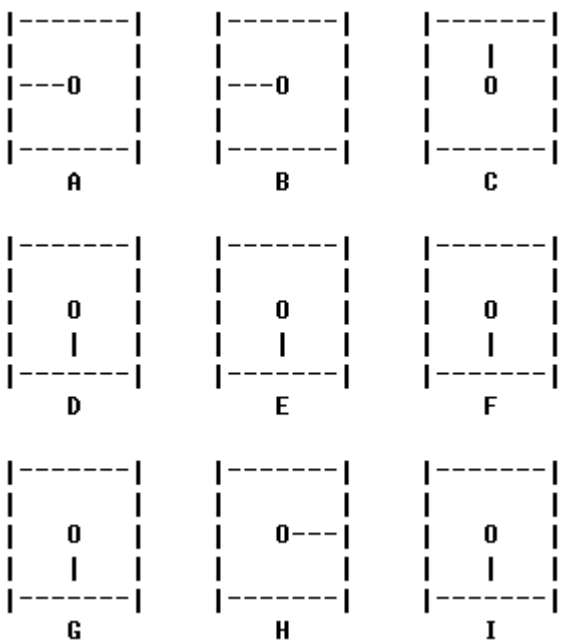
```
                    if ( j < minx )      minx=j;
                    if ( j > maxx )      maxx=j;
                }
        }
}
if ( tag==0 )
{
        a[k][0]=-1;
        continue;
}
num++;
for ( j=minx,i=miny; j<=maxx; j++ )
{
        if ( map[i][j]!=k+'A'-1 && a[k][map[i][j]-'A'+1]==0 )
        {
                a[k][0]++;
                a[k][map[i][j]-'A'+1]=1;
        }
}
for ( j=minx,i=maxy; j<=maxx; j++ )
{
        if ( map[i][j]!=k+'A'-1 && a[k][map[i][j]-'A'+1]==0 )
        {
                a[k][0]++;
                a[k][map[i][j]-'A'+1]=1;
        }
}
for ( i=miny,j=minx; i<=maxy; i++ )
{
        if ( map[i][j]!=k+'A'-1 && a[k][map[i][j]-'A'+1]==0 )
        {
                a[k][0]++;
                a[k][map[i][j]-'A'+1]=1;
        }
}

for ( i=miny,j=maxx; i<=maxy; i++ )
{
        if ( map[i][j]!=k+'A'-1 && a[k][map[i][j]-'A'+1]==0 )
        {
                a[k][0]++;
                a[k][map[i][j]-'A'+1]=1;
        }
}
```

```
            }
            for ( i=1; i<=26; i++ )
            {
                flag[i]=0;
            }
            k=0;
            m=0;
            sort(0);
            if ( m>1 )
            {
                qsort (q, 0, m-1 );
            }
            for ( i=0; i<m; i++ )
            {
                printf("%s\n", q[i]);
            }
        }
        return 0;
}
```

The Clocks PKU_1166

题目描述：



在 2008 北京奥运会雄伟的主会场的墙上，挂着如上图所示的 3*3 的九个挂钟（一开始指针即时针指向的位置请根据输入数据调整）。然而此次奥运会给与了大家一个机会，去用最少的移动操作改变上面的挂钟的时间全部为 12 点正（我们只考虑时针）。然而每一次操作并不是任意的，我们必须按照下面给出的列表对于挂钟进行改变。每一次操作我们给而且必须给指定的操作挂钟进行，每一个挂钟顺时针转动 90 度。列表如下：

操作	指定的操作挂钟
1	ABDE
2	ABC
3	BCEF
4	ADG
5	BDEFH
6	CFI
7	DEGH
8	GHI
9	EFHI

输入格式 Input Format

你的程序按照标准的 3*3 格式读入，一共 9 个 0-3 的数。0 代表 12 点，1 代表 3 点，2 代表 6 点，3 代表 9 点。

Your program is to read from standard input. Nine numbers give the start positions of the dials. 0=12 o'clock, 1=3 o'clock, 2=6 o'clock, 3=9 o'clock.

输出格式 Output Format

你的程序需要写出标准的输出。输出一个最短的能够使所有挂钟指向 12 点的移动操作序列，中间以空格隔开，最后有空格，加回车。这一条最短操作需要是所有最短操作中最小的，也就是说选择最小的第一个操作数，如果第一个操作数相等，那么选择最小的第二个操作数……以此类推。值得肯定的是，这一条操作序列是唯一的。

Sample Input

3 3 0
2 2 2
2 1 2

Sample Output

4 5 8 9

算法描述

枚举

代码

```
#include<stdio.h>
#define N 10
int a[N][N]=
{     // A,B,C,D,E,F,G,H,I
      {0,0,0,0,0,0,0,0,0,0},
      {0,1,1,0,1,1,0,0,0,0},//1
      {0,1,1,1,0,0,0,0,0,0},//2
      {0,0,1,1,0,1,1,0,0,0},//3
      {0,1,0,0,1,0,0,1,0,0},//4
      {0,0,1,0,1,1,1,0,1,0},//5
      {0,0,0,1,0,0,1,0,0,1},//6
      {0,0,0,0,1,1,0,1,1,0},//7
      {0,0,0,0,0,0,0,1,1,1},//8yy
      {0,0,0,0,0,1,1,0,1,1} //9
};
int main ()
{
      int i, j, k, flag, m[N], clock[N], temp[N];
      for ( i=1; i<=9; i++ )
      {
            scanf ( "%d", &clock[i]);
      }
      for (m[1]=0;m[1]<=3;m[1]++)
            for (m[2]=0;m[2]<=3;m[2]++)
                  for (m[3]=0;m[3]<=3;m[3]++)
                        for (m[4]=0;m[4]<=3;m[4]++)
                              for (m[5]=0;m[5]<=3;m[5]++)
                                    for (m[6]=0;m[6]<=3;m[6]++)
                                          for (m[7]=0;m[7]<=3;m[7]++)
                                                for (m[8]=0;m[8]<=3;m[8]++)
                                                      for (m[9]=0;m[9]<=3;m[9]++)
                                                      {
                                                            for ( i=1; i<=9; i++ )
                                                                  temp[i]=clock[i];
                                                            for ( i=1; i<=9; i++)
                                                            {
                                                                  if ( m[i] > 0 )
                                                                  {
                                                                        for ( j=1; j<=9; j++ )
                                                                        {
                                                                              if ( a[i][j] == 1 )
                                                                              {
```

```
                                temp[j] += m[i];
                                temp[j] = temp[j] % 4;
                            }
                        }
                    }
                }
                flag=1;
                for ( i=1; i<=9; i++ )
                {
                    if ( temp[i]!=0 )
                    {
                        flag=0;break;
                    }
                }
                if ( flag )
                {
                    for ( i=1; i<=9; i++ )
                    {
                        k=m[i];
                        while ( k )
                        {
                            printf("%d ", i );
                            k--;
                        }
                    }
                    printf( "\n");
                    return 0;
                }
            }
}
```

Party Lamps PKU_1176

题目描述：

To brighten up the gala dinner of the IOI'98 we have a set of N coloured lamps numbered from 1 to N. The lamps are connected to four buttons:

button 1 -- when this button is pressed, all the lamps change their state: those that are ON are turned OFF and those that are OFF are turned ON.

button 2 -- changes the state of all the odd numbered lamps.

button 3 -- changes the state of all the even numbered lamps.

button 4 -- changes the state of the lamps whose number is of the form 3K+1 (with K >= 0), i.e., 1,4,7,...

There is a counter C which records the total number of button presses.
When the party starts, all the lamps are ON and the counter C is set to zero.

You are given the value of counter C and information on the final state of some of the lamps. Write a program to determine all the possible final configurations of the N lamps that are consistent with the given information, without repetitions.

Input

Your program is to read from standard input. The input contains four lines, describing the number N of lamps available, the number C of button presses, and the state of some of the lamps in the final configuration.
The first line contains the number N and the second line the final value of counter C. The third line lists the lamp numbers you are informed to be ON in the final configuration, separated by one space and terminated by the integer -1. The fourth line lists the lamp numbers you are informed to be OFF in the final configuration, separated by one space and terminated by the integer -1.

The parameters N and C are constrained by:
10 <= N <= 100
1 <= C <= 10000
The number of lamps you are informed to be ON, in the final configuration, is less than or equal to 2.The number of lamps you are informed to be OFF, in the final configuration, is less than or equal to 2.

Output

Your program is to write to standard output. The output must contain all the possible final configurations (without repetitions) of all the lamps. There is at least one possible final configuration. Each possible configuration must be written on a different line. Each line has N characters, where the first character represents the state of lamp 1 and the last character represents the state of lamp N. A 0 (zero) stands for a lamp that is OFF, and a 1 (one) stands for a lamp that is ON. Configurations should be listed in binary ascending order.

Sample Input

10
1
-1
7 -1

Sample Output

0000000000
0101010101
0110110110

算法描述

枚举+剪枝

代码

```
#include<stdio.h>
#include<string.h>
#define N 102
void qsort( char l[N][N], int start, int end )
{
	int i,j;
	char temp1[N];
	if( start < end )
	{
		i=start,j=end;
		strcpy(temp1,l[i]);
		while(i!=j)
		{
			while(j>i&&strcmp(l[j],temp1)>=0) j--;
			if(i<j)
			{
				strcpy(l[i],l[j]);
				i++;
			}
			while(i<j&&strcmp(l[i],temp1)<=0) i++;
			if(i<j)
			{
				strcpy(l[j],l[i]);
				j--;
			}
		}
		strcpy(l[i],temp1);
		qsort(l,start,i-1);
		qsort(l,i+1,end);
	}
}
int main ()
{
	int i, k, n, c, t, on[N], off[N], m[N], b[5], num1, num2, flag;
	char p[N][N];
	scanf ( "%d%d", &n, &c ) ;
	i=0;
```

```
while( 1 )
{
	scanf ( "%d", &t);
	if ( t == -1 )
		break;
	on[i++] = t;
}
num1=i;
i=0;
while( 1 )
{
	scanf ( "%d", &t);
	if ( t == -1 )
		break;
	off[i++] = t;
}
num2=i;
k=0;
for ( b[1] = 0; b[1] < 2; b[1]++ )//总共只有 16 中可能，枚举所有可能
{
	for ( b[2] = 0; b[2] < 2; b[2]++ )
	{
		for ( b[3] = 0; b[3] < 2; b[3]++ )
		{
			for ( b[4] = 0; b[4] < 2; b[4]++ )
			{
				for ( i = 0; i < n; i++ )
				{
					m[i] = 1;
				}
				//一开一关，c 会累积
				if ( c < b[1] + b[2] + b[3] + b[4]  )//c 存储的是最多的操作次数
				{
					continue;
				}
				if ( c > b[1] + b[2] + b[3] + b[4] )
				{
					if ( (c - (b[1] + b[2] + b[3] + b[4])) %2 ==1 )
					{
						continue;
					}
				}
				if ( b[1] == 1 )
				{
```

```
        for ( i = 0; i < n; i++ )
        {
                m[i] += 1;
                m[i] = m[i] % 2;
        }
}
if ( b[2] == 1 )
{
        for ( i = 0; i < n ; i=i+2 )
        {
                m[i] += 1;
                m[i] = m[i] % 2;
        }
}
if ( b[3] == 1 )
{
        for ( i = 1; i < n ; i=i+2 )
        {
                m[i] += 1;
                m[i] = m[i] % 2;
        }
}
if ( b[4] == 1 )
{
        for ( i = 0; i < n ; i=i+3 )
        {
                m[i] += 1;
                m[i] = m[i] % 2;
        }
}
flag = 1;
for ( i = 0; i < num1; i++ )
{
        if ( m[on[i]-1] != 1 )
        {
                flag = 0;
                break;
        }
}
for ( i = 0; i < num2; i++ )
{
        if ( m[off[i]-1] != 0 )
        {
                flag = 0;
```

```
                                break;
                            }
                        }
                        if ( flag )
                        {
                            for ( i = 0; i < n; i++ )
                            {
                                p[k][i] = m[i] + '0';
                            }
                            p[k][i]='\0';
                            k++;
                        }
                    }
                }
            }
        }
        qsort ( p, 0, k-1);
        for ( i = 0; i < k; i++ )
        {
            printf( "%s\n", p[i] );
        }
        return 0;
}
```

生日蛋糕 PKU_1190

题目描述：

7 月 17 日是 Mr.W 的生日，ACM-THU 为此要制作一个体积为 Nπ的 M 层生日蛋糕，每层都是一个圆柱体。
设从下往上数第 i(1 <= i <= M)层蛋糕是半径为 Ri, 高度为 Hi 的圆柱。当 i < M 时，要求 Ri > Ri+1 且 Hi > Hi+1。
由于要在蛋糕上抹奶油，为尽可能节约经费，我们希望蛋糕外表面（最下一层的下底面除外）的面积 Q 最小。
令 Q = Sπ
请编程对给出的 N 和 M，找出蛋糕的制作方案（适当的 Ri 和 Hi 的值），使 S 最小。
（除 Q 外，以上所有数据皆为正整数）

Input

有两行，第一行为 N（N <= 10000），表示待制作的蛋糕的体积为 Nπ；第二行为 M(M <= 20)，表示蛋糕的层数为 M。

Output

仅一行，是一个正整数 S（若无解则 S = 0）。

Sample Input

100
2

Sample Output

68

Hint

圆柱公式
体积 $V = \pi R^2 H$
侧面积 $A' = 2\pi RH$
底面积 $A = \pi R^2$

算法描述

Dfs+剪枝

代码

```
#include<stdio.h>
int minv[101], maxv[101][101];
int maxr=100;
int maxh=100;
long int mins;
int n, m;
void init()//估算最小最大体积
{
	int i, j;
	minv[0]=0;
	for ( i=1; i<=maxr; i++)
	{
		minv[i]=minv[i-1]+i*i*i;
	}
	for ( i=0; i<=maxr; i++)
	{
		maxv[i][0]=0;
		maxv[0][i]=0;
	}
```

```
    for ( i=1; i<=maxr; i++)
    {
        for ( j=1; j<=maxh; j++)
        {
            maxv[i][j]=maxv[i-1][j-1]+i*i*j;
        }
    }
}
void dfs (int f, int r, int h, int s, int v )
{
    v -= r*r*h;
    s += 2*r*h;
    if ( f==m )
    {
        if ( v==0&&s<mins)
                mins=s;
        return;
    }
    if ( v<=0 || s>mins)//剪枝
        return;
    int left=m-f;
    int i, j;
    if ( v<minv[left] )//剪枝，如果剩余体积小于当前层以上所有层的最小体积则剪枝
        return ;
    if ( v>maxv[r-1][h-1]-maxv[r-1-left][h-1-left])//剪枝，如果剩余体积大于当前层以上所有层的最大体积则剪枝
        return ;
    f++;
    for ( i=r-1;i>=left;i--)
    {
        for ( j=h-1; j>=left; j--)
        {
            dfs( f, i, j, s, v);
        }
    }
}
int main ()
{
    int    i, j;
    scanf ("%d%d", &n, &m);
    init();
    mins=1000000;
    for ( i=maxr; i>=m; i--)
    {
```

```
            for (j=maxh; j>=m; j--)
            {
                dfs(1, i, j, i*i, n);
            }
        }
        if ( mins==1000000)
            printf("0\n");
        else
            printf("%ld\n", mins);
        return 0;
}
```

The Alphabet Game PKU_1231

题目描述：

Little Dara has recently learned how to write a few letters of the English alphabet (say k letters). He plays a game with his little sister Sara. He draws a grid on a piece of paper and writes p instances of each of the k letters in the grid cells. He then asks Sara to draw as many side-to-side horizontal and/or vertical bold lines over the grid lines as she wishes, such that in each rectangle containing no bold line, there would be p instances of one letter or nothing. For example, consider the sheet given in Figure 1, where Sara has drawn two bold lines creating four rectangles meeting the condition above. Sara wins if she succeeds in drawing the required lines. Dara being quite fair to Sara, wants to make sure that there would be at least one solution to each case he offers Sara. You are to write a program to help Dara decide on the possibility of drawing the right lines.

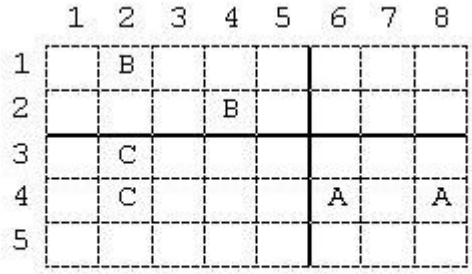


Figure 1. Part of a sample sheet and two dividing bold lines: The data for this figure is given in the first test case of the sample input

Input

The first line of the input file contains a single integer t (1 <= t <= 10), the number of test cases, followed by the input data for each test case. The first line of each test case consists of two integers k (1 <= k <= 26), the number of different letters, and p (1 <= p <= 10), the number of instances of each letter. Followed by the first line, there are k lines, one for each letter, each

containing p pairs of integers (xi, yi) for 1 <= i <= p. A pair indicates coordinates of the cell on the paper where one instance of the letter is written. The coordinates of the upper left cell of the paper is assumed to be (1,1). Coordinates are positive integers less than or equal to 1,000,000. You may assume that no cell contains more than one letter.

Output

There should be one line per test case containing a single word YES or NO depending on whether the input paper can be divided successfully according to the constraints stated in the problem.

Sample Input

```
2
3 2
6 4 8 4
4 2 2 1
2 3 2 4
3 3
1 1 3 1 5 1
2 1 4 1 6 1
2 2 4 2 8 1
```

Sample Output

```
YES
NO
```

算法描述

枚举切割线的位置

代码

```
#include<stdio.h>
#include<string.h>
int main ()
{
    int i, j, n, flag, tag, a, b, p, k, grid[6][9], x, y, g;
    scanf ( "%d", &n);
    while ( n-- )
    {
        scanf ( "%d%d", &k, &p);
        for ( i=0; i<6; i++)
        {
            for ( j=0; j<9; j++ )
```

```
        {
            grid[i][j]=0;
        }
}
for ( i=0; i<k; i++ )
{
        for ( j=0; j<p; j++ )
        {
            scanf ( "%d%d", &a, &b);
            grid[b][a]=i+1;
        }
}
y=0;
x=0;
for ( a=0; a<8; a++ )
{
        for ( b=0, flag=0; b<5; b++)
        {
            for ( i=0,g=1; i<=a; i++ )
            {
                for ( j=0; j<=b; j++)
                {
                    if ( g&&grid[j][i]!=0 )
                    {
                        x=grid[j][i];
                        g=0;
                        continue;
                    }
                    if ( grid[j][i]!=x && grid[j][i]!=y )
                    {
                        flag=1;
                        break;
                    }
                }
                if (flag)
                    break;
                for ( i=a+1,g=1; i<=8; i++ )
                {
                    for ( j=0; j<=b; j++)
                    {
                        if ( g&&grid[j][i]!=0 )
                        {
                            x=grid[j][i];
                            g=0;
```

```
                continue;
            }
            if ( grid[j][i]!=x && grid[j][i]!=y )
            {
                flag=1;
                break;
            }
        }
    }
    if (flag)
        break;
    for ( i=0,g=1; i<=a; i++ )
    {
        for ( j=b+1; j<=5; j++)
        {
            if ( g&&grid[j][i]!=0 )
            {
                x=grid[j][i];
                g=0;
                continue;
            }
            if ( grid[j][i]!=x && grid[j][i]!=y )
            {
                flag=1;
                break;
            }
        }
        if (flag)
            break;
    }
    if (flag)
        break;
    for ( i=a+1,g=1; i<=8; i++ )
    {
        for ( j=b+1; j<=5; j++)
        {
            if ( g&&grid[j][i]!=0 )
            {
                x=grid[j][i];
                g=0;
                continue;
            }
            if ( grid[j][i]!=x && grid[j][i]!=y )
            {
```

```
                                flag=1;
                                break;
                            }
                        }
                    }
                    if (flag)
                        break;
                }
                tag=0;
                if (flag==0)
                {
                    printf("YES\n");
                    tag=1;
                    break;
                }
            }
            if (tag)
                break;
        }
        if (tag==0)
            printf( "NO\n");
    }
    return 0;
}
```

Anagram PKU_1256

题目描述：

You are to write a program that has to generate all possible words from a given set of letters.
Example: Given the word "abc", your program should - by exploring all different combination of the three letters - output the words "abc", "acb", "bac", "bca", "cab" and "cba".
In the word taken from the input file, some letters may appear more than once. For a given word, your program should not produce the same word more than once, and the words should be output in alphabetically ascending order.

Input

The input consists of several words. The first line contains a number giving the number of words to follow. Each following line contains one word. A word consists of uppercase or lowercase letters from A to Z. Uppercase and lowercase letters are to be considered different. The length of each word is less than 13.

Output

For each word in the input, the output should contain all different words that can be generated with the letters of the given word. The words generated from the same input word should be output in alphabetically ascending order. An upper case letter goes before the corresponding lower case letter.

Sample Input

3
aAb
abc
acba

Sample Output

Aab
Aba
aAb
abA
bAa
baA
abc
acb
bac
bca
cab
cba
aabc
aacb
abac
abca
acab
acba
baac
baca
bcaa
caab
caba
cbaa

Hint

An upper case letter goes before the corresponding lower case letter.
So the right order of letters is 'A'<'a'<'B'<'b'<...<'Z'<'z'.

算法描述

先排序，无需剪枝的深度搜索

代码

```
#include<stdio.h>
#include<string.h>
#define N 1000
char s[N], q[N];
int p[N], n;
double h[N];
void qsort(double *l,char *k, int start,int end)
{
	int i,j;
	double temp1;
	char temp2;
	if(start<end)
	{
		i=start,j=end;
		temp1=l[i];
		temp2=k[i];
		while(i!=j)
		{
			while(j>i&&l[j]>=temp1) j--;
			if(i<j)
			{
				l[i]=l[j];
				k[i]=k[j];
				i++;
			}
			while(i<j&&l[i]<=temp1) i++;
			if(i<j)
			{
				l[j]=l[i];
				k[j]=k[i];
				j--;
			}
		}
		l[i]=temp1;
		k[i]=temp2;
		qsort(l,k,start,i-1);
		qsort(l,k,i+1,end);
```

```
        }
}
void search ( int k )
{
        int i;
        if ( k==n )
        {
                q[k]='\0';
                puts(q);
        }
        else
        {
                for ( i=0; i<n; i++)
                {
                        if(p[i]!=1)
                        {
                                p[i]=1;
                                q[k]=s[i];//printf("q[%d]=%c\n", k, q[k]);
                        }
                        else
                                continue;
                        search(k+1);//printf("!\n");
                        p[i]=0;
                        while(s[i]==s[i+1]) i++;
                }
        }
}
int main ()
{
        int i,t;
        scanf ("%d", &t);
        getchar();
        while ( t-- )
        {
                gets(s);
                n=strlen(s);
                for ( i=0; i<n; i++)
                {
                        if(s[i]<='Z' &&s[i]>='A')
                        {
                                h[i]=s[i]-'A'+0.5;
                        }
                        else
                                h[i]=s[i]-'a'+1;
```

```
            }
            qsort(h,s,0,n-1);
            search(0);
        }
        return 0;
}
```

Following Orders PKU_1270

题目描述：

Order is an important concept in mathematics and in computer science. For example, Zorn's Lemma states: ``a partially ordered set in which every chain has an upper bound contains a maximal element.'' Order is also important in reasoning about the fix-point semantics of programs.

This problem involves neither Zorn's Lemma nor fix-point semantics, but does involve order.
Given a list of variable constraints of the form x < y, you are to write a program that prints all orderings of the variables that are consistent with the constraints.

For example, given the constraints x < y and x < z there are two orderings of the variables x, y, and z that are consistent with these constraints: x y z and x z y.

Input

The input consists of a sequence of constraint specifications. A specification consists of two lines: a list of variables on one line followed by a list of contraints on the next line. A constraint is given by a pair of variables, where x y indicates that x < y.

All variables are single character, lower-case letters. There will be at least two variables, and no more than 20 variables in a specification. There will be at least one constraint, and no more than 50 constraints in a specification. There will be at least one, and no more than 300 orderings consistent with the contraints in a specification.

Input is terminated by end-of-file.

Output

For each constraint specification, all orderings consistent with the constraints should be printed. Orderings are printed in lexicographical (alphabetical) order, one per line.

Output for different constraint specifications is separated by a blank line.

Sample Input

a b f g
a b b f
v w x y z
v y x v z v w v

Sample Output

abfg
abgf
agbf
gabf

wxzvy
wzxvy
xwzvy
xzwvy
zwxvy
zxwvy

算法描述

拓扑排序+深度搜索

代码

```
#include<stdio.h>
#include<string.h>
#define N 300
char s[N], q[N];
int p[27][27], n, flag[N], tag[N];
void sort (int k)//深度搜索过程
{
	int i, j;
	if ( k==n )
	{
		q[k]='\0';
		puts(q);
	}
	else
```

```
    {
        for ( i=1; i<=26; i++)
        {
            if ( p[i][0]== 0 && flag[i]==0 )
            {
                flag[i]=1;
                for ( j=1; j<=26; j++ )
                {
                    if (p[j][i]==1)
                    {
                        p[j][0]--;
                        p[j][i]=2;
                    }
                }
                q[k] = i+'a'-1;
            }
            else
                continue;
            sort(k+1);
            flag[i]=0;
            for ( j=1; j<=26; j++ )
            {
                if(p[j][i]==2)
                {
                    p[j][0]++;
                    p[j][i]=1;
                }
            }
        }
    }
}
int main ()
{
    int i, j, k, m;
    char a[N], b[N];
    while ( gets(s)!=NULL)
    {
        n=strlen(s);
        for ( i=0, j=0; i<n; i++)
        {
            if ( s[i]==' ')
                continue;
            s[j++]=s[i];
        }
```

```
            s[j]='\0';
            n=strlen(s);
            gets(q);
            m=strlen(q);
            for ( i=1; i<=26; i++)
            {
                  p[i][0]=-1;
                  for ( k=0; k<n; k++)
                  {
                        if ( i==s[k]-'a'+1)
                              p[i][0]=0;
                  }
                  for ( j=1; j<=26; j++)
                  {
                        p[i][j]=0;
                  }
            }
            for ( i=0, j=0; i<m; i=i+4)
            {
                  a[j]=q[i];
                  b[j++]=q[i+2];
            }
            m=j;
            for ( i=0; i<n; i++)//作顺序的标记
            {
                  for( j=0; j<m; j++)
                  {
                        if ( s[i]==b[j] )
                        {
                              p[b[j]-'a'+1][0]++;
                              p[b[j]-'a'+1][a[j]-'a'+1]=1;
                        }
                  }
            }
            for ( i=1; i<=26; i++)
            {
                  flag[i]=0;
                  tag[i]=0;
            }
            sort(0);
            printf("\n");
      }
      return 0;
}
```

Perfect Cubes PKU_1543

题目描述：

For hundreds of years Fermat's Last Theorem, which stated simply that for n > 2 there exist no integers a, b, c > 1 such that a^n = b^n + c^n, has remained elusively unproven. (A recent proof is believed to be correct, though it is still undergoing scrutiny.) It is possible, however, to find integers greater than 1 that satisfy the "perfect cube" equation a^3 = b^3 + c^3 + d^3 (e.g. a quick calculation will show that the equation 12^3 = 6^3 + 8^3 + 10^3 is indeed true). This problem requires that you write a program to find all sets of numbers {a,b,c,d} which satisfy this equation for a <= N.

Input

One integer N (N <= 100).

Output

The output should be listed as shown below, one perfect cube per line, in non-decreasing order of a (i.e. the lines should be sorted by their a values). The values of b, c, and d should also be listed in non-decreasing order on the line itself. There do exist several values of a which can be produced from multiple distinct sets of b, c, and d triples. In these cases, the triples with the smaller b values should be listed first.

Sample Input

24

Sample Output

Cube = 6, Triple = (3,4,5)
Cube = 12, Triple = (6,8,10)
Cube = 18, Triple = (2,12,16)
Cube = 18, Triple = (9,12,15)
Cube = 19, Triple = (3,10,18)
Cube = 20, Triple = (7,14,17)
Cube = 24, Triple = (12,16,20)

算法描述

枚举

代码

```
#include<stdio.h>
long int n, p;
int main()
{
	long int i, j, k, h;
	scanf ("%ld", &n);
	for ( h = 3; h <= n; h++)//枚举
	{
		for ( i = 2; i < h; i++)
		{
			for ( j = i; j < h; j++)
			{
				for ( k = j; k < h; k++)
				{
					if ( i*i*i + j*j*j + k*k*k == h*h*h )
					{
						printf("Cube = %ld, Triple = (%ld,%ld,%ld)\n", h, i, j, k);
						break;
					}
				}
			}
		}
	}
	return 0;
}
```

Function Run Fun PKU_1579

题目描述：

We all love recursion! Don't we?

Consider a three-parameter recursive function w(a, b, c):

if a <= 0 or b <= 0 or c <= 0, then w(a, b, c) returns:
1

if a > 20 or b > 20 or c > 20, then w(a, b, c) returns:
w(20, 20, 20)

if a < b and b < c, then w(a, b, c) returns:
w(a, b, c-1) + w(a, b-1, c-1) - w(a, b-1, c)

otherwise it returns:
w(a-1, b, c) + w(a-1, b-1, c) + w(a-1, b, c-1) - w(a-1, b-1, c-1)

This is an easy function to implement. The problem is, if implemented directly, for moderate values of a, b and c (for example, a = 15, b = 15, c = 15), the program takes hours to run because of the massive recursion.

Input

The input for your program will be a series of integer triples, one per line, until the end-of-file flag of -1 -1 -1. Using the above technique, you are to calculate w(a, b, c) efficiently and print the result.

Output

Print the value for w(a,b,c) for each triple.

Sample Input

1 1 1
2 2 2
10 4 6
50 50 50
-1 7 18
-1 -1 -1

Sample Output

w(1, 1, 1) = 2
w(2, 2, 2) = 4
w(10, 4, 6) = 523
w(50, 50, 50) = 1048576
w(-1, 7, 18) = 1

算法描述

记忆化搜索

代码

```
#include<stdio.h>
#define N 21
int main ()
```

```
{
    __int64 i, j, k, w[N][N][N], a, b, c;
    while ( scanf ("%I64d%I64d%I64d", &a, &b, &c)!=EOF)
    {
        if ( a==-1 &&b==-1&&c==-1)
            break;
        for ( i=0; i<=20; i++)
        {
            for ( j=0; j<=20; j++)
            {
                for ( k=0; k<=20; k++)
                {
                    if ( i==0 || j==0 || k==0 )
                        w[i][j][k] = 1;
                    else if ( i<j && j<k)
                        w[i][j][k] = w[i][j][k-1] + w[i][j-1][k-1] - w[i][j-1][k];
                    else
                        w[i][j][k] = w[i-1][j][k] + w[i-1][j-1][k] + w[i-1][j][k-1] - w[i-1][j-
1][k-1];
                }
            }
        }
        if ( a<=0 ||b <=0 || c<=0)
            printf("w(%I64d, %I64d, %I64d) = %I64d\n", a, b, c, w[0][0][0]);
        else if ( a>20 || b>20 ||c>20 )
            printf("w(%I64d, %I64d, %I64d) = %I64d\n", a, b, c, w[20][20][20]);
        else
            printf("w(%I64d, %I64d, %I64d) = %I64d\n", a, b, c, w[a][b][c]);
    }
    return 0;
}
```

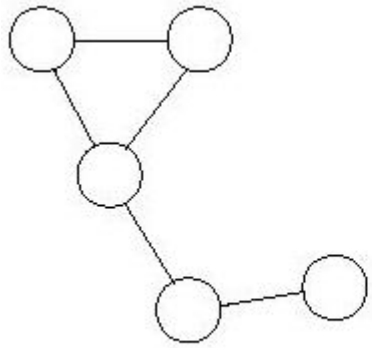
Phone Home PKU_1620

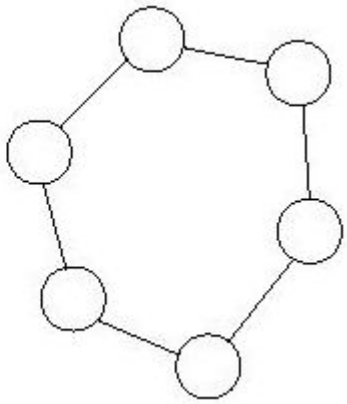
题目描述：

When relay towers for mobile telephones communicate with the mobile phones in their area, there is always the possibility of interference. So, when assigning the transmission frequency, the FCC makes sure that nearby towers have frequencies that aren't too close. On the other hand, the FCC does not want to assign too many different frequencies; they want to save as many as possible for other uses.

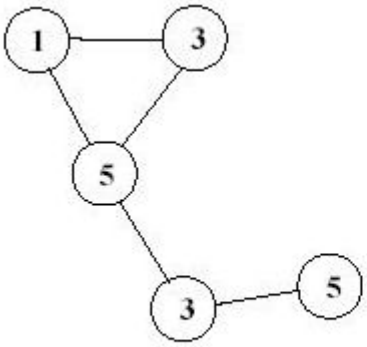
Your job is to find an optimal assignment of frequencies.

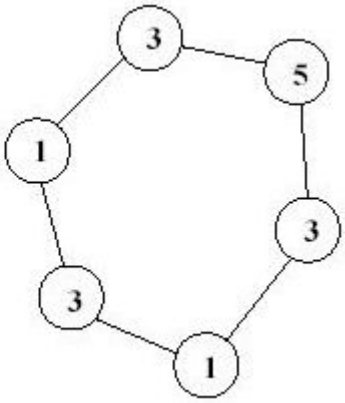
In this problem, the frequencies will be integers. Nearby towers must be assigned frequencies that differ by at least 2. You'll find an assignment using as few frequencies as possible. For example, consider the following two arrangements of towers. Two towers near each other are indicated by the connecting line.





Note that the following are legal frequency assignments to these two tower configurations. However,the second arrangement does not use the fewest number of frequencies possible, since the tower with frequency 5 could have frequency 1.





Input

There will be multiple test cases. Input for each test case will consist of two lines: the first line will contain the integer n, indicating the number of towers. The next line will be of the form x1 y1 x2 y2 ... xn yn where xi yi are the coordinates of tower i. A pair of towers are considered "near" each other if the distance between them is no more than 20. There will be no more than 12 towers and no tower will have more than 4 towers near it. A value of n = 0 indicates end of input.

Output

For each test case, you should print one line in the format:
The towers in case n can be covered in f frequencies.
where you determine the value for f. The case numbers, n, will start at 1.

Sample Input

5

0 0 5 7.5 1 -3 10.75 -20.1 12.01 -22
6
0 1 19 0 38 1 38 21 19 22 0 21
0

Sample Output

```
The towers in case 1 can be covered in 3 frequencies.
The towers in case 2 can be covered in 2 frequencies.
```

算法描述

深度搜索，着色问题变种

代码

```
#include<stdio.h>
#include<math.h>
#define N 13
int main()
{
    int i, j, n, a[N][N], max, color[N], k, t=0;
    float x[N], y[N], d;
    while ( scanf ( "%d", &n ) != EOF )
    {
        t++;
        if(n==0)
            break;
        for ( i=1; i<=n; i++ )
            scanf ( "%f %f", &x[i], &y[i] );
        for ( i=1; i<=n; i++ )
        {
            for ( j=1; j<=n; j++ )
                a[i][j]=0;
        }
        for ( i=1; i<=n; i++ )
        {
            for ( j=i+1; j<=n; j++ )
            {
                d = sqrt( ( x[i] - x[j] ) * ( x[i] - x[j] ) + ( y[i] - y[j] ) * ( y[i] - y[j] ) );
                if ( d <= 20 )
                {
                    a[i][j]=1;
```

```
                        a[j][i]=1;
                    }
                }
            }
            for ( i=1; i<=n ;i++ )
                color[i]=0;
            k=1;
            while ( k >= 1 )//dfs
            {
                color[k]++;
                for ( i = 1; i < k; i++ )
                    if ( color[i]==color[k] && a[i][k]==1 )
                        color[k]++;
                if ( n == k )
                {
                    for ( i=1, max=1; i<=n; i++ )
                        if ( color[i] > max )
                            max=color[i];
                    printf( "The towers in case %d can be covered in %d frequencies.\n", t, max );
                    break;
                }
                else
                    k++;
            }
        }
        return 0;
}
```

放苹果 PKU_1664

题目描述：

把 M 个同样的苹果放在 N 个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的分法？（用 K 表示）5，1，1 和 1，5，1 是同一种分法。

Input

第一行是测试数据的数目 t（0 <= t <= 20）。以下每行均包含二个整数 M 和 N，以空格分开。1<=M，N<=10。

Output

对输入的每组数据 M 和 N，用一行输出相应的 K。

Sample Input

1
7 3

Sample Output

8

算法描述

递归的深度搜索

代码

```
#include<stdio.h>
int n, m, cnt, num;
void dfs ( int k, int p )
{
	int	i;
	if ( k==n )
	{
		if ( cnt == m)
			num++;
		else
			return ;
	}
	else
	{
		for ( i=p; i>=0; i--)
		{
			if ( cnt + i <= m	)
				cnt += i;
			else
				continue;
			dfs ( k+1, i);
			cnt -= i;
		}
	}
}
int main ()
{
	int t ;
	scanf ( "%d", &t );
```

```
    while ( t-- )
    {
        scanf ("%d%d", &m, &n);
        cnt=0;
        num=0;
        dfs ( 0, m );
        printf( "%d\n", num);
    }
    return 0;
}
```

Orders PKU_1731

题目描述：

The stores manager has sorted all kinds of goods in an alphabetical order of their labels. All the kinds having labels starting with the same letter are stored in the same warehouse (i.e. in the same building) labelled with this letter. During the day the stores manager receives and books the orders of goods which are to be delivered from the store. Each order requires only one kind of goods. The stores manager processes the requests in the order of their booking.

You know in advance all the orders which will have to be processed by the stores manager today, but you do not know their booking order. Compute all possible ways of the visits of warehouses for the stores manager to settle all the demands piece after piece during the day.

Input

Input contains a single line with all labels of the requested goods (in random order). Each kind of goods is represented by the starting letter of its label. Only small letters of the English alphabet are used. The number of orders doesn't exceed 200.

Output

Output will contain all possible orderings in which the stores manager may visit his warehouses. Every warehouse is represented by a single small letter of the English alphabet -- the starting letter of the label of the goods. Each ordering of warehouses is written in the output file only once on a separate line and all the lines containing orderings have to be sorted in an alphabetical order (see the example). No output will exceed 2 megabytes.

Sample Input

bbjd

Sample Output

bbdj

bbjd
bdbj
bdjb
bjbd
bjdb
dbbj
dbjb
djbb
jbbd
jbdb
jdbb

算法描述

深度搜索+剪枝

代码

```
#include<stdio.h>
#include<string.h>
#define N 201
char s[N], p[N];
int n, visited[N];
void qsort ( int start, int end)
{
	int i, j;
	char temp;
	i=start;
	j=end;
	temp=s[i];
	if ( start < end )
	{
		while ( i!=j)
		{
			while ( i<j && s[j]>=temp)	j--;
			if ( i<j)
			{
				s[i]=s[j];
				i++;
			}
			while ( i<j && s[i] <= temp ) i++;
			if ( i<j)
```

```
                {
                    s[j]=s[i];
                    j--;
                }
            }
            s[i]=temp;
            qsort ( start, i-1);
            qsort ( i+1, end);
        }
}
void dfs ( int k )
{
        int i, j;
        if ( k == n )
        {
            p[k]='\0';
            puts( p);
            j=0;
        }
        else
        {
            for ( i=0; i<n; i++)
            {
                if ( visited[i]==0)
                {
                    p[k]=s[i];
                    visited[i]=1;
                }
                else
                    continue;
                dfs(k+1);
                visited[i]=0;
                while ( s[i]==s[i+1]) i++;//剪枝
            }
        }
}
int main ()
{
        int i;
        while ( gets(s)!=NULL)
        {
            n = strlen (s);
            for ( i=0; i<n; i++)
                visited[i]=0;
```

```
            qsort(0, n-1);
            dfs(0);
        }
        return 0;
}
```

Divisibility PKU_1745

题目描述：

Consider an arbitrary sequence of integers. One can place + or - operators between integers in the sequence, thus deriving different arithmetical expressions that evaluate to different values. Let us, for example, take the sequence: 17, 5, -21, 15. There are eight possible expressions: 17 + 5 + -21 + 15 = 16
17 + 5 + -21 - 15 = -14
17 + 5 - -21 + 15 = 58
17 + 5 - -21 - 15 = 28
17 - 5 + -21 + 15 = 6
17 - 5 + -21 - 15 = -24
17 - 5 - -21 + 15 = 48
17 - 5 - -21 - 15 = 18
We call the sequence of integers divisible by K if + or - operators can be placed between integers in the sequence in such way that resulting value is divisible by K. In the above example, the sequence is divisible by 7 (17+5+-21-15=-14) but is not divisible by 5.

You are to write a program that will determine divisibility of sequence of integers.

Input

The first line of the input file contains two integers, N and K (1 <= N <= 10000, 2 <= K <= 100) separated by a space.
The second line contains a sequence of N integers separated by spaces. Each integer is not greater than 10000 by it's absolute value.

Output

Write to the output file the word "Divisible" if given sequence of integers is divisible by K or "Not divisible" if it's not.

Sample Input

4 7
17 5 -21 15

Sample Output

Divisible

算法描述

搜索超时，该算法是将每一阶段可能获得的值通过+k,-k,%k 等操作将值标记到一个从 0 到 k 的数组中，全部处理完毕以后，再看数组中下标为 0 的位置是否被表示，若是则 divisible.

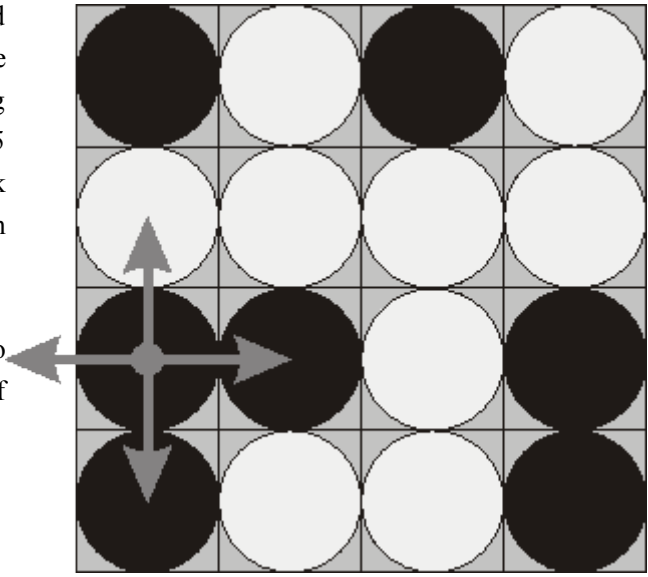
代码

```
#include<stdio.h>
int main ()
{
    int old[101], newone[101];
    int n, k, i, j, t, a;
    scanf ( "%d%d", &n, &k);
    for ( i=0; i<=k; i++)
    {
        old[i]=0;
        newone[i]=0;
    }
    old[0]=1;
    for ( i = 0; i < n; i++ )
    {
        scanf ( "%d", &t );
        for ( j = 0 ;j <= k; j++)
        {
            if ( old[j] )
            {
                a = j + t;
                while ( a < 0 )
                    a = a + k;
                newone[a % k]=1;
                a = j - t;
                while ( a < 0 )
                    a = a + k;
                newone[a % k]=1;
            }
        }
        for ( j=0; j<=k; j++)
        {
            old[j] = newone[j];
            newone[j] = 0;
```

```
            }
        }
        if ( old[0] )
            printf( "Divisible\n");
        else
            printf( "Not divisible\n");
        return 0;
}
```

Flip Game PKU_1753

题目描述：



Flip game is played on a rectangular 4x4 field with two-sided pieces placed on each of its 16 squares. One side of each piece is white and the other one is black and each piece is lying either it's black or white side up. Each round you flip 3 to 5 pieces, thus changing the color of their upper side from black to white and vice versa. The pieces to be flipped are chosen every round according to the following rules:
Choose any one of the 16 pieces.
Flip the chosen piece and also all adjacent pieces to the left, to the right, to the top, and to the bottom of the chosen piece (if there are any).

Consider the following position as an example:

bwbw
wwww
bbwb
bwwb
Here "b" denotes pieces lying their black side up and "w" denotes pieces lying their white side up. If we choose to flip the 1st piece from the 3rd row (this choice is shown at the picture), then the field will become:

bwbw
bwww
wwwb
wwwb
The goal of the game is to flip either all pieces white side up or all pieces black side up. You are to write a program that will search for the minimum number of rounds needed to achieve this goal.

Input

The input consists of 4 lines with 4 characters "w" or "b" each that denote game field position.

Output

Write to the output file a single integer number - the minimum number of rounds needed to achieve the goal of the game from the given position. If the goal is initially achieved, then write 0. If it's impossible to achieve the goal, then write the word "Impossible" (without quotes).

Sample Input

```
bwwb
bbwb
bwwb
bwww
```

Sample Output

```
4
```

算法描述

将二维用一维数组存储，黑与白用二进制表示，翻与不翻也用二进制表示。
深度搜索

代码

```
#include<stdio.h>
#include<string.h>
long int num, min, flag[16];
int ok(long int *l)
{
	int i;
	for ( i=1; i<16; i++)
	{
		if ( l[i]!=l[i-1] )
			return 0;
	}
	return 1;
}
void dfs ( int k)
{
	int i, j;
	if ( ok(flag) )
	{
```

```
            if ( num <min)
                min=num;
        }
        if ( k==16)
        {
            return ;
        }
        else
        {
            for( i=1; i>=0; i--)
            {
                if ( i==1)
                {
                    flag[k]=(flag[k]+1)%2;
                    if ( k>=4)
                        flag[k-4]=(flag[k-4]+1)%2;
                    if ( k<=11)
                        flag[k+4]=(flag[k+4]+1)%2;
                    if ( k%4>0)
                        flag[k-1]=(flag[k-1]+1)%2;
                    if ( k%4<3)
                        flag[k+1]=(flag[k+1]+1)%2;
                    num++;
                }
                dfs(k+1);
                if ( i==1)
                {
                    flag[k]=(flag[k]+1)%2;
                    if ( k>=4)
                        flag[k-4]=(flag[k-4]+1)%2;
                    if ( k<=11)
                        flag[k+4]=(flag[k+4]+1)%2;
                    if ( k%4>0)
                        flag[k-1]=(flag[k-1]+1)%2;
                    if ( k%4<3)
                        flag[k+1]=(flag[k+1]+1)%2;
                    num--;
                }
            }
        }
    }
    int main ()
    {
        int i, j, k;
```

```
    char chess[4][4];
    for ( i=0; i<4; i++)
        gets(chess[i]);
    for ( i=0, k=0; i<4; i++)
    {
        for ( j=0; j<4; j++)
        {
            if ( chess[i][j]=='w')
                flag[k++]=1;
            else
                flag[k++]=0;
        }
    }
    if ( ok(flag))
    {
        printf("0\n");
        return 0;
    }
    num=0;
    min=65535;
    dfs(0);
    if ( min==65535)
        printf( "Impossible\n");
    else
        printf("%ld\n", min);
    return 0;
}
```

Knight Moves PKU_1915

题目描述：

Background

Mr Somurolov, fabulous chess-gamer indeed, asserts that no one else but him can move knights from one position to another so fast. Can you beat him?

The Problem

Your task is to write a program to calculate the minimum number of moves needed for a knight to reach one point from another, so that you have the chance to be faster than Somurolov.

For people not familiar with chess, the possible knight moves are shown in Figure 1.

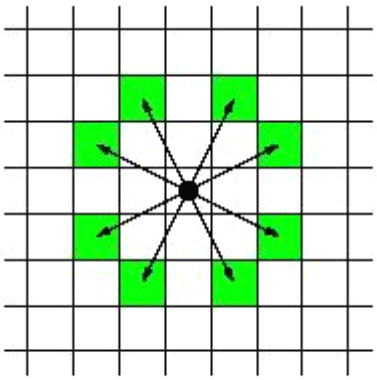


Figure 1: Possible knight moves on the board.

Input

The input begins with the number n of scenarios on a single line by itself.

Next follow n scenarios. Each scenario consists of three lines containing integer numbers. The first line specifies the length l of a side of the chess board (4 <= l <= 300). The entire board has size l * l. The second and third line contain pair of integers {0, ..., l-1}*{0, ..., l-1} specifying the starting and ending position of the knight on the board. The integers are separated by a single blank. You can assume that the positions are valid positions on the chess board of that scenario.

Output

For each scenario of the input you have to calculate the minimal amount of knight moves which are necessary to move from the starting point to the ending point. If starting point and ending point are equal,distance is zero. The distance must be written on a single line.

Sample Input

```
3
8
0 0
7 0
100
0 0
30 50
10
1 1
1 1
```

Sample Output

```
5
28
0
```

算法描述

HASH+BFS

代码

```
#include<stdio.h>
#define N 300
int t, l, x, y, num[N]={0}, visited[N][N], px, py, head, tail, queue1[N*N], queue2[N*N], n;
int a[8]={1, 2, 2, 1, -1, -2, -2, -1};
int b[8]={-2, -1, 1, 2, 2, 1, -1, -2};
int bfs ()
{
    int i;
    while ( head < tail )
    {
        while ( num[n]-- )
        {
            x = queue1[head];
            y = queue2[head];
            head++;
            visited[x][y]=1;
            for ( i=0; i<8; i++)
            {
                if (x + a[i] >= 0 &&x + a[i]< l &&  y + b[i] >= 0 &&  y + b[i] < l
                    && visited[x + a[i]][ y + b[i]] != 1 )
                {
                    if (x + a[i] == px && y + b[i] ==py )
                        return (n);
                    queue1[tail]= x + a[i];
                    queue2[tail]= y + b[i];
                    tail++;
                    visited[x + a[i]][ y + b[i]] = 1;
                    num[n+1]++;
                }
            }
        }
        n++;
    }
    return -1;
}
int main ()
{
```

```
    int i, j;
    scanf ( "%d", &t);
    while ( t-- )
    {
        scanf ( "%d", &l);
        for ( i=0; i<l; i++)
        {
            for ( j=0; j<l; j++)
            {
                visited[i][j]=0;
            }
        }
        for ( i=0; i<N; i++)
            num[i]=0;
        head=0;
        tail=0;
        scanf ( "%d%d", &x, &y);
        queue1[tail]=x;
        queue2[tail]=y;
        tail++;
        n=0;
        num[n]=1;
        scanf( "%d%d", &px, &py);
        if ( px==x && py==y)
        {
            printf("0\n");
            continue;
        }
        bfs();
        printf( "%d\n", n+1);
    }
    return 0;
}
```

Lake Counting PKU_2386

题目描述：

Due to recent rains, water has pooled in various places in Farmer John's field, which is represented by a rectangle of N x M (1 <= N <= 100; 1 <= M <= 100) squares. Each square contains either water ('W') or dry land ('.'). Farmer John would like to figure out how many ponds have formed in his field. A pond is a connected set of squares with water in them, where a square is considered

adjacent to all eight of its neighbors.

Given a diagram of Farmer John's field, determine how many ponds he has.

Input

* Line 1: Two space-separated integers: N and M

* Lines 2..N+1: M characters per line representing one row of Farmer John's field. Each character is either 'W' or '.'. The characters do not have spaces between them.

Output

* Line 1: The number of ponds in Farmer John's field.

Sample Input

10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.

Sample Output

3

Hint

OUTPUT DETAILS:

There are three ponds: one in the upper left, one in the lower left,and one along the right side.

算法描述

DFS+HASH 标记

代码

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

```
#define N 100
int n, m;
int a[8]={-1,-1,-1,0,1,1, 1, 0 };
int b[8]={-1, 0, 1,1,1,0,-1,-1 };
char map[N][N];
void dfs ( int x, int y )
{
	int i;
	for ( i=0; i<8; i++)
	{
		if ( x+a[i]>=0 && x+a[i]<n && y+b[i]<m && y+b[i]>=0
			&& map[x+a[i]][y+b[i]] =='W' )
		{
			map[x+a[i]][y+b[i]]='Q';
			dfs(x+a[i],y+b[i]);
		}
	}
}
int main ()
{
	int	num, i, j;
	scanf ("%d%d", &n, &m);
	getchar();
	for ( i=0; i<n; i++)
		gets(map[i]);
	for ( i=0, num=0; i<n; i++)
	{
		for ( j=0; j<m; j++)
		{
			if (	map[i][j]=='W')
			{
				num++;
				dfs(i, j);
			}
		}
	}
	printf( "%d\n", num);
	return 0;
}
```

单挑女飞贼 BASHU_1453

题目描述：

已知林月如和女飞贼站在一个矩阵中，矩阵中有某些障碍物不可穿越。月如使出的铜钱镖可攻击 8 个方向，但不可穿越障碍物（可视为不能穿墙的重狙）。每个单位时间，月如可向上下左右 4 个方向移动一格，攻击不浪费时间。当然，月如想尽快结束这场无聊的战斗，所以她想在最短的时间内消灭女飞贼。

Input

第一行为 2 个数 N,M 表示矩阵的规模(N 为高,M 为宽)。
接下来是 N*M 的矩阵,O 表示空地,X 表示障碍物。
下面是若干行数据，每行为一对数据，分别是女飞贼的位置和林月如的位置，显然她们都不可能在障碍物上。
以"0 0 0 0"为输入结束标志。

Output

每一组数据输出一行，仅一个整数，表示能消灭掉女飞贼的最短时间。
显然若能直接打到女飞贼，则时间为 0。
若无法消灭，则输出"Impossible!"。（不含引号）

Sample Input

3 4
OXXO
XXOO
XOOO
3 2 2 4
3 3 1 1
0 0 0 0

Sample Output

1
Impossible!

Hint

对于 30%的数据,有 N*M<=100
对于 50%的数据,有 N*M<=400
对于 100%的数据,有 N*M<=20000
对于 100%的数据,测试数据组数不超过 20 组

算法描述

双向广度搜索，也可以是 BFS+HASH

代码

```
#include<stdio.h>
#define N 1000
int a[4]={-1, 0, 1, 0};
int    b[4]={ 0, -1, 0, 1};
int c[8]={-1, -1, 0, 1, 1,    1,    0, -1};
int d[8]={ 0,    1, 1, 1, 0, -1, -1, -1};
int n, m, x, y, visited[N][N], cnt, px, py, head, tail;
int num[N], queue1[1000000], queue2[1000000], h, mark[N][N];
char cell[N][N];
void ok ()
{
      int i, k, flag[8]={0};
      k=1;
      mark[px][py]=1;
      while ( k <= (n > m?n:m) )
      {
            for ( i=0; i<8; i++)
            {
                  if(    flag[i]==0&&px    +    c[i]*k    >0    &&    px    +    c[i]*k    <=n    &&
py+d[i]*k>0&&py+d[i]*k<=m
                        &&cell[px+c[i]*k][py+d[i]*k]!='X')
                        mark[px+c[i]*k][py+d[i]*k]=1;
                  else
                        flag[i]=1;
            }
            k++;
      }
}
int bfs ()
{
      int i;
      h=0;
      num[h]=1;
      while ( head < tail )
      {
          while ( num[h]-- )
          {
```

```
            x=queue1[head];
            y=queue2[head];
            head++;
            for ( i=0; i<4; i++)
            {
                if ( x+a[i]<=n && x+a[i]>=1 && y+b[i]>=1 && y+b[i]<=m
                    &&visited[x+a[i]][y+b[i]]==0 &&cell[x+a[i]][y+b[i]]!='X')
                {
                    if ( mark[x+a[i]][y+b[i]] )
                        return (1);
                    queue1[tail]=x+a[i];
                    queue2[tail]=y+b[i];
                    tail++;
                    visited[x+a[i]][y+b[i]]=1;
                    num[h+1]++;
                }
            }
        }
        h++;
    }
    return -1;
}
int main ()
{
    int i, j, k;
    while ( scanf ( "%d %d", &n, &m) != EOF )
    {
        getchar();
        for ( i=1; i<=n; i++)
        {
            for ( j=1; j<=m; j++)
            {
                scanf ( "%c", &cell[i][j]);
            }
            getchar();
        }
        while(1)
        {
            scanf ( "%d%d%d%d", &px, &py, &x, &y);
            if ( x==0 &&y ==0&&px==0&&py==0)
            {
                return 0;
            }
            for ( i=1; i<=n; i++)
```

```
            {
                for ( j=1; j<=m; j++)
                {
                    visited[i][j]=0;
                    mark[i][j]=0;
                }
            }
            for ( i=0; i<N; i++)
                num[i]=0;
            ok();
            head=0;
            tail=0;
            if ( mark[x][y] )
            {
                printf("0\n");
                continue;
            }
            queue1[tail]=x;
            queue2[tail]=y;
            tail++;
            visited[x][y]=1;k=0;
            k=bfs();
            if (k==-1)
                printf("Impossible!\n");
            else
                printf("%d\n", h+1);
        }
    }
    return 0;
}
```

拯救博士 BASHU_1458

题目描述：

描述 Description

话说在到 BASHU 国集中的途中， Mike 博士神秘失踪了，最后发现是被外星人绑架了，幸好外星人目前还是在地球上活动，并且知道外星人不了解地球，所以使用的交通工具是最土的汽车。而且 Mike 博士身上装有用于发射移动路线的装置。

那个装置太旧了，以至于只能发射有关的移动路线的方向信息。

寻找 Mike 博士的任务便交给了你，编写程序，通过使用一张地图帮助研究所找到 Mike 博士。程序必须能表示出该车最终所有可能的位置。

地图是矩形的，上面的符号用来标明哪儿可以行车哪儿不行。"."表示那块地方是可以行进的，而符号"X"表示此处不通。而用"*"表示发现 Mike 博士失踪的地点.

车能向四个方向移动：向北(向上)，向南(向下)，向西(向左)，向东(向右)。

Input

第一行包含两个用空格隔开的自然数 R 和 C，1≤K≤200，1≤C≤200，分别表示地图中的行数和列数。

以下的 K 行中每行都包含一组符号("."或"X"或"*")用来描述地图上相应的部位。

接下来的一行包含一个自然数 N，1≤N≤1000，表示一组方向的长度。

接下来的 N 行幅行包含下述单词中的任一个：NORTH(北)、SOUTH(南)、WEST(西)和EAST(东)，表示汽车移动的方向，任何两个连续的方向都不相同。

Output

输出文件应包含用 K 行表示的地图(象输入文件中一样)，字符"*"应该仅用来表示 Mike 博士最终可能出现的位置。

Sample Input

```
3 3
...
..X
*..
3
EAST
NORTH
WEST
```

Sample Output

```
*..
*.X
...
```

算法描述

代码

```
#include <stdio.h>
int main()
{
    char map1[201][201], map2[201][201], work[1001][6];
    int mod[201][201], r, c, n, i, j, k, p;
    scanf("%d %d", &r, &c);
    for(i = 0; i < r; i++)
    {
```

```
        getchar();
        for(j = 0; j < c; j++)
        {
            scanf("%c", &map1[i][j]);
            if(map1[i][j] == 'X')
            {
                map2[i][j] = 'X';
                mod[i][j] = -1;
            }
            else if(map1[i][j] == '.')
            {
                map2[i][j] = '.';
                mod[i][j] = -2;
            }
            else if(map1[i][j] == '*')
            {
                map2[i][j] = '.';
                mod[i][j] = 0;
            }
        }
    }
    scanf("%d", &n);
    getchar();
    for(i = 0; i < n; i++)
    {
        gets(work[i]);
    }
    for(p = 0; p < n; p++)
    {
        if(p == n - 1)
        {
            for(i = 0; i < r; i++)
            {
                for(j = 0; j < c; j++)
                {
                    if(mod[i][j] == p)
                    {
                        if(work[p][0] == 'N')
                        {
                            for(k = i - 1;k >= 0; k--)
                            {
                                if(mod[k][j] == -1)
                                    break;
                                map2[k][j] = '*';
```

```
                        }
                    }
                    else if(work[p][0] == 'S')
                    {
                        for(k = i + 1; k < r; k++)
                        {
                            if(mod[k][j] == -1)
                                break;
                            map2[k][j] = '*';
                        }
                    }
                    else if(work[p][0] == 'W')
                    {
                        for(k = j - 1; k >= 0; k--)
                        {
                            if(mod[i][k] == -1)
                                break;
                            map2[i][k] = '*';
                        }
                    }
                    else if(work[p][0] == 'E')
                    {
                        for(k = j + 1; k < c; k++)
                        {
                            if(mod[i][k] == -1)
                                break;
                            map2[i][k] = '*';
                        }
                    }
                }
            }
        }
    }
    else
    {
        for(i = 0; i < r; i++)
        {
            for(j = 0; j < c; j++)
            {
                if(mod[i][j] == p)
                {
                    if(work[p][0] == 'N')
                    {
                        for(k = i - 1;k >= 0; k--)
```

```
                                {
                                    if(mod[k][j] == -1)
                                        break;
                                    mod[k][j] = p + 1;
                                }
                            }
                            else if(work[p][0] == 'S')
                            {
                                for(k = i + 1; k < r; k++)
                                {
                                    if(mod[k][j] == -1)
                                        break;
                                    mod[k][j] = p + 1;
                                }
                            }
                            else if(work[p][0] == 'W')
                            {
                                for(k = j - 1; k >= 0; k--)
                                {
                                    if(mod[i][k] == -1)
                                        break;
                                    mod[i][k] = p + 1;
                                }
                            }
                            else if(work[p][0] == 'E')
                            {
                                for(k = j + 1; k < c; k++)
                                {
                                    if(mod[i][k] == -1)
                                        break;
                                    mod[i][k] = p + 1;
                                }
                            }
                            mod[i][j] = -2;
                        }
                    }
                }
            }
        }
    }
    for(i = 0; i < r; i++)
    {
        for(j = 0; j < c; j++)
            printf("%c", map2[i][j]);
        printf("\n");
```

```
    }
    return 0;
}
```

亲戚 BASHU_1083

题目描述：

若某个家族人员过于庞大，要判断两个是否是亲戚，确实还很不容易，现在给出某个亲戚关系图，求任意给出的两个人是否具有亲戚关系。
规定：x 和 y 是亲戚，y 和 z 是亲戚，那么 x 和 z 也是亲戚。如果 x,y 是亲戚，那么 x 的亲戚都是 y 的亲戚，y 的亲戚也都是 x 的亲戚。

Input

第一行：三个整数 n,m,p,（ n<=5000,m<=5000,p<=5000），分别表示有 n 个人，m 个亲戚关系，询问 p 对亲戚关系。
以下 m 行：每行两个数 Mi，Mj，1<=Mi，Mj<=N，表示 Ai 和 Bi 具有亲戚关系。
接下来 p 行：每行两个数 Pi，Pj，询问 Pi 和 Pj 是否具有亲戚关系。

Output

P 行，每行一个’Yes’或’No’。表示第 i 个询问的答案为“具有”或“不具有”亲戚关系。

Sample Input

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

Sample Output

```
Yes
Yes
No
```

算法描述

BFS+连通分量

代码

```
#include <stdio.h>
int a[5001][5001]={0}, con[5001], b[5001]={0}, visited[5001], q[100000];
int rear, front;
void bfs ( int i )
{
	int x, j;
	front=0;
	rear=0;
	q[rear++]=i;
	while ( front <= rear )
	{
		x=q[front++];
		if(!visited[x])
			visited[x]=1;
		j=0;
		while ( a[x][j]!=0 )
		{
			if ( !visited[a[x][j]] )
			{
				q[rear++]=a[x][j];
				visited[a[x][j]]=1;
				con[a[x][j]]=con[x];
			}
			j++;
		}
	}
}
int main ()
{
	int i,j, n, m, p, x, y, b[5001]={0};
	scanf ( "%d%d%d", &n, &m, &p);
	while ( m-- )
	{
		scanf ( "%d%d", &x, &y);
		a[x][b[x]]=y;
		b[x]++;
		a[y][b[y]]=x;
```

```
        b[y]++;
    }
    for ( i=1;i<=n; i++)
    {
        con[i]=i;
        visited[i]=0;
    }
    bfs (1);
    for ( i=1; i<=n; i++)//若是非连通图，则要穷举所有不连通的的树
    {
        if(!visited[i])
            bfs(i);
    }
    while ( p--)
    {
        scanf ( "%d%d", &x, &y);
        if (con[x]==con[y])
            printf("Yes\n");
        else
            printf("No\n");
    }
    return 0;
}
```

环游大同 80 天 VIJOS_P1107

题目描述：

一年一度的 xuzhenyiOI2006 在风景如画的大同中学举行。按照以往惯例，在激烈的比赛过程后，选手们会应邀去风景名胜区游览。今年当然也不例外。为了确定一条最佳的旅游路线，组委会请各位选手编一程序帮忙寻找。假设所有的风景点都集中在一个 C 列 R 行的矩阵中，矩阵的每一元素代表风景点或者障碍物。现在你要寻找一条满足下列条件的最佳旅游路线：

●这条路线上的每一点结点必须是风景点（即不能为障碍物）

●每个风景点最多游览一次

●这条路线必须是连续的相邻风景点的序列（若风景点 A 和 B 分别位于矩阵的位置（a1，a2）及（b1，b2），且|a1-b1|+|a2-b2|=1，则风景点 A 和 B 是相邻的）

●在满足上述条件下，游览的风景点尽可能多

假设任意的两个风景点都有且仅有一条路径（无回路）相连。显然，任意一个风景点都可以作为游览路线的起点或者终点。

Input Format

第一行是两个整数 C 和 R（3≤C，R≤1000），表示矩阵的列数和行数。

接下来有 R 行，每行有 C 个字符，每个字符都只能是‘#’或‘．’，‘#’表示障碍物，‘．’表示风

景点。行手行末无多余空格。

输出格式 Output Format

只有一行，输出最佳路线的长度。

样例输入 Sample Input

4 4
###.
##..
##.#
##.#

样例输出 Sample Output

0

算法描述

1）需要 2 次 DFS；
2）本题难点在两次 DFS,第一次所选择的起点很可能不是能求得最长路径的那个点，第二次 DFS 时就用这个最远点来作为起点，0ms 通过；
3）BFS 同样可行，思路差不多。

代码

```
#include<stdio.h>
#define N 3000
char map[N][N];
int visited[N][N];
int a[4]={1,0,-1,0};
int b[4]={0,1,0,-1};
int cnt, r, c, maxi, maxj;
void    dfs (int k ,int i, int j)
{
    int h;
    if(k>cnt)
    {
        cnt=k;
        maxi=i;
        maxj=j;
    }
    for( h=0; h<4; h++)
    {
        if( !visited[i+a[h]][j+b[h]]&&map[i+a[h]][j+b[h]]=='.'
            &&i+a[h]>=0&&i+a[h]<r&&j+b[h]>=0&&j+b[h]<c)
            visited[i+a[h]][j+b[h]]=1;
        else
```

```
            continue;
        dfs(k+1,i+a[h],j+b[h]);
        //该处不用恢复 visited 的状态
    }
    return ;
}
int main ()
{
    int   i, j, max, x, y;
    scanf("%d%d", &r, &c);
    getchar();
    for( i=0; i<r; i++)
    {
        gets(map[i]);
        for(j=0; j<c; j++)
            visited[i][j]=0;
    }
    for(i=0,max=0; i<r; i++)
    {
        for( j=0; j<c; j++)
        {
            if(map[i][j]=='.'&&!visited[i][j])
            {
                visited[i][j]=1;
                dfs(0, i, j);//找最远的那个端点
                for(x=0;x<r;x++)//将 visited 恢复到初始状态。
                {
                    for(y=0;y<c;y++)
                        visited[x][y]=0;
                }
                cnt=0;
                visited[maxi][maxj]=1;
                dfs(0, maxi, maxj);//从最远的那个点再次深度搜索
            }
            if(cnt>max)
                max=cnt;
        }
    }
    printf("%d\n", max);
    return 0;
}
```

题目描述：

算法描述

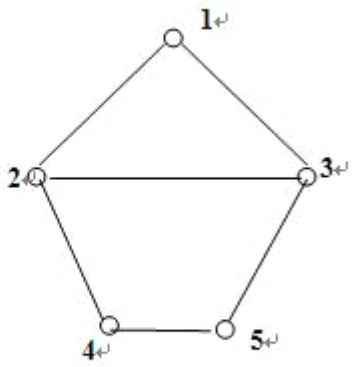
代码

算法名称：图论

Geodetic 集合 BASHU_1427

题目描述：

图 G 是一个无向连通图，没有自环，并且两点之间至多只有一条边。我们定义顶点 v,u 最短路径就是从 v 到 u 经过边最少的路径。所有包含在 v-u 的最短路径上的顶点被称为 v-u 的 Geodetic 顶点，这些顶点的集合记作 I(v, u)。
我们称集合 I(v, u)为一个 Geodetic 集合。
例如下图中，I(2, 5)={2, 3, 4, 5}，I(1, 5)={1, 3, 5}，I(2, 4)={2, 4}。



给定一个图 G 和若干点对 v,u，请你分别求出 I(v, u)。

Input

第一行两个整数 n,m，分别表示图 G 的顶点数和边数（顶点编号 1-n）
下接 m 行，每行两个整数 a,b 表示顶点 a 和 b 之间有一条无向边。
第 m+2 行有一个整数 k，表示给定的点对数。
下接 k 行，每行两个整数 v,u。

Output

共 k 行，每行对应输入文件中每一个点对 v,u，按顶点编号升序输出 I(v, u)。同一行的每个数之间用空格分隔。

Sample Input

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

Sample Output

2 3 4 5
1 3 5
2 4
Hint
100%的数据，n<=40

算法描述

Floyd 算法: 所有顶点对的最短路径,先算出所有顶点对的最短路径，然后再枚举所有经过枚举的点到达终点的距离小于等于最小距离的点

代码

```
#include<stdio.h>
#define N 1000
int main ()
{
    int i, j, n, m, k, v, u, h, edge[41][41];
    scanf("%d%d", &n, &m);
    for ( i=1; i<=n; i++)
    {
        for( j=1; j<=n; j++)
        {
            edge[i][j]=N;
            if(i==j)
```

```
                edge[i][j]=0;
            }
        }
        for( i=0; i<m ;i++)
        {
            scanf ("%d%d", &v, &u);
            edge[v][u]=1;
            edge[u][v]=1;
        }
        for( h=1;h<=n; h++)
        {
            for(i=1;i<=n;i++)
            {
                for( j=1; j<=n; j++)
                {
                    if (edge[i][j]>=(edge[i][h]+edge[h][j]))
                    {
                        edge[i][j]=edge[i][h]+edge[h][j];
                        edge[j][i]=edge[i][j];
                    }
                }
            }
        }
        scanf("%d", &k);
        while(k--)
        {
            scanf ("%d%d", &v, &u);
            for( i=1; i<=n; i++)
            {
                if(edge[v][i]+edge[i][u]<=edge[v][u])
                    printf("%d ", i);
            }
            printf("\n");
        }
        return 0;
}
```

最短网络 BASHU_1411

题目描述：

农民约翰被选为他们镇的镇长！他其中一个竞选承诺就是在镇上建立起互联网，并连接到所

有的农场。当然，他需要你的帮助。约翰已经给他的农场安排了一条高速的网络线路，他想把这条线路共享给其他农场。为了用最小的消费，他想铺设最短的光纤去连接所有的农场。你将得到一份各农场之间连接费用的列表，你必须找出能连接所有农场并所用光纤最短的方案。

Input

该题含有多组测试数据。

第一行为 M 表示有 M 组测试数据。

每组数据第一行为农场的个数，N（3<=N<=100）。

接下去为一个 N*N 的矩阵,表示每个农场之间的距离。(农场之间的距离小于 100000)

Output

每组数据只有一个输出，其中包含连接到每个农场的光纤的最小长度。

Sample Input

```
1
4
0 4 9 21
4 0 8 17
9 8 0 16
21 17 16 0
```

Sample Output

```
28
```

算法描述

算法类似 Dijkstra,单源点最短路径

代码

```
#include<stdio.h>
#define N 100000
int main ()
{
    long int m, n, edge[101][101], min, sum, i, j, v, mark[101], flag, d[101];
    scanf("%ld", &m);
    while(m--)
    {
        scanf("%ld", &n);
        for(i=0;i<n;i++)
        {
            for( j=0; j<n; j++)
                scanf("%ld", &edge[i][j]);
        }
        for(i=0; i<n; i++)
```

```
        {
            d[i]=edge[0][i];//d 记录当前情况下，从入选集合中的点到飞入选集合中的点的
最短距离
            mark[i]=0;
        }
        mark[0]=1;
        flag=0;
        for(i=1, sum=0; i<n; i++)
        {
            for(j=0, min=N; j<n; j++)
            {
                if (!mark[j]&&d[j]<min)
                {
                    min=d[j];
                    v=j;
                }
            }
            mark[v]=1;
            for(j=0; j<n; j++)
            {
                if(!mark[j]&&edge[v][j]<d[j])
                {
                    d[j]=edge[v][j];
                }
            }
            sum+=min;
        }
        printf("%ld\n", sum);
    }
    return 0;
}
```

算法名称：杂题

Herd Sums PKU_2140

题目描述：

The cows in farmer John's herd are numbered and branded with consecutive integers from 1 to N (1 <= N <= 10,000,000). When the cows come to the barn for milking, they always come in sequential order from 1 to N.

Farmer John, who majored in mathematics in college and loves numbers, often looks for patterns. He has noticed that when he has exactly 15 cows in his herd, there are precisely four ways that the numbers on any set of one or more consecutive cows can add up to 15 (the same as the total number of cows). They are: 15, 7+8, 4+5+6, and 1+2+3+4+5.

When the number of cows in the herd is 10, the number of ways he can sum consecutive cows and get 10 drops to 2: namely 1+2+3+4 and 10.

Write a program that will compute the number of ways farmer John can sum the numbers on consecutive cows to equal N. Do not use precomputation to solve this problem.

Input

* Line 1: A single integer: N

Output

* Line 1: A single integer that is the number of ways consecutive cow brands can sum to N.

Sample Input

15

Sample Output

4

算法描述

代码

```
#include<stdio.h>
int main ()
{
	long int i, n, num;
	while ( scanf ( "%d", &n )!= EOF)
	{
		for ( i=1, num=0; i * ( i + 1 ) / 2 <= n; i++ )
		{
			if ( ( n - ( i-1 ) * i / 2) % i == 0 )
				num ++;
		}
		printf( "%d\n", num);
	}
	return 0;
```

}

Moo Volume PKU_2231

题目描述：

Farmer John has received a noise complaint from his neighbor, Farmer Bob, stating that his cows are making too much noise.

FJ's N cows (1 <= N <= 10,000) all graze at various locations on a long one-dimensional pasture. The cows are very chatty animals. Every pair of cows simultaneously carries on a conversation (so every cow is simultaneously MOOing at all of the N-1 other cows). When cow i MOOs at cow j, the volume of this MOO must be equal to the distance between i and j, in order for j to be able to hear the MOO at all. Please help FJ compute the total volume of sound being generated by all N*(N-1) simultaneous MOOing sessions.

Input

* Line 1: N

* Lines 2..N+1: The location of each cow (in the range 0..1,000,000,000).

Output

There are five cows at locations 1, 5, 3, 2, and 4.

Sample Input

5
1
5
3
2
4

Sample Output

40

Hint

INPUT DETAILS:

There are five cows at locations 1, 5, 3, 2, and 4.

OUTPUT DETAILS:

Cow at 1 contributes 1+2+3+4=10, cow at 5 contributes 4+3+2+1=10, cow at 3 contributes 2+1+1+2=6, cow at 2 contributes 1+1+2+3=7, and cow at 4 contributes 3+2+1+1=7. The total volume is (10+10+6+7+7) = 40.

算法描述

代码

```
#include<stdio.h>
#define N 10001
void qsort(__int64 *l,__int64 start,__int64 end)
{
	__int64 i,j,temp1;
	if(start<end)
	{
		i=start,j=end;
		temp1=l[i];
		while(i!=j)
		{
			while(j>i&&l[j]<=temp1) j--;
			if(i<j)
			{
				l[i]=l[j];
				i++;
			}
			while(i<j&&l[i]>=temp1) i++;
			if(i<j)
			{
				l[j]=l[i];
				j--;
			}
		}
		l[i]=temp1;
		qsort(l,start,i-1);
		qsort(l,i+1,end);
	}
}

int main ()
{
	__int64 i, n;
	__int64 x, y, z, num, k,a[N];
```

```
    scanf ("%I64d", &n);
    for ( i=1; i<=n; i++)
    {
        scanf ("%I64d",&a[i]);
    }
    if (n==1)
    {
        printf("0\n");
        return 0;
    }
    qsort(a, 1, n);
    for ( i=2,x=0,k=0; i<=n; i++)
    {
        k +=a[i-1];
        x +=k;
    }
    for ( i=1,y=0; i<=n; i++)
    {
        y+=(n-2*i+1)*a[i];
    }
    for ( i=2,z=0,k=0; i<=n; i++)
    {
        k+=a[i];
    }
    for ( i=2,z=0; i<=n; i++)
    {
        z+=k;
        k-=a[i];
    }
    num = x+y-z;
    printf("%I64d\n", num);
    return 0;
}
```

Lotto

题目描述：

In the German Lotto you have to select 6 numbers from the set {1,2,...,49}. A popular strategy to play Lotto - although it doesn't increase your chance of winning - is to select a subset S containing k (k > 6) of these 49 numbers, and then play several games with choosing numbers only from S. For example, for k=8 and S = {1,2,3,5,8,13,21,34} there are 28 possible games: [1,2,3,5,8,13],

[1,2,3,5,8,21], [1,2,3,5,8,34], [1,2,3,5,13,21], ... [3,5,8,13,21,34].

Your job is to write a program that reads in the number k and the set S and then prints all possible games choosing numbers only from S.

Input

The input will contain one or more test cases. Each test case consists of one line containing several integers separated from each other by spaces. The first integer on the line will be the number k ($6 < k < 13$). Then k integers, specifying the set S, will follow in ascending order. Input will be terminated by a value of zero (0) for k.

Output

For each test case, print all possible games, each game on one line. The numbers of each game have to be sorted in ascending order and separated from each other by exactly one space. The games themselves have to be sorted lexicographically, that means sorted by the lowest number first, then by the second lowest and so on, as demonstrated in the sample output below. The test cases have to be separated from each other by exactly one blank line. Do not put a blank line after the last test case.

Sample Input

```
7 1 2 3 4 5 6 7
8 1 2 3 5 8 13 21 34
0
```

Sample Output

```
1 2 3 4 5 6
1 2 3 4 5 7
1 2 3 4 6 7
1 2 3 5 6 7
1 2 4 5 6 7
1 3 4 5 6 7
2 3 4 5 6 7

1 2 3 5 8 13
1 2 3 5 8 21
1 2 3 5 8 34
1 2 3 5 13 21
1 2 3 5 13 34
1 2 3 5 21 34
1 2 3 8 13 21
1 2 3 8 13 34
1 2 3 8 21 34
1 2 3 13 21 34
1 2 5 8 13 21
```

1 2 5 8 13 34
1 2 5 8 21 34
1 2 5 13 21 34
1 2 8 13 21 34
1 3 5 8 13 21
1 3 5 8 13 34
1 3 5 8 21 34
1 3 5 13 21 34
1 3 8 13 21 34
1 5 8 13 21 34
2 3 5 8 13 21
2 3 5 8 13 34
2 3 5 8 21 34
2 3 5 13 21 34
2 3 8 13 21 34
2 5 8 13 21 34
3 5 8 13 21 34

算法描述

穷举

代码

```
#include<stdio.h>
#define N 50
int main ()
{
    int set[N], a[N], i, k=6, n, position,w=0;//set 放 k 个元素，a 放所有元素
    while ( scanf( "%d", &n) != EOF )
    {
        if (n==0) break;
        for ( i=0; i<n; i++ )
            scanf ( "%d", &a[i] );
        for ( i=0; i<k; i++ )
            set[i] = a[i];
        position = k-1;//给 position 定位在 set 的最后一位
        w= position;
        for ( i=0; i<k; i++ )//将 a 中前 k 个元素输出
            printf( "%d ", set[i] );
        printf ( "\n" );
        while (1)//组合
```

```
        {

            if ( set[k-1] == a[n-1] )//position 表示要增值的起始位
            {
                position--;
                w=position;
            }
            else
                position = k-1;
            while(a[w]<=set[position])
                w++;
            if ( w<n )
            {
                for ( i=position; i<k; i++, w++)//position 右边的各位需要依次增加
                    set[i] = a[w];
                w--;
            }
            else w=position;
            for ( i=0; i<k; i++ )
                printf ( "%d ", set[i] );
            printf ( "\n" );
            if ( set[0] >= a[n-k] )
                break;
        }
        printf( "\n" );
    }
    return 0;
}
```

Euclid's Game

题目描述：

Two players, Stan and Ollie, play, starting with two natural numbers. Stan, the first player, subtracts any positive multiple of the lesser of the two numbers from the greater of the two numbers, provided that the resulting number must be nonnegative. Then Ollie, the second player, does the same with the two resulting numbers, then Stan, etc., alternately, until one player is able to subtract a multiple of the lesser number from the greater to reach 0, and thereby wins. For example, the players may start with (25,7):

25 7

11 7

4 7

4 3

1 3

1 0

an Stan wins.

Input

The input consists of a number of lines. Each line contains two positive integers giving the starting two numbers of the game. Stan always starts.

Output

For each line of input, output one line saying either Stan wins or Ollie wins assuming that both of them play perfectly. The last line of input contains two zeroes and should not be processed.

Sample Input

```
34 12
15 24
0 0
```

Sample Output

```
Stan wins
Ollie wins
```

算法描述 1：

博弈：27　5
可以 27-5*1,27-5*2,27-5*3,27-5*4,27-5*5;
27-5*i,i<5 时，下一次仍然只能取 5
因此，可以看作是有一堆个数为[27/5]=5 的石子，
两人轮流取完这堆这后,又有一堆数目为[5/2]=2 的石子，
接下来就是一堆数目为[2/1]=2 的石子,然后就取完了.
现在，问题就变成 ,已知大小的 N 堆石子，依次取完每堆中的石子,个数不限(但只能在一堆中取)，取得最后一堆的人胜.
如果第一堆中石子的个数大于 1,则先取者必胜，因为,如果取完第一堆的人有必胜法，则他可以取直接取完第一堆，如果取完第一堆者必败，则他可以让第一堆只剩一个子，这样，败局就落到对手的头上.
由此推出，谁能够首先碰到一堆数目大于 1 的(前几堆都只有一个)，谁就胜.
就是这样了.

代码

```
#include<stdio.h>
int main ()
{
	int a, b, t, flag;
	while (scanf ( "%d%d", &a, &b)!=EOF)
	{
		if(a==0&&b==0)
			break;
		if (a<b)
		{
			t=a;
			a=b;
			b=t;
		}
		if(a==b||b==1)
		{
			printf("Stan wins\n");
			continue;
		}
		flag=1;
		while ( a/b<=1 )
		{
			flag=(flag+1)%2;
			t=a;
			a=b;
			b=t%b;
		}
		if (flag==1)
			printf("Stan wins\n");
		else
			printf("Ollie wins\n");
	}
	return 0;
}
```

算法描述 2：

黄金分割判断
把一条线段分割为两部分，使其中一部分与全长之比等于另一部分与这部分之比。其比值是一个无理数，取其前三位数字的近似值是 0.618。由于按此比例设计的造型十分美丽，因此称为黄金分割，也称为中外比。这是一个十分有趣的数字，我们以 0.618 来近似，通过简单

的计算就可以发现：
1/0.618=1.618
(1-0.618)/0.618=0.618

代码

```
#include<stdio.h>
int main ()
{
	double a, b, t;
	while (scanf ("%lf%lf", &a, &b)!=EOF )
	{
		if (a==0 &&b==0)
			break;
		 if (a<b)
		 {
			t=a;
			a=b;
			b=t;
		 }
		 if ( a==b || b==1 || a * 0.618 > b)
			printf("Stan wins\n");
		 else
			printf("Ollie wins\n");
	}
	return 0;
}
```

Japan

题目描述：

Japan plans to welcome the ACM ICPC World Finals and a lot of roads must be built for the venue. Japan is tall island with N cities on the East coast and M cities on the West coast (M <= 1000, N <= 1000). K superhighways will be build. Cities on each coast are numbered 1, 2, ... from North to South. Each superhighway is straight line and connects city on the East coast with city of the West coast. The funding for the construction is guaranteed by ACM. A major portion of the sum is determined by the number of crossings between superhighways. At most two superhighways cross at one location. Write a program that calculates the number of the crossings between superhighways.

Input

The input file starts with T - the number of test cases. Each test case starts with three numbers – N, M, K. Each of the next K lines contains two numbers – the numbers of cities connected by the superhighway. The first one is the number of the city on the East coast and second one is the number of the city of the West coast.

Output

For each test case write one line on the standard output:
Test case (case number): (number of crossings)

Sample Input

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

Sample Output

Test case 1: 5

算法描述

判断有多少个交叉点的好方法，搜会超时

代码

```
#include <stdio.h>
#include <string.h>
#define    N 1001
long a[N][N];
long main ()
{
      long t,n, m, k, s=0, i, j, p[N], x, y,cnt;
      __int64 sum;
      scanf ( "%ld", &t );
      while ( t-- )
      {
            s++;
            scanf ( "%ld%ld%ld", &n, &m, &k);
            memset(a,0,sizeof(a));
```

```
            memset(p,0,sizeof(p));
            for ( i=0; i<k; i++)
            {
                scanf ( "%ld%ld", &x, &y);
                a[x][y]=1;
            }
            for (sum=0, i=1; i<=n; i++)
            {
                for (cnt=0, j=1; j<=m; j++)
                {
                    sum += cnt * p[j];
                    if (a[i][j])
                    {
                        cnt++;
                        p[j]++;
                    }
                }
            }
            printf( "Test case %ld: %I64d\n", s, sum);
        }
        return 0;
}
```

新年趣事之红包

题目描述：

xiaomengxian 一进门，发现外公、外婆、叔叔、阿姨……都坐在客厅里等着他呢。经过仔细观察，xiaomengxian 发现他们所有人正好组成了一个凸多边形。最重要的是，他们每个人手里都拿着一个红包（^o^）。于是非常心急，xiaomengxian 决定找一条最短的路线，拿到所有的红包。

假设屋里共有 N 个人拿着红包，把他们分别从 1 到 N 编号。其中，编号为 1 的人就坐在大门口，xiaomengxian 必须从这里出发去拿其它的红包。一条合法的路线必须经过所有的点一次且仅一次。

Input

第一行为一个整数 N（1<=N<=800）。

以下 N 行，每行两个实数 Xi，Yi，表示该点的坐标。

各个点按照逆时针顺序依次给出。

Output

一个实数，表示最短的路线长度（保留三位小数）。

Sample Input

4
50.0 1.0
5.0 1.0
0.0 0.0
45.0 0.0

Sample Output

50.211

算法描述

计算几何的凸包问题
周长减去最长边

代码

```
#include<stdio.h>
#include<math.h>
double dis ( double x1, double y1, double x, double y )
{
	return ( sqrt((x1-x)*(x1-x) + (y1-y)*(y1-y)));
}
int main ()
{
	int n, i;
	double x1,y1, x, y, d, sum, max, x2,y2;
	scanf ("%d", &n);
	scanf ("%lf%lf", &x1, &y1);
	x2=x1;
	y2=y1;
	for ( i=0, max=0, sum=0; i<n-1;i++)
	{
		scanf ( "%lf%lf", &x, &y);
		d=dis(x,y,x1,y1);
		x1=x;
		y1=y;
		sum += d;
		if ( d>max)
			max=d;
	}
	d=dis(x,y,x2,y2);
```

```
        sum += d;
        if ( d>max)
                max=d;
        printf("%.3lf", sum-max);
        return 0;
}
```

弱弱的战壕

题目描述：

永恒和 mx 正在玩一个即时战略游戏，名字嘛~~~~~~恕本人记性不好，忘了-_-b。

mx 在他的基地附近建立了 n 个战壕，每个战壕都是一个独立的作战单位，射程可以达到无限（“mx 不赢定了？！？”永恒 ftING...@_@）。

但是，战壕有一个弱点，就是只能攻击它的左下方，说白了就是横纵坐标都不大于它的点（mx：“我的战壕为什么这么菜”ToT）。这样，永恒就可以从别的地方进攻摧毁战壕，从而消灭 mx 的部队。

战壕都有一个保护范围，同它的攻击范围一样，它可以保护处在它左下方的战壕。所有处于它保护范围的战壕都叫做它的保护对象。这样，永恒就必须找到 mx 的战壕中保护对象最多的点，从而优先消灭它。

现在，由于永恒没有时间来计算，所以拜托你来完成这个任务：

给出这 n 个战壕的坐标 xi、yi，要你求出保护对象个数为 0，1，2……n-1 的战壕的个数。

Input

第一行，一个正整数 n（1<=n<=15000）
接下来 n 行，每行两个数 xi,yi，代表第 i 个点的坐标
（1<=xi,yi<=32000）
注意：可能包含多重战壕的情况（即有数个点在同一坐标）

Output

输出 n 行，分别代表保护对象为 0，1，2……n-1 的战壕的个数。

Sample Input

```
5
1 1
5 1
7 1
```

3 3
5 5

Sample Output

1
2
1
1
0

算法描述

特殊之处：处理用一坐标上多个点的情况

代码

```
#include<stdio.h>
#define N 15000
int x[N], y[N];
void qsort ( int *a, int *b, int start, int end)
{
	int i, j, temp1, temp2;
	if ( start<end)
	{
		i=start;
		j=end;
		temp1=a[i];
		temp2=b[i];
		while ( i!=j)
		{
			while ( i<j && a[j]>=temp1 )	j--;
			if ( i<j)
			{
				a[i]=a[j];
				b[i]=b[j];
				i++;
			}
			while ( i<j && a[i]<=temp1 ) i++;
			if ( i<j)
			{
				a[j]=a[i];
				b[j]=b[i];
```

```
                j--;
            }
        }
        a[i]=temp1;
        b[i]=temp2;
        qsort(a,b,start,i-1);
        qsort(a,b,i+1,end);
    }
}
int main ()
{
    int n, cnt, i, j, num[N], s;
    scanf ( "%d", &n);
    for ( i=0; i<n; i++)
    {
        scanf ("%d%d", &x[i], &y[i]);
        num[i]=0;
    }
    qsort(x, y,  0, n-1);
    for ( i=0; i<n; i++)
    {
        s=i;
        while ( x[i+1]==x[i] && i<n-1 )  i++;
        qsort ( y,x,s,i);
    }
    for (i=0; i<n; i++)
    {
        s=i;
        while (i < n-1&&x[i]==x[i+1] && y[i] == y[i+1] ) i++;
        for ( cnt=0,j=0; j<i; j++)
        {
            if ( y[j]<=y[i])
                cnt++;
        }
        num[cnt]+=i-s+1;
    }
    for ( i=0; i<n; i++)
        printf("%d\n", num[i]);
    return 0;
}
```

序列 BASHU_1425

题目描述：

有一个非递减的整数序列 S1，S2，S3，……，Sn+1（Si<=Si+1）。定义序列 m1，m2，…，mn¬为 S 的“M 序列”，其中 mi=(Si+Si+1)/2。
例如，S=(1, 3, 3, 5)，则 m=(2, 3, 4)。
现在给你序列 m，要你求有多少个 S 序列的“M 序列”是序列 m。

Input

第一行一个整数 n，
下接 n 行，每行一个整数 mi

Output

一个整数，表示有多少个 S 序列的“M 序列”是序列 m

Sample Input

3
2
5
9

Sample Output

4

Hint

样例说明：存在如下四个数列 S 满足要求：
2，2，8，10；
1，3，7，11；
0，4，6，12；
-1，5，5，13。
数据范围
50%的数据 n<=1000，mi<=20000
100%的数据 2<=n<=100000，mi<=10^9

算法描述

找规律：纵向成递增或递减

代码

```
#include<stdio.h>
#define N 100001
int main ()
{
	__int64 m[N], i, n,pre, k, max;
	scanf ("%I64d", &n);
	if(n==0||n==1)
		return 0;
	for( i=0; i<n; i++)
		scanf("%I64d", &m[i]);
	pre=m[0]-m[0]*2+m[1]+1;
	max=m[0];
	for ( i=1; i<n; i++)
	{
		max=m[i-1]*2-max;
		k=max;
		if( pre+max-1 <m[i])
			max=pre+max-1;
		else
			max=m[i];
		pre=max-k+1;
	}
	if(pre<0)
		printf("0\n");
	else
		printf("%I64d\n", pre);
	return 0;
}
```

回文数 bashu_1085

题目描述：

一个自然数如果把所有数字倒过来以后和原来的一样，那么我们称它为回文数。例如 151 和 753357。我们可以把所有回文数从小到大排成一排：1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 22, 33, ...注意 10 不是回文数，虽然我们可以把它写成 010，但是在本题中前导 0 是不允许的。
你的任务是求出第 i 小的回文数。例如第 1,12,24 大的回文数分别是 1,33,151。

Input

输入只有一行，即 i(1<=i<=2*10^9)。

Output

输出只有一行，即第 i 小的回文数。

Sample Input

24

Sample Output

151

算法描述

代码

```
#include <stdio.h>
#include <string.h>
#define N 20
__int64 a[N], c[N];
int b[N];
int flag;
__int64 pow10 ( int n )
{
	__int64 sum = 1;
	int i;
	for (i = 0; i < n; i++)
		sum *= 10;
	return sum;
}
void init()
{
	int i;
	memset(a, 0, sizeof(a));
	memset(c, 0, sizeof(c));
	for (i = 1; i < N; i++)
	{
		a[i] += a[i - 1] + 9 * pow10((i - 1)/ 2);
		c[i] = 9 * pow10(i - 1);
	}
}
void f(int t,int flag2)
{
```

```
    int i, tt, j, k;
    tt = flag2;
    if (t == 1)
    {
        b[1] = flag2;
    }
    else
    {
        tt = flag2 - 1;
        k = t;
        while (tt > 0)
        {
            b[k] = tt % 10;
            tt = tt / 10;
            k--;
        }
        b[1]++;
    }
}
int main()
{
    __int64 n;
    int i, flag2, t, t2 ;
    init();
    for ( i=1; i<N; i++)
        printf( "a %I64d\n", a[i]);
    for ( i=1; i<N; i++)
        printf( "c %I64d\n", c[i]);
    while (scanf("%I64d", &n) != EOF)
    {
        memset(b, 0, sizeof(b));
        for (i = 0; i < N - 1; i++)
        {
            if (a[i] < n && a[i + 1] >= n)
            {
                flag = i + 1;
                break;
            }
        }
        flag2 = n - a[flag - 1];
        t2 = (flag + 1) / 2;
        f(t2, flag2);
        if (flag == 1)
            printf("%d\n", b[flag]);
```

```
            else
            {
                if (flag % 2)
                {
                    for (i = 1; i <= t2; i++)
                        printf("%d", b[i]);
                    for (i = t2 - 1; i > 1 ; i--)
                        printf("%d", b[i]);
                    printf("%d\n", b[1]);
                }
                else
                {
                    for (i = 1; i <= t2; i++)
                        printf("%d", b[i]);
                    for (i = t2; i > 1 ; i--)
                        printf("%d", b[i]);
                    printf("%d\n", b[1]);
                }
            }
        }
        return 0;
}
```

Subsequence PKU_3302

题目描述：

Given a string s of length n, a subsequence of it, is defined as another string $s' = s_{u1}s_{u2}...s_{um}$ where $1 \le u_1 < u_2 < ... < u_m \le n$ and s_i is the ith character of s. Your task is to write a program that, given two strings *s1* and *s2*, checks whether either *s2* or its reverse is a subsequence of *s1* or not.

Input

The first line of input contains an integer T, which is the number of test cases. Each of the next T lines contains two non-empty strings *s1* and *s2* (with length at most 100) consisted of only alpha-numeric characters and separated from each other by a single space.

Output

For each test case, your program must output "YES", in a single line, if either *s2* or its reverse is a subsequence of *s1*. Otherwise your program should write "NO".

Sample Input

5

arash aah
arash hsr
kick kkc
A a
a12340b b31

Sample Output

YES
YES
NO
NO
YES

算法描述

代码

```
#include<stdio.h>
#include<string.h>
#define N 1000
int main ()
{
	int t, cnt, na, nb;
	char a[N], b[N], *p, *q;
	scanf ("%d", &t);
	while (t--)
	{
		scanf ("%s %s", a, b);
		p=a, q=b;
		na = strlen (a);
		nb = strlen (b);
		cnt=0;
		while ( p<a+na && q<b+nb )
		{
			if ( *p == *q )
			{
				cnt ++;
				p++;
				q++;
			}
			else
			{
```

```
                p++;
            }
        }
        if (cnt==nb)
        {
            printf( "YES\n");
        }
        else
        {
            q=b+nb-1;
            cnt=0;p=a;//注意对指针归位
            while ( p<a+na && q>=b    )
            {
                if ( *p == *q )
                {
                    cnt ++;
                    p++;
                    q--;
                }
                else
                {
                    p++;
                }
            }
            if ( cnt==nb)
            {
                printf( "YES\n");
            }
            else
                printf("NO\n");
        }
    }
    return 0;
}
```

Barbara Bennett's Wild Numbers PKU_3340

题目描述：

A *wild number* is a string containing digits and question marks (like 36?1?8). A number X matches a wild number W if they have the same length, and every non-question mark character in X is equal to the character in the same position in W (it means that you can replace a question mark with any

digit). For example, 365198 matches the wild number 36?1?8, but 360199, 361028, or 36128 does not. Write a program that reads a wild number *W* and a number *X* from input, both of length *n*, and determines the number of *n*-digit numbers that match *W* and are greater than *X*.

Input

There are multiple test cases in the input. Each test case consists of two lines of the same length. The first line contains a wild number *W*, and the second line contains an integer number *X*. The length of input lines is between 1 and 10 characters. The last line of input contains a single character #.

Output

For each test case, write a single line containing the number of *n*-digit numbers matching *W* and greater than *X* (*n* is the length of *W* and *X*).

Sample Input

```
36?1?8
236428
8?3
910
?
5
#
```

Sample Output

```
100
0
4
```

算法描述

代码

```
#include<stdio.h>
#include<string.h>
#define N 10
int main()
{
	char a[N], b[N];
	int i, j, n, num, w, x;
	while(scanf("%s", a)!=NULL)
	{
		if(a[0]=='#')
			break;
```

```
        scanf("%s", b);
        n=strlen(a);
        num=0;
        for( i=0; i<n; i++)
        {
            if(a[i]=='?')
                num++;
        }
        x=0;
        for(i=0; i<n; i++)
        {
            while( a[i]==b[i]&&a[i]!='\0')
                i++;
            if(a[i]=='?')
            {
                w=1;
                num--;
                for(j=0; j <num; j++)
                    w *= 10;
                x += w*(9-(b[i]-'0'));
            }
            else if(a[i]>b[i])
            {
                w=1;
                for( j=0; j<num; j++)
                    w*=10;
                x+=w;
                break;
            }
            else if(a[i]<b[i])
                break;
        }
        printf("%d\n", x);
    }
    return 0;
}
```

题目描述：

算法描述

代码

数据结构

快速排序

整数快排

代码：

```
#include<stdio.h>
#define N 100

void qsort(int *l,int start,int end)
{
	int i,j,temp1;
	if(start<end)
	{
		i=start,j=end;
		temp1=l[i];
		while(i!=j)
		{
			while(j>i&&l[j]>=temp1) j--;
			if(i<j)
			{
				l[i]=l[j];
				i++;
			}
			while(i<j&&l[i]<=temp1) i++;
			if(i<j)
			{
				l[j]=l[i];
				j--;
			}
```

```
            }
            l[i]=temp1;
            qsort(l,start,i-1);
            qsort(l,i+1,end);
        }
}
int main()
{
        int l[N], i, n;
        scanf ("%d", &n);
        for (i=0; i<n ;i++)
        {
            scanf ("%d", &l[i]);
        }
        qsort( l, 0, n-1);
        for (i=0; i<n ;i++)
            printf ("%d ", l[i]);
        printf("\n");
        return 0;
}
```

字符串快排

代码

```
#include<stdio.h>
#include<string.h>
#define N 100
void qsort( char l[N][N], int start, int end )
{
        int i,j;
        char temp1[N];
        if( start < end )
        {
            i=start,j=end;
            strcpy(temp1,l[i]);
            while(i!=j)
            {
                while(j>i&&strcmp(l[j],temp1)>=0) j--;
                if(i<j)
                {
                    strcpy(l[i],l[j]);
                    i++;
```

```
                }
                while(i<j&&strcmp(l[i],temp1)<=0) i++;
                if(i<j)
                {
                    strcpy(l[j],l[i]);
                    j--;
                }
            }
            strcpy(l[i],temp1);
            qsort(l,start,i-1);
            qsort(l,i+1,end);
        }
}
int main ()
{
    int i, n;
    char a[N][N];
    while ( scanf ( "%d", &n) !=EOF )
    {
        for ( i=0; i<n; i++ )
        {
            gets (a[i]);
        }
        qsort ( a, 0, n-1);
        for ( i=0; i<n; i++ )
        {
            puts (a[i]);
        }
    }
    return 0;
}
```

希尔排序

代码

```
void ShellPass( SeqList R，int d)
{//希尔排序中的一趟排序，d 为当前增量
    for(i=d+1;i<=n；i++) //将 R[d+1. . n]分别插入各组当前的有序区
        if(R[i].key<R[i-d].key){
            R[0]=R[i];j=i-d； //R[0]只是暂存单元，不是哨兵
                do {//查找 R[i]的插入位置
                    R[j+d]=R[j]； //后移记录
```

```
                    j=j-d;  //查找前一记录
                }while(j>0&&R[0].key<R[j].key);
                R[j+d]=R[0];  //插入 R[i]到正确的位置上
        }
}

void  ShellSort(SeqList R)
{
    int increment=n;  //增量初值，不妨设 n>0
        do {
            increment=increment/3+1;  //求下一增量
                ShellPass(R，increment);  //一趟增量为 increment 的 Shell 插入排序
        }while(increment>1)
} //ShellSort
  /*  注意：
      当增量 d=1 时，ShellPass 和 InsertSort 基本一致，只是由于没有哨兵而在内循环中增加了一个循环判定条件"j>0"，以防下标越界。*/
```

归并排序

数据结构分析

1、稳定性:归并排序是一种稳定的排序。

2、存储结构要求:可用顺序存储结构。也易于在链表上实现。

3、时间复杂度:对长度为 n 的文件，需进行 趟二路归并，每趟归并的时间为 O(n)，故其时间复杂度无论是在最好情况下还是在最坏情况下均是 O(nlgn)。

4、空间复杂度:需要一个辅助向量来暂存两有序子文件归并的结果，故其辅助空间复杂度为 O(n)，显然它不是就地排序。

代码

```
#include<stdio.h>
#define N 1000
int n;
void merge ( int *a, int *b, int x, int y, int z )//一次归并
{
    int i, j, k, t;
    i=x;
    j=y+1;
```

```
    k=x;
    while ( i <= y && j <= z )
    {
        if ( a[i] <= a[j] )
        {
            b[k] = a[i];
            i++;
        }
        else
        {
            b[k] = a[j];
            j++;
        }
        k++;
    }
    if ( i <= y )
    {
        for ( t=i; t<=y; t++ )
            b[k+t-i] = a[t];
    }
    else
    {
        for ( t=j; t<=z; t++ )
            b[k+t-j] = a[t];
    }
}

void mergepass ( int *a,    int *b, int len )
{
    int i, j;
    i=1;
    while ( i <= n-2*len+1 )
    {
        merge ( a, b, i, i+len-1, i+2*len-1 );//一次归并
        i = i + 2 * len;//置于下一个一次归并的起始位置
    }
    if ( i + len -1 < n )//对剩下的 1 个长为 len，另一个长度不足 len，终点为 n 的两个有序段
归并
        merge ( a, b, i, i+len-1, n );
    else
        for ( j = i; j <= n; j++ )
            b[j] = a[j];
}
```

```
void mergesort ( int *a )
{
    int len;
    int temp[2*N];
    len = 1;
    while ( len < n )
    {
        mergepass ( a,   temp, len );
        len = 2 * len;
        mergepass ( temp, a, len );
        len = 2 * len;
    }
}

int main ()
{
    int i, a[N];
    scanf ( "%d", &n );
    for ( i=1; i<=n ; i++)
        scanf ( "%d", &a[i] );
    mergesort (a);
    for ( i=1; i<=n ; i++ )
        printf ( "%d ", a[i] );
    printf( "\n");
    return 0;
}
```

堆排序

数据结构分析

(1)堆排序的时间，主要由建立初始堆和反复重建堆这两部分的时间开销构成，它们均是通过调用 Heapify 实现的。

(2) 堆排序的最坏时间复杂度为 O(nlgn)。堆排序的平均性能较接近于最坏性能。

(3)由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的文件。

(4)堆排序是就地排序，辅助空间为 O(1)

(5)它是不稳定的排序方法。

代码

```
#include<stdio.h>
#define N 1000
```

```
//Get the max heap
int out[N];
void sift( int *a, int n )//筛选算法
{
	int i, j, k;
	for ( i = (n/2-1); i >= 0; i-- )
	{
		k = i;
		j = 2*k + 1;
		int temp = *(a+k);
		int flag = 0;
		while ( !flag && j<n )
		{
			if ( *(a+j) < *(a+j+1) )
				j++;
			if ( temp < *(a+j) )
			{
				*(a+k) = *(a+j);
				k = j;
			}
			else
				flag	=1;
			*(a+k) = temp;
		}
	}
}
void heapsort ( int *a, int n )
{
	int i , j=0, temp;
	sift ( a, n );
	for ( i=n-1; i>1; i-- )
	{
		out[j++] = *a;
		temp = *a;//swap
		*a = *(a+i);
		*(a+i) = temp;
		sift ( a, i-1 );
	}
	out[j] = a[0] > a[1] ? a[0] : a[1];
	out[j+1] = a[0] < a[1] ? a[0] : a[1];
}
int main()
{
	int table[N], i, n;
```

```
    scanf ( "%d", &n);
    for ( i = 0; i < n; i++ )
        scanf("%d", &table[i]);
    heapsort ( table, n );
    printf("table :\n");
    for ( i=0; i<n; i++ )
        printf("%d ", table[i]);//从小到大的排序
    printf("\n");
    printf("out :\n");
    for ( i=0; i<n; i++ )
        printf("%d ", out[i]);//从大到小的排序
    printf("\n");
    return 0;
}
```