# 实验一

### Problem A

### 二分查找

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

给定一个单调递增的整数序列，问某个整数是否在序列中。

输入：

第一行为一个整数n，表示序列中整数的个数；第二行为n（n不超过10000）个整数；第三行为一个整数m（m不超过50000），表示查询的个数；接下来m行每行一个整数k。

输出：

每个查询的输出占一行，如果k在序列中，输出Yes，否则输出No。

输入样例：

5  
1 3 4 7 11  
3  
3  
6  
9

输出样例：

Yes  
No  
No

来源：

#include<stdio.h>

#include<stdlib.h>

void binarysearch(int n,int k,int a[])

{

int left=0;

int right=n-1;

int r=0;

while(left<=right)

{

int middle=(left+right)/2;

if(k==a[middle]) r=1;

if(k>a[middle]) left=middle+1;

else right=middle-1;

}

if(r==1) printf("Yes\n");

else printf("No\n");

}

int main()

{

int i,n,m,a[10000],b[50000];

scanf("%d",&n);

for(i=0;i<n;i++)

scanf("%d",&a[i]);

scanf("%d",&m);

for(i=0;i<m;i++)

scanf("%d",&b[i]);

for(i=0;i<m;i++)

binarysearch(n,b[i],a);

return 0;

}

### Problem B

### 归并排序

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

给定一个数列，用归并排序算法把它排成升序。

输入：

第一行是一个整数n（n不大于10000），表示要排序的数的个数；  
下面一行是用空格隔开的n个整数。

输出：

输出排序后的数列，每个数字占一行。

输入样例：

5  
3 2 1 4 5

输出样例：

1  
2  
3  
4  
5

#include<stdio.h>

// 一个递归函数

void mergesort(int \*num,int start,int end);

// 这个函数用来将两个排好序的数组进行合并

void merge(int \*num,int start,int middle,int end);int main()

{

// 测试数组

int num[10000];

int i,n;

// 排序之前

scanf("%d",&n);

for (i = 0; i < n; i++) {

scanf("%d",&num[i]);

}

// 进行合并排序

mergesort(num,0,n-1);

// 排序之后

for (i=0; i<n; i++)

{

printf("%d\n",num[i]);

}

return 0;}

//这个函数用来将问题细分

void mergesort(int \*num,int start,int end){

int middle;

if(start<end)

{

middle=(start+end)/2;

// 归并的基本思想

// 排左边

mergesort(num,start,middle);

// 排右边

mergesort(num,middle+1,end);

// 合并

merge(num,start,middle,end);

}

}

//这个函数用于将两个已排好序的子序列合并

void merge(int \*num,int start,int middle,int end)

{

int n1=middle-start+1;

int n2=end-middle;

// 动态分配内存，声明两个数组容纳左右两边的数组

int \*L=new int[n1+1];

int \*R=new int[n2+1];

int i,j=0,k;

//将新建的两个数组赋值

for (i=0; i<n1; i++)

{

\*(L+i)=\*(num+start+i);

}

// 哨兵元素

\*(L+n1)=1000000;

for (i=0; i<n2; i++)

{

\*(R+i)=\*(num+middle+i+1);

}

\*(R+n2)=1000000;

i=0;

// 进行合并

for (k=start; k<=end; k++)

{

if(L[i]<=R[j])

{

num[k]=L[i];

i++;

}

else

{

num[k]=R[j];

j++;

}

}

delete [] L;

delete [] R;

}

### Problem C

### 快速排序

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

给定一个数列，用快速排序算法把它排成升序。

输入：

第一行是一个整数n，表示要排序的数的个数；下面一行是用空格隔开的n个整数。

输出：

输出排序后的数列，每个数字占一行。

输入样例：

5  
3 2 1 4 5

输出样例：

1  
2  
3  
4  
5

#include<stdio.h>

#include<stdlib.h>

void QuickSort(int A[],int n,int left,int right)

{

int i,j,t;

if(left<right){

i=left;

j=right+1;

while(1){

while(i+1<n&&A[++i]<A[left]);

while(j-1>-1&&A[--j]>A[left]);

if(i>=j) break;

t=A[i];

A[i]=A[j];

A[j]=t;

}

t=A[left],A[left]=A[j],A[j]=t;

QuickSort(A,n,left,j-1);

QuickSort(A,n,j+1,right);

}

}

int main()

{

int i,n,A[10000];

scanf("%d",&n);

for(i=0;i<n;i++)

scanf("%d",&A[i]);

QuickSort(A,n,0,n-1);

for(i=0;i<n;i++)

printf("%d\n",A[i]);

return 0;

}

### Problem D

### 走迷宫

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

判断是否能从迷宫的入口到达出口

输入：

先输入两个整数表示迷宫的行数m和列数n，再输入口和出口的坐标，最后分m行输入迷宫，其中1表示墙，0表示空格每个数字之间都有空格。

输出：

若能到达，则输出"Yes"，否则输出"No"，结果占一行。

输入样例：

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

输出样例：

Yes

#include <stdio.h>

#include <stdlib.h>

typedef struct MNode{

int row;

int col;

struct MNode \* next;

} Node,\* PNode;

PNode head = NULL,tail = NULL;

int dir[4][2] = {{-1,0},{0,1},{1,0},{0,-1}};

int maxRow ,maxCol ;

int i,j,inrow ,incol ,outrow ,outcol ;

int \*\*mmap;

void removeItem(){

PNode tmp = head;

head = head->next;

free(tmp);

}

int haveItem(){

if(head==NULL) return 0;

return 1;

}

void addItem(int row,int col){

PNode newPoint = (PNode)malloc(sizeof(Node));

newPoint->row = row;

newPoint->col = col;

newPoint->next = NULL;

tail->next = newPoint;

tail = newPoint;

}

int isCan(int row,int col)

{

if(row<0||col<0||row>=maxRow||col>=maxCol) return 0;

if(mmap[row][col]!=0) return 0;

return 1;

}

int main()

{

scanf("%d%d%d%d%d%d",&maxRow,&maxCol,&inrow,&incol,&outrow,&outcol);

mmap = (int \*\*)malloc(sizeof(int\*)\*maxRow);

for(i = 0;i<maxRow;i++){

mmap[i] = (int\*)malloc(sizeof(int)\*maxCol);

}

for(i = 0;i<maxRow;i++)

for(j = 0;j<maxCol;j++){

scanf("%d",&mmap[i][j]);

}

head = (PNode)malloc(sizeof(Node));

head->row = inrow;

head->col = incol;

head->next = NULL;

tail = head;

for(;haveItem();){

if(head->row==outrow&&head->col == outcol){

printf("Yes\n");

for(;haveItem();){

removeItem();//free more

}

return 0;

}

for(i = 0;i<4;i++){

if(isCan(head->row+dir[i][0],head->col+dir[i][1])){

addItem(head->row+dir[i][0],head->col+dir[i][1]);

}

}

mmap[head->row][head->col] = 2;

removeItem();

}

printf("No\n");

return 0;

}

### Problem E

### 穷举n位二进制数

#### 时限：100ms 内存限制：10000K 总时限：300ms

描述：

输入一个小于20的正整数n，要求按从小到大的顺序输出所有的n位二进制数，每个数占一行。

输入：

输入一个小于20的正整数n。

输出：

按从小到大的顺序输出所有的n位二进制数，每个数占一行。

输入样例：

3

输出样例：

000  
001  
010  
011  
100  
101  
110  
111

#include <iostream>

using namespace std;

int n,c[20];

void Search(int m)

{

int i;

if(m==n)

{

for(i=0;i<n;i++)

cout<<c[i];

cout<<endl;

}

else

{

c[m]=0;Search(m+1);

c[m]=1;Search(m+1);

}

}

int main()

{

int m=0;

cin>>n;

Search(m);

return 0;

}

### Problem F

### 循环赛日程表

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

用分治算法生成循环赛日程表（1到2的n次方个人）

输入：

一个整数n

输出：

循环赛日程表（1到2的n次方个人）

输入样例：

3

输出样例：

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

#include <stdio.h>

#include <stdlib.h>

int change(int \*\*arr,int row,int n){

int i,j;

if(n==1){

return 0;

}

change(arr,row,n/2);

change(arr,row+n/2,n/2);

//方格转换

for(i = 0;i<n/2;i++){

for(j = 0;j<n/2;j++){

arr[row+i][n/2+j] = arr[row+n/2+i][j];

arr[row+n/2+i][n/2+j] = arr[row+i][j];

}

}

return 0;

}

int main()

{

int i,j,input,n = 1,\*\*arr;

scanf("%d",&input);

for(i = 0;i<input;i++){

n \*= 2;

}

arr = (int\*\*)malloc(sizeof(int\*)\*n);

for(i = 0;i<n;i++){

arr[i] = (int\*)malloc(sizeof(int)\*n);

arr[i][0] = i+1;

}

change(arr,0,n);

for(i = 0;i<n;i++){

for(j = 0;j<n;j++){

if(j==n-1) printf("%d\n",arr[i][j]);

else printf("%d ",arr[i][j]);

}

}

return 0;

}

# 实验二

### Problem A

### 0-1背包问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

需对容量为c 的背包进行装载。从n 个物品中选取装入背包的物品，每件物品i 的重量为wi ，价值为pi 。对于可行的背包装载，背包中物品的总重量不能超过背包的容量，最佳装载是指所装入的物品价值最高。

输入：

多个测例，每个测例的输入占三行。第一行两个整数：n（n<=10）和c，第二行n个整数分别是w1到wn，第三行n个整数分别是p1到pn。  
n 和 c 都等于零标志输入结束。

输出：

每个测例的输出占一行，输出一个整数，即最佳装载的总价值。

输入样例：

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

输出样例：

1  
4

#include <stdio.h>

void readdata();

void search(int);

void checkmax();

int c,n; //c： 背包容量；n：物品数

int w[100], v[100]; //w[i]、v[i]：第i件物品的重量和价值

int a[100], max; //a数组存放当前解各物品选取情况；max：记录最大价值

//a[i]=0表示不选第i件物品，a[i]=1表示选第i件物品

int main()

{

while(1){

max=0;

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

if(n==0&&c==0)

break;

readdata(); //读入数据

search(0);

printf("%d\n",max);

}

return 0;

}

void search(int m)

{

if(m>=n)

checkmax(); //检查当前解是否是可行解，若是则把它的价值与max比较

else

{

a[m]=0; //不选第m件物品

search(m+1); //递归搜索下一件物品

a[m]=1; //不选第m件物品

search(m+1); //递归搜索下一件物品

}

}

void checkmax()

{

int i, weight=0, value=0;

for(i=0;i<n;i++)

{

if(a[i]==1) //如果选取了该物品

{

weight = weight + w[i]; //累加重量

value = value + v[i]; //累加价值

}

}

if(weight<=c) //若为可行解

if(value>max) //且价值大于max

max=value; //替换max

}

void readdata()

{

int i;

for(i=0;i<n;i++)

scanf("%d",&w[i]);

for(i=0;i<n;i++)

scanf("%d",&v[i]);//读入第i件物品重量和价值

}

### Problem B

### 装载问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

有两艘船，载重量分别是c1、 c2，n个集装箱，重量是wi (i=1…n)，且所有集装箱的总重量不超过c1+c2。确定是否有可能将所有集装箱全部装入两艘船。

输入：

多个测例，每个测例的输入占两行。第一行一次是c1、c2和n（n<=10）；第二行n个整数表示wi (i=1…n)。n等于0标志输入结束。

输出：

对于每个测例在单独的一行内输出Yes或No。

输入样例：

7 8 2  
8 7  
7 9 2  
8 8  
0 0 0

输出样例：

Yes  
No

#include <stdio.h>

void readdata();

void search(int);

void checkmax();

int c1,c2,n;

int w[100];

int a[100], max;

int main()

{

int wei,i;

while(1){

max=0,wei=0;

scanf("%d%d%d",&c1,&c2,&n);

if(c1==0&&c2==0&&n==0)

break;

readdata();

for(i=0;i<n;i++)

wei=wei+w[i];

search(0);

if((wei-max)<=c2)

printf("Yes\n");

else printf("No\n");

}

return 0;

}

void search(int m)

{

if(m>=n)

checkmax();

else

{

a[m]=0;

search(m+1);

a[m]=1;

search(m+1);

}

}

void checkmax()

{

int i, weight=0;

for(i=0;i<n;i++)

{

if(a[i]==1)

{

weight = weight + w[i];

}

}

if(weight<=c1)

if(weight>max)

max=weight;

}

void readdata()

{

int i;

for(i=0;i<n;i++)

scanf("%d",&w[i]);

}

### Problem C

### 堡垒问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

城堡是一个4×4的方格，为了保卫城堡，现需要在某些格子里修建一些堡垒。城堡中的某些格子是墙，其余格子都是空格，堡垒只能建在空格里，每个堡垒都可以向上下左右四个方向射击，如果两个堡垒在同一行或同一列，且中间没有墙相隔，则两个堡垒都会把对方打掉。问对于给定的一种状态，最多能够修建几个堡垒。

输入：

每个测例以一个整数n（1<=n<=4）开始，表示城堡的大小。接下来是n行字符每行n个，‘X’表示该位置是墙，‘.’表示该位置是空格。n等于0标志输入结束。

输出：

每个测例在单独的一行输出一个整数：最多修建堡垒的个数。

输入样例：

4  
.X..  
....  
XX..  
....  
2  
XX  
.X  
3  
.X.  
X.X  
.X.  
3  
...  
.XX  
.XX  
4  
....  
....  
....  
....  
0

输出样例：

5  
1  
5  
2  
4

### Problem D

### 8皇后问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

输出8皇后问题所有结果。

输入：

没有输入。

输出：

每个结果第一行是No n：的形式，n表示输出的是第几个结果；下面8行，每行8个字符，‘A’表示皇后，‘.’表示空格。不同的结果中，先输出第一个皇后位置靠前的结果；第一个皇后位置相同，先输出第二个皇后位置靠前的结果；依次类推。

输入样例：

输出样例：

输出的前几行：  
No 1:  
A.......  
....A...  
.......A  
.....A..  
..A.....  
......A.  
.A......  
...A....  
No 2:  
A.......  
.....A..  
.......A  
..A.....  
......A.  
...A....  
.A......  
....A...

#include <stdio.h>

#include <math.h>

void search(int);

void printresult(); //打印结果

int canplace(int,int); //判断该位置能否放置皇后

void place(int,int); //在该位置能否放置皇后

void takeout(int,int); //把该位置放置皇后去掉

int a[8],num=0; //a[i]存放第i个皇后的位置

int main()

{

search(0); //递归搜索

}

void search(int m)

{

int i;

if(m>=8){

num++;

printf("No %d:\n",num);//当已经找出一组解时

printresult();

}

//输出当前结果

else

{

for(i=0;i<8;i++) //对当前行0到7列的每一个位置

{

if(canplace(m,i)) //判断第m个格子是否能放堡垒

{

place(m,i); //在(m,i)格子上放置一个皇后

search(m+1); //递归搜索下一行

takeout(m,i); //把(m,i)格子上的皇后去掉

}

}

}

}

int canplace(int row, int col)

{

int i;

for(i=0;i<row;i++)

if(abs(i-row)==abs(a[i]-col)||a[i]==col)

return(0);

return(1);

}

void place(int row, int col)

{

a[row]=col;

}

void takeout(int row, int col)

{

a[row]=-1;

}

void printresult()

{

int i,j;

for(i=0;i<8;i++)

{

for(j=0;j<8;j++)

if(a[i]==j)

printf("A");

else

printf(".");

printf("\n");

}

}

### Problem E

### 素数环问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

把1到20这重新排列，使得排列后的序列A满足：  
a. 任意相邻两个数之和是素数  
b. 不存在满足条件a的序列B使得：A和B的前k（0 <= k <= 19）项相同且B的第k+1项比A的第k+1项小。

输入：

没有输入。

输出：

输出A，两个数字之间用一个空格隔开，第一个数字前面和最后一个数字后面没有空格。

输入样例：

输出样例：

#include<stdio.h>

void search(int t);

int c[41]={0,1,1,0,1,0,1,0,0,0,1,0,1,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,1};

int a[20],b[20],s=0;

int main()

{

int i;

for(i=0;i<20;i++)

b[i]=0;

a[0]=1;

b[0]=1;

search(1);

return 0;

}

void search(int t)

{

if(s==1)

return ;

int i;

if(t>19)

{

for(i=0;i<19;i++)

printf("%d ",a[i]);

printf("%d\n",a[19]);

s=1;

}

else if(t!=19)

{

for(i=0;i<20;i++)

{

if(b[i]==0&&c[a[t-1]+i]==1)

{

b[i]=1;

a[t]=i+1;

search(t+1);

b[i]=0;

}

}

}

else

{

for(i=0;i<20;i++)

{

if(b[i]==0&&c[a[t-1]+i]==1&&c[1+i]==1)

{

b[i]=1;

a[t]=i+1;

search(t+1);

b[i]=0;

}

}

}

}

### Problem F

### 迷宫问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

给一个20×20的迷宫、起点坐标和终点坐标，问从起点是否能到达终点。

输入：

多个测例。输入的第一行是一个整数n，表示测例的个数。接下来是n个测例，每个测例占21行，第一行四个整数x1，y1，x2，y2是起止点的位置（坐标从零开始），（x1，y1）是起点，（x2，y2）是终点。下面20行每行20个字符，’.’表示空格；’X’表示墙。

输出：

每个测例的输出占一行，输出Yes或No。

输入样例：

2  
0 0 19 19  
....................  
XXXXXXXXXXXXXXXXXXXX  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
....................  
0 0 19 19  
....................  
XXXXXXXXXXXXXXXXXXX.  
....................  
.XXXXXXXXXXXXXXXXXXX  
....................  
XXXXXXXXXXXXXXXXXXX.  
....................  
.XXXXXXXXXXXXXXXXXXX  
....................  
XXXXXXXXXXXXXXXXXXX.  
....................  
.XXXXXXXXXXXXXXXXXXX  
....................  
XXXXXXXXXXXXXXXXXXX.  
XXXXXXXXXXXXXXXXXXX.  
XXXXXXXXXXXXXXXXXXX.  
XXXXXXXXXXXXXXXXXXX.  
....................  
.XXXXXXXXXXXXXXXXXXX  
....................

输出样例：

No  
Yes

#include<stdio.h>

int han(char A[][100],int i,int j,int e,int f, int \*fi)

{

int ii=i,ji=j;

if(ii==e&&ji==f&&A[ii][ji]!='X')

(\*fi)=1;

else if(A[ii][ji]=='.'){

A[ii][ji]='X';

han(A,ii+1,ji,e,f,fi);

han(A,ii-1,ji,e,f,fi);

han(A,ii,ji+1,e,f,fi);

han(A,ii,ji-1,e,f,fi);

}

return 0;

}

int main()

{

int b[1000],a,bi,c,k,l,m,n,o,pan=0;

char B[100][100];

int col=0;

scanf("%d\n",&a);

for(bi=0;bi<a;bi++){

col=0;

scanf("%d %d ",&c,&k);

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

for(n=0;n<20;n++)

for(o=0;o<21;o++)

scanf("%c",&B[n][o]);

/\*if(B[c][k]=='X'&&c==l&&k==m)

b[pan]=1;\*/

han(B,c,k,l,m,&col);

if(col==1)

b[pan]=1;

else b[pan]=0;

pan++;

}

for(bi=0;bi<a;bi++)

{

if(b[bi]==1)

printf("Yes\n");

else printf("No\n");

}

return 0;

}

### Problem G

### 踩气球

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

六一儿童节，小朋友们做踩气球游戏，气球的编号是1～100，两位小朋友各踩了一些气球，要求他们报出自己所踩气球的编号的乘积。现在需要你编一个程序来判断他们的胜负，判断的规则是这样的：如果两人都说了真话，数字大的人赢；如果两人都说了假话，数字大的人赢；如果报小数字的人说的是真话而报大数字的人说谎，则报小数字的人赢（注意：只要所报的小数字是有可能的，即认为此人说了真话）。

输入：

输入为两个数字，0 0表示结束；

输出：

输出为获胜的数字。

输入样例：

36 62

49 343

0 0

输出样例：

62

49

#include <iostream>

#include <algorithm>

using namespace std;

bool fa, fb;

void dfs(int a, int b, int k)

{

if(b == 1)

{

fb = true;

if(a == 1)

fa = true;

}

if(k == 1 || (fa&&fb))

return;

if(a % k == 0)

dfs(a/k, b, k-1);

if(b% k == 0)

dfs(a, b/k, k-1);

dfs(a, b, k-1);

}

int main()

{

int a, b;

while(cin >> a >> b && (a != 0 && b != 0))

{

if(a < b)

swap(a, b);

fa = fb = 0;

dfs(a, b, 100);

if(!fa && fb)

cout << b << endl;

else

cout << a << endl;

}

return 0;

}

### Problem H

### 字母转换

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

通过栈交换字母顺序。给定两个字符串，要求所有的进栈和出栈序列（i表示进栈，o表示出栈），使得字符串2在求得的进出栈序列的操作下，变成字符串1。输出结果需满足字典序。例如TROT 到 TORT:  
[  
i i i i o o o o  
i o i i o o i o  
]

输入：

给定两个字符串，第一个字符串是源字符串，第二个字符是目标目标字符串。

输出：

所有的进栈和出栈序列,输出结果需满足字典序

输入样例：

madam

adamm

bahama

bahama

long

short

eric

rice

输出样例：

[

i i i i o o o i o o

i i i i o o o o i o

i i o i o i o i o o

i i o i o i o o i o

]

[

i o i i i o o i i o o o

i o i i i o o o i o i o

i o i o i o i i i o o o

i o i o i o i o i o i o

]

[

]

[

i i o i o i o o

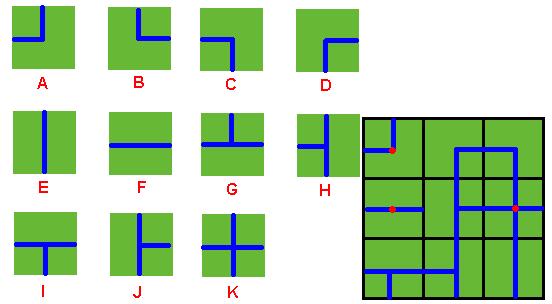
]

### Problem I

### 农场灌溉问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

一农场由图所示的十一种小方块组成，蓝色线条为灌溉渠。若相邻两块的灌溉渠相连则只需一口水井灌溉。  


输入：

给出若干由字母表示的最大不超过50×50具体由(m，n)表示，的农场图

输出：

编程求出最小需要打的井数。每个测例的输出占一行。当M=N=-1时结束程序。

输入样例：

2 2

DK

HF

3 3

ADC

FJK

IHE

-1 -1

输出样例：

2

3

#include <iostream>

using namespace std;

int m, n;

struct Maze

{

char ch; //农场类型

int visited; //访问标记

int left, right, up, down; //连通线

}maze[100][100];

int dir[4][2] = {{1,0}, {-1,0}, {0,-1}, {0,1}}; //上下左右四个方向

bool Place(int x, int y, int nx, int ny)

{

//未越界且没有被访问过

if(nx>0 && nx<=m && ny>0 && ny<=n && !maze[nx][ny].visited)

{

//向左

if(nx==x && ny==y-1 && maze[x][y].left && maze[nx][ny].right)

return true;

//向下

if(nx==x+1 && ny==y && maze[x][y].down && maze[nx][ny].up)

return true;

//向右

if(nx==x && ny==y+1 && maze[x][y].right && maze[nx][ny].left)

return true;

//向上

if(nx==x-1 && ny==y && maze[x][y].up && maze[nx][ny].down)

return true;

}

return 0;

}

void dfs(int x, int y)

{

maze[x][y].visited = 1; //当前点已访问

for(int i = 0; i < 4; i++)

{

int nx = x+dir[i][0];

int ny = y+dir[i][1];

//将所有可连通的区域全部标记

if(Place(x, y, nx, ny))

dfs(nx, ny);

}

}

int main()

{

while(cin >> m >> n && (m!=-1 || n!=-1))

{

int cnt = 0;

for(int i = 1; i <= m; i++)

for(int j = 1; j <= n; j++)

{

cin >> maze[i][j].ch;

//初始化

maze[i][j].visited = 0;

maze[i][j].down = 0;

maze[i][j].up = 0;

maze[i][j].right = 0;

maze[i][j].left = 0;

switch(maze[i][j].ch)

{

case'A': maze[i][j].left=1;

maze[i][j].up=1;

break;

case'B': maze[i][j].right=1;

maze[i][j].up=1;

break;

case'C': maze[i][j].left=1;

maze[i][j].down=1;

break;

case'D': maze[i][j].right=1;

maze[i][j].down=1;

break;

case'E': maze[i][j].up=1;

maze[i][j].down=1;

break;

case'F': maze[i][j].left=1;

maze[i][j].right=1;

break;

case'G': maze[i][j].left=1;

maze[i][j].right=1;

maze[i][j].up=1;

break;

case'H': maze[i][j].left=1;

maze[i][j].up=1;

maze[i][j].down=1;

break;

case'I': maze[i][j].left=1;

maze[i][j].right=1;

maze[i][j].down=1;

break;

case'J': maze[i][j].right=1;

maze[i][j].up=1;

maze[i][j].down=1;

break;

case'K': maze[i][j].left=1;

maze[i][j].right=1;

maze[i][j].up=1;

maze[i][j].down=1;

break;

}

}

//连同区域的个数

for(int i = 1; i <= m; i++)

for(int j = 1; j <= n; j++)

{

if(maze[i][j].visited == 0) //未被访问过

{

dfs(i, j);

cnt ++;

}

}

cout << cnt << endl;

}

return 0;

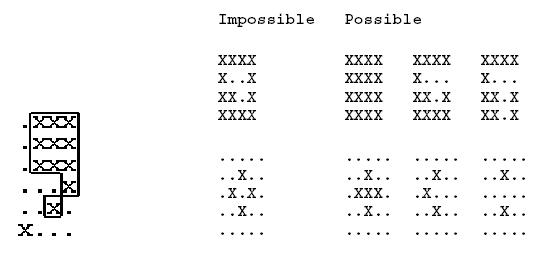
}

### Problem J

### 求图像的周长

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

给一个用 . 和X表示的图形，图形在上、下、左、右、左上、左下、右上、右下8个方向都被看作是连通的，并且图像中间不会出现空洞，求这个图形的边长。  


输入：

首先给出m、n、x、y四个正整数，下面给出m×n的图形，x、y表示点击的位置，全0表示结束。

输出：

点击的图形的周长。

输入样例：

2 2 2 2

XX

XX

6 4 2 3

.XXX

.XXX

.XXX

...X

..X.

X...

0 0 0 0

输出样例：

8

18

#include <iostream>

#include <cstring>

#include <queue>

#include <algorithm>

using namespace std;

char maze[50][50]; //地图

int visited[50][50]; //访问标记

int m, n;

int cnt; //计数

int dx[8] = {0,1,1,1,0,-1,-1,-1}; //8联通

int dy[8] = {1,1,0,-1,-1,-1,0,1};

typedef struct Point

{

int x;

int y;

}point;

int main()

{

queue<point> q;

point s;

while(cin >> m >> n >> s.x >> s.y && (s.x!=0||s.y!=0||n!=0||m!=0))

{

//初始化

cnt = 0;

memset(visited, 0, sizeof(visited));

while(!q.empty())

q.pop();

for(int i = 1; i <= m; i++)

for(int j = 1; j <= n; j++)

cin >> maze[i][j];

//第一列最后一列初始化为。

for(int i = 0; i <= m+1; i++)

{

maze[i][0] = '.';

maze[i][n+1] = '.';

}

//第一行最后一行初始化为。

for(int j = 0; j <= n+1; j++)

{

maze[0][j] = '.';

maze[m+1][j] = '.';

}

q.push(s); //放入初始结点

visited[s.x][s.y] = 1; //标记已访问

while(!q.empty())

{

s = q.front();

q.pop();

for(int i = 0; i < 8; i ++)

{

point temp;

temp.x = s.x + dx[i];

temp.y = s.y + dy[i];

if(temp.x>=1 && temp.x<=m && temp.y>=1 && temp.y<=n && //未越界

maze[temp.x][temp.y]=='X' && !visited[temp.x][temp.y]) //可达且未访问

{

q.push(temp);

visited[temp.x][temp.y] = 1;

}

}

}

//计算边长

for(int i = 1; i <= m; i++)

{

for(int j = 1; j <= n; j++)

{

if(visited[i][j])

{

if(maze[i-1][j] == '.')

cnt ++;

if(maze[i][j-1] == '.')

cnt ++;

if(maze[i+1][j] == '.')

cnt ++;

if(maze[i][j+1] == '.')

cnt ++;

}

}

}

cout << cnt << endl;

}

return 0;

}

# 实验三

### Problem A

### 最长公共子序列

#### 时限：1000ms 内存限制：200000K 总时限：3000ms

描述：

一个给定序列的子序列是在该序列中删去若干元素后得到的序列。确切地说，若给定序列X=<x1, x2,…, xm>，则另一序列Z=<z1, z2,…, zk>是X的子序列是指存在一个严格递增的下标序列 <i1, i2,…, ik>，使得对于所有j=1,2,…,k有：  
  
Xij ＝ Zj  
  
如果一个序列S即是A的子序列又是B的子序列，则称S是A、B的公共子序列。  
求A、B所有公共子序列中最长的序列的长度。

输入：

输入共两行，每行一个由字母和数字组成的字符串，代表序列A、B。A、B的长度不超过200个字符。

输出：

一个整数，表示最长各个子序列的长度。  
格式：printf("%d\n");

输入样例：

programming  
contest

输出样例：

2

#include<stdio.h>

#include<string.h>

int lcs\_length(char x[], char y[]);

int main()

{

char x[201],y[201];

int len;

scanf("%s%s",x,y);

len=lcs\_length(x,y);

printf("%d\n",len);

}

int lcs\_length(char x[], char y[] )

{

int m,n,i,j,l[201][201];

m=strlen(x);

n=strlen(y);

for(i=0;i<m+1;i++)

l[i][0]=0;

for(j=0;j<n+1;j++)

l[0][j]=0;

for(i=1;i<=m;i++)

for(j=1;j<=n;j++)

if(x[i-1]==y[j-1]) //i,j从1开始，但字符串是从0开始

l[i][j]=l[i-1][j-1]+1;

else if(l[i][j-1]>l[i-1][j])

l[i][j]=l[i][j-1];

else

l[i][j]=l[i-1][j];

return l[m][n];

}

### Problem B

### 防卫导弹

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

一种新型的防卫导弹可截击多个攻击导弹。它可以向前飞行，也可以用很快的速度向下飞行，可以毫无损伤地截击进攻导弹，但不可以向后或向上飞行。但有一个缺点，尽管它发射时可以达到任意高度，但它只能截击比它上次截击导弹时所处高度低或者高度相同的导弹。现对这种新型防卫导弹进行测试，在每一次测试中，发射一系列的测试导弹（这些导弹发射的间隔时间固定，飞行速度相同），该防卫导弹所能获得的信息包括各进攻导弹的高度，以及它们发射次序。现要求编一程序，求在每次测试中，该防卫导弹最多能截击的进攻导弹数量，一个导弹能被截击应满足下列两个条件之一：  
a)它是该次测试中第一个被防卫导弹截击的导弹；  
b)它是在上一次被截击导弹的发射后发射，且高度不大于上一次被截击导弹的高度的导弹。

输入：

多个测例。  
每个测例第一行是一个整数n（n不超过100），第二行n个整数表示导弹的高度（数字的顺序即发射的顺序）。  
n＝0表示输入结束。

输出：

每个测例77独的一行内输出截击导弹的最大数目。

输入样例：

5  
5 6 100 6 61  
 0

输出样例：

2

#include<stdio.h>

int h[100]={0};//靶弹高度

int n;//靶弹数量

int Fanwei()

{

int i,j;

int num[101]={0};//num[j]存放从j号靶弹开始截击,能截击的最大数量(包括j号靶弹)

int maxi,MAX;

num[n-1]=1;//从最后一枚靶弹开始截击，能截击的最大数量为1

for(i=n-2;i>=0;i--)//i=n-2,n-1.....2,1,0

{

maxi=0;//截击i号导弹后,能截击的最大数量(不包括i号靶弹)

for(j=i+1;j<n;j++)

if(h[i]>=h[j] &&num[j]>maxi)

maxi=num[j];

num[i]=maxi+1;//从i号靶弹开始截击,能截击的最大数量

}

for(i=0;i<n;i++)//取最大截击数量

if(num[i]>MAX)

MAX=num[i];

return(MAX);

}

int main()

{

int i;

scanf("%d",&n);//n个靶弹

while(n!=0)

{

for(int i=0;i<n;i++)

scanf("%d",&h[i]); //靶弹的高度

printf("%d\n",Fanwei());

for(i=0;i<100;i++) h[0]=0;//初始化，测试下一组数据

scanf("%d",&n);

}

return 0;

}

### Problem C

### 田忌赛马(tian ji racing)

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

田忌与齐王赛马，双方各有n匹马参赛（n<=100），每场比赛赌注为1两黄金，现已知齐王与田忌的每匹马的速度，并且齐王肯定是按马的速度从快到慢出场，现要你写一个程序帮助田忌计算他最好的结果是赢多少两黄金（输用负数表示）。  
Tian Ji and the king play horse racing, both sides have n horse (n is no more the 100), every game a bet of 1 gold, now known king and Tian Ji each horse's speed, and the king is definitely on the horse speed from fast to slow, we want you to write a program to help Tian Ji his best result is win the number gold (lost express with the negative number).

输入：

多个测例。  
每个测例三行：第一行一个整数n，表示双方各有n匹马；第二行n个整数分别表示田忌的n匹马的速度；第三行n个整数分别表示齐王的n匹马的速度。  
n＝0表示输入结束。  
A plurality of test cases.  
Each test case of three lines: the first line contains an integer n, said the two sides each have n horse; second lines of N integers n Tian Ji horse speed; third lines of N integers King n horse speed.  
N = 0 indicates the end of input.

输出：

每行一个整数，田忌最多能赢多少两黄金。  
how many gold the tian ji win

输入样例：

3  
92 83 71  
95 87 74  
2  
20 20  
20 20  
2  
20 19  
22 18  
3  
20 20 10  
20 20 10  
0

输出样例：

1  
0  
0  
0

#include <iostream>

#include <vector>

using namespace std;

void change(int &a, int &b)

{

int temp;

temp = a;

a = b;

b = temp;

}

void quickSort(int\* a, int l, int u)

{

int i, m;

if (l >= u) return;

m = l;

for (i = l + 1; i <= u; i++)

if (a[i] > a[l])

change(a[++m], a[i]);

change(a[l], a[m]);

quickSort(a, l, m - 1);

quickSort(a, m + 1, u);

}

int main()

{

vector<int> result;

vector<int> ra;

vector<int> rb;

int num;

int\* a;

int\* b;

while (cin>>num)

{

if (num == 0)

break;

a = new int[num];

b = new int[num];

for (int i = 0; i < num; i++)

cin>>a[i];

for (int i = 0; i < num; i++)

cin>>b[i];

quickSort(a, 0, num-1);

quickSort(b, 0, num-1);

int win = 0;

int fail = 0;

int draw = 0;

int ib = 0, jb = 0;

int ie = num - 1, je = num - 1;

while (ib <= ie)

{

if (a[ie] > b[je])

{

win++;

ie--;

je--;

}else if (a[ie] < b[je])

{

fail++;

ie--;

jb++;

}else

{

if (a[ib] > b[jb])

{

win++;

ib++;

jb++;

}else

{

if (a[ie] < b[jb])

fail++;

ie--;

jb++;

}

}

}

result.push\_back(1\*(win - fail));

}

for (size\_t i = 0; i != result.size(); i++)

{

cout<<result[i]<<endl;

}

return 0;

}

### Problem D

### 计算矩阵连乘积

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

在科学计算中经常要计算矩阵的乘积。矩阵A和B可乘的条件是矩阵A的列数等于矩阵B的行数。若A是一个p×q的矩阵，B是一个q×r的矩阵，则其乘积C=AB是一个p×r的矩阵。计算C=AB总共需要p×q×r次乘法。  
现在的问题是，给定n个矩阵{A1,A2,…,An}。其中Ai与Ai+1是可乘的，i=1,2,…,n-1。  
要求计算出这n个矩阵的连乘积A1A2…An最少需要多少次乘法。

输入：

输入数据的第一行是一个整树n（0 < n <= 10），表示矩阵的个数。  
接下来的n行每行两个整数p,q( 0 < p,q < 100)，分别表示一个矩阵的行数和列数。

输出：

输出一个整数：计算连乘积最少需要乘法的次数。

输入样例：

10   
1 2   
2 3   
3 4   
4 5   
5 6   
6 7   
7 8   
8 9   
9 10   
10 11

输出样例：

438

#include <iostream>

#include <vector>

using namespace std;

void change(int &a, int &b)

{

int temp;

temp = a;

a = b;

b = temp;

}

void quickSort(int\* a, int l, int u)

{

int i, m;

if (l >= u) return;

m = l;

for (i = l + 1; i <= u; i++)

if (a[i] > a[l])

change(a[++m], a[i]);

change(a[l], a[m]);

quickSort(a, l, m - 1);

quickSort(a, m + 1, u);

}

int main()

{

vector<int> result;

vector<int> ra;

vector<int> rb;

int num;

int\* a;

int\* b;

while (cin>>num)

{

if (num == 0)

break;

a = new int[num];

b = new int[num];

for (int i = 0; i < num; i++)

cin>>a[i];

for (int i = 0; i < num; i++)

cin>>b[i];

quickSort(a, 0, num-1);

quickSort(b, 0, num-1);

int win = 0;

int fail = 0;

int draw = 0;

int ib = 0, jb = 0;

int ie = num - 1, je = num - 1;

while (ib <= ie)

{

if (a[ie] > b[je])

{

win++;

ie--;

je--;

}else if (a[ie] < b[je])

{

fail++;

ie--;

jb++;

}else

{

if (a[ib] > b[jb])

{

win++;

ib++;

jb++;

}else

{

if (a[ie] < b[jb])

fail++;

ie--;

jb++;

}

}

}

result.push\_back(1\*(win - fail));

}

for (size\_t i = 0; i != result.size(); i++)

{

cout<<result[i]<<endl;

}

return 0;

}

### Problem E

### 石子合并

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

在一个圆形操场的四周摆放着n堆石子(n<= 100)，现要将石子有次序地合并成一堆。规定每次只能选取相邻的两堆合并成新的一堆,并将新的一堆的石子数,记为该次合并的得分。编一程序，读入石子堆数n及每堆的石子数(<=20)。选择一种合并石子的方案,使得做n－1次合并,得分的总和最小；比如有4堆石子：4 4 5 9 则最佳合并方案如下：  
4 4 5 9 score: 0  
8 5 9 score: 8  
13 9 score: 8 + 13 = 21  
22 score: 8 + 13 + 22 = 43

输入：

可能有多组测试数据。 当输入n=0时结束! 第一行为石子堆数n(1<=n<=100)；第二行为n堆的石子每堆的石子数,每两个数之间用一个空格分隔。

输出：

合并的最小得分，每个结果一行。

输入样例：

4

4 4 5 9

0

输出样例：

43

#include<stdio.h>

#define N 100

/\*

\*求合并过程中

\*最少合并堆数目

\*\*/

int MatrixChain\_min(int p[N],int n)

{

//定义二维数组m[i][j]来记录i到j的合并过成中最少石子数目

//此处赋值为-1

int m[N][N];

for(int x=1;x<=n;x++)

for(int z=1;z<=n;z++)

{

m[x][z]=-1;

}

int min=0;

//当一个单独合并时，m[i][i]设为0，表示没有石子

for(int g = 1;g<=n;g++) m[g][g]=0;

//当相邻的两堆石子合并时，此时的m很容易可以看出是两者之和

for(int i=1;i<=n-1;i++)

{

int j=i+1;

m[i][j]=p[i]+p[j];

}

//当相邻的3堆以及到最后的n堆时，执行以下循环

for(int r=3; r<=n;r++)

for(int i=1;i<=n-r+1;i++)

{

int j = i+r-1; //j总是距离i r-1的距离

int sum=0;

//当i到j堆石子合并时最后里面的石子数求和得sum

for(int b=i;b<=j;b++)

sum+=p[b];

// 此时m[i][j]为i~j堆石子间以m[i][i]+m[i+1][j]+sum结果，这是其中一种可能，不一定是最优

//要与下面的情况相比较，唉，太详细了

m[i][j] = m[i+1][j]+sum;

//除上面一种组合情况外的其他组合情况

for(int k=i+1;k<j;k++)

{

int t=m[i][k]+m[k+1][j]+sum;

if(t<m[i][j])

m[i][j] = t;

}

}

//最终得到最优解

min=m[1][n];

return min;

}

/\*

\*求合并过程中

\*最多合并堆数目

\*\*/

int MatrixChain\_max(int p[N],int n)

{

int m[N][N];

for(int x=1;x<=n;x++)

for(int z=1;z<=n;z++)

{

m[x][z]=-1;

}

int max=0;

//一个独自组合时

for(int g = 1;g<=n;g++) m[g][g]=0;

//两个两两组合时

for(int i=1;i<=n-1;i++)

{

int j=i+1;

m[i][j]=p[i]+p[j];

}

for(int r=3; r<=n;r++)

for(int i=1;i<=n-r+1;i++)

{

int j = i+r-1;

int sum=0;

for(int b=i;b<=j;b++)

sum+=p[b];

m[i][j] = m[i+1][j]+sum;

for(int k=i+1;k<j;k++)

{

int t=m[i][k]+m[k+1][j]+sum;

if(t>m[i][j])

m[i][j] = t;

}

}

max=m[1][n];

return max;

}

int main()

{

int stone[N];

int min=0;

int max=0;

int n;

scanf("%d",&n);

while(n!=0)

{

for(int i=1;i<=n;i++)

scanf("%d",&stone[i]);

min= MatrixChain\_min(stone,n);

max= MatrixChain\_max(stone,n);

//因为题目要求圆的原因，要把所有情况都要考虑到，总共有n种情况。

for(int j=1;j<=n-1;j++)

{

int min\_cache=0;

int max\_cache=0;

int cache= stone[1];

for(int k=2;k<=n;k++)

{

stone[k-1]=stone[k];

}

stone[n]=cache;

min\_cache= MatrixChain\_min(stone,n);

max\_cache= MatrixChain\_max(stone,n);

if(min\_cache<min)

min=min\_cache;

if(max\_cache>max)

max=max\_cache;

}

printf("%d\n",min);

/\*printf("%d\n",max);\*/

scanf("%d",&n);

}

return 1;

}

### Problem F

### 旅游预算

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

一个旅行社需要估算乘汽车从某城市到另一城市的最小费用，沿路有若干加油站，每个加油站收费不一定相同。旅游预算有如下规则：若油箱的油过半，不停车加油，除非油箱中的油不可支持到下一站；每次加油时都加满；在一个加油站加油时，司机要花费2元买东西吃；司机不必为其他意外情况而准备额外的油；汽车开出时在起点加满油箱；计算精确到分（1元=100分）。编写程序估计实际行驶在某路线所需的最小费用。

输入：

第一行为起点到终点的距离（实数）第二行为三个实数，后跟一个整数，每两个数据间用一个空格隔开。其中第一个数为汽车油箱的容量（升），第二个数是每升汽油行驶的公里数，第三个数是在起点加满油箱的费用（精确到分），第四个数是加油站的数量。（〈=50）。接下去的每行包括两个实数，每个数据之间用一个空格分隔，其中第一个数是该加油站离起点的距离，第二个数是该加油站每升汽油的价格（元/升）。加油站按它们与起点的距离升序排列。所有的输入都有一定有解。

输出：

共两行，每行都有换行第一行为一个实数和一个整数，实数为旅行的最小费用，以元为单位，精确到分，整数表示途中加油的站的N。第二行是N个整数，表示N个加油的站的编号，按升序排列。数据间用一个空格分隔，最后一个数据后也输出空格，此外没有多余的空格。

输入样例：

516.3

15.7 22.1 20.87 3

125.4 1.259

297.9 1.129

345.2 0.999

输出样例：

38.09 1

2

#include<stdio.h>

struct gstation

{

int num;//number of gas station

double dis;// distance to the start

double fee;// cost of per L

} GS[51];

int count=0,n;

double b[1000]= {0}; //cost of one routine

double restv[51];//the rest of oil when arrived to NO.i station

double distance;

double volume,MilesPerL,cost1;

int a[51],c[51];//state of gas station

double min=100000;

void check()

{

int i;

for(i=1; i<=n; i++)

if(a[i]==1)

{

b[count]+=(volume-restv[i])\*GS[i].fee+2;

}

b[count]+=cost1;

if(b[count]<min)

{

min=b[count++];

for(i=1; i<=n; i++)

c[i]=a[i];

}

}

void search(int m)

{

if(m>n)

check();

else if(m<n)

{

if(restv[m]\*MilesPerL>=GS[m+1].dis-GS[m].dis)

{

a[m]=0;

restv[m+1]=restv[m]-(GS[m+1].dis-GS[m].dis)/MilesPerL;

search(m+1);

}

if(restv[m]\*MilesPerL<GS[m+1].dis-GS[m].dis ||restv[m]<=0.5\*volume) //attention: the rest of oil must be <= half of the volume!

{

a[m]=1;

restv[m+1]=volume-(GS[m+1].dis-GS[m].dis)/MilesPerL;

search(m+1);

}

}

else

{

if(restv[m]\*MilesPerL>=distance-GS[m].dis)

{

a[m]=0;

search(m+1);

}

else

{

a[m]=1;

search(m+1);

}

}

}

int main()

{

int i,j=0,flag=0;

scanf("%lf",&distance);

scanf("%lf %lf %lf %d",&volume,&MilesPerL,&cost1,&n);

for(i=1; i<=n; i++)

{

scanf("%lf %lf",&GS[i].dis,&GS[i].fee);

GS[i].num=i;

}

restv[1]=volume-GS[1].dis/MilesPerL;

search(1);

for(j=1; j<=n; j++)

if(c[j]==1) flag++;

printf("%.2lf %d\n",min,flag);

for(j=1; j<=n; j++)

if(c[j]==1)

printf("%d ",j); //printf("%d",c[n]);

printf("\n");

return 0;

}

### Problem G

### 花生米（二）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

五一长假第二天，Tom和Jerry在仓库散步的时候又发现了一堆花生米（这个仓库还真奇怪）。这次Tom制定分花生米规则如下：   
       1、Tom和Jerry轮流从堆中取出k粒花生米吃掉，k可以是1，5，10中的任意一个数字；   
       2、为显示规则的公平性，Jerry可以选择先取或者后取。   
Jerry当然还是希望最后一粒花生米被Tom吃掉。请计算，Jerry为了达到目的应该先取还是后取。

输入：

本题有多个测例，每个测例的输入是一个整数n，n大于零小于等于1000，代表花生米的数量。  
n等于0表示输入结束，不需要处理。

输出：

每个测例在单独的一行内输出一个整数：Jerry先取输出1；Tom先取输出0。

输入样例：

1  
2  
3  
4  
0

输出样例：

0  
1  
0  
1

#include<stdio.h>

int List[1001]={0};//对取第i粒花生米的判断(List[0]不用)

void search()

{

int i;

for(int i=1;i<=10;i++)//1~10粒时每次人工判断得出规律

{ //可以看出;若List[i]=1,则必有List[i-1]=0，List[i-5]=0

if(i%2==0) List[i]=1;

else List[i]=0;

}

for(i=11;i<=1000;i++)

{

if(List[i-1]==1 &&List[i-5]==1 &List[i-10]==1)

List[i]=0;//如果取1,5,10粒后应由Jerry先取才会赢，则当前让Tom先取，不论Tom怎么取，Jerry都会赢

else

List[i]=1;//如果Tom先取Jerry会输，则Jerry先取Jerry会赢（为什么？），当前让Jerry先取

}

}

int main()

{

int n;

search();

scanf("%d",&n);//输入第一个测例

while(n)

{

printf("%d\n",List[n]);

scanf("%d",&n);

}

return 0;

}

### Problem H

### 花生米（三）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

五一长假第三天，Tom和Jerry在仓库散步的时候又发现了一堆花生米（仓库，又见仓库……）。这次Tom制定分花生米规则如下：   
???????1、Tom和Jerry轮流从堆中取出k粒花生米吃掉；   
2、第一次取花生米的人只能取一粒，以后取花生米的数量不能超过前一个人取花生米数量的两倍；   
3、为显示规则的公平性，Jerry可以选择先取或者后取。   
Jerry当然还是希望最后一粒花生米被Tom吃掉。请计算，Jerry为了达到目的应该先取还是后取。

输入：

本题有多个测例，每个测例的输入是一个整数n，n大于零小于等于1000，代表花生米的数量。  
n等于0表示输入结束，不需要处理。

输出：

每个测例在单独的一行内输出一个整数：Jerry先取输出1；Tom先取输出0。

输入样例：

1  
2  
3  
4  
5  
0

输出样例：

0  
1  
0  
0  
1

#include<iostream>

using namespace std;

bool dp[1001][1001];

//dp[i][j]表示还剩i粒花生米，并且当前可以取j粒的时候,要使对方取走最后一粒，是否应该先取

int main()

{

int n;

cin>>n;

while(n!=0)

{

for(int i=2;i<=n;++i)

{

for(int j=1;j<i-1;++j)

{

bool flag=false;

for(int k=1;k<=j&&k<=i;++k)

if(!dp[i-k][2\*k])

{

flag=true;

break;

}

dp[i][j]=flag;

}

for(int j=i-1;j<=n;++j)

dp[i][j]=true;

}

cout<<dp[n][1]<<endl;

cin>>n;

}

}

### Problem I

### 花生米（四）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

五一长假第四天，Tom逛了N个bbs，做了N^2个脑筋急转弯题，终于发现了每次吃完花生米嘴里都是苦味的原因：自己总会吃到最后一粒花生米。   
五一长假第五天，Tom和Jerry在仓库散步的时候发现了n堆花生米（这是什么仓库啊？！）。这次Tom制定分花生米规则如下：   
???????1、首先选出最苦的一粒花生米，放到一个瓶子里；   
???????2、Tom和Jerry轮流从任意一堆花生米中取出任意粒吃掉；   
???????3、如果Tom吃到最后一粒花生米（瓶子里的花生米除外），Jerry必须吃掉瓶子里的花生米；如果Jerry吃到最后一粒花生米（瓶子里的花生米除外），Tom自愿吃掉瓶子里的花生米；   
?????? 4、为显示规则的公平性，Jerry可以选择先取或者后取。   
Jerry当然希望瓶子里的花生米被Tom吃掉。请计算，Jerry为了达到目的应该先取还是后取。

输入：

本题有多个测例。  
每个测例的输入第一行是一个整数n，n大于零小于等于10，代表花生米的堆数（取出最苦花生米后）；接下来n行每行一个不超过100的正整数，分别代表一堆花生米的粒数（取出最苦花生米后）。  
n等于0表示输入结束，不需要处理。

输出：

每个测例在单独的一行内输出一个整数：Jerry先取输出1；Tom先取输出0。

输入样例：

2  
1  
1  
2  
1  
100  
0

输出样例：

0  
1

#include<stdio.h>

int num[10]={0};//每堆花生米数量

int n;//(0~10)

int search()//直接根据n的只判断Jerry取的先后次序(算法中没有循环)

{

int count=0;

if(n==1) return 1;//只有一堆花生米，Jerry取走该堆数量-1即可赢

else

{ for(int i=0;i<n;i++)

if(num[i]==1)//统计单粒堆的堆数

count++;

if(count==n)//n堆花生米，都是单粒堆

{

if(n%2==0)//由于n取0~10，与花生米(二)的前10粒算法相同

return 0;

else

return 1;

}

else//至少有一堆不是单粒堆

{

if(n==2)//只有2堆花生米

{

//若Jerry取完后两堆只剩下(1,1)(2,2)....(k,k),则Jerry必赢

//因为只要之后每次Tom取之后，Jerry取与Tom相同的数量就必赢

if(num[0]==num[1]) return 0;

else return 1;

}

else//多于2堆花生米

{ int L1,L2;//标记单粒堆和多粒堆数量的奇偶

if(count>0 &&count%2==0)//(有单粒堆)单粒堆的数量为偶数（注意都不大于10）

L1=0; //让对方先取，Jerry能取道最后一粒

else L1=1;

if((n-count)%2==0)//多粒堆的数量为偶数

L2=0;

else L2=1;

//////////////////////////////////////??????

if(L1==1 &&L2==1)//单粒堆数量和多粒堆数量均为奇数

return 1;

else

return 0;//单粒堆数量和多粒堆数量中有偶数，则让对方取数成(单粒堆数量和多粒堆数量)均奇，

//在对方取数成均奇之前Jerry则一直取数保持(单粒堆数量和多粒堆数量)非均奇

//////////////////////////////////////

}

}

}

}

int main()

{

scanf("%d",&n);//输入花生米堆数(0~10之间)

while(n)

{

for(int i=0;i<n;i++)

scanf("%d",&num[i]);//每堆花生米的数量num[i](0~100之间)（除去苦的那粒花生米 ）

int temp=search();

printf("%d\n",temp);

scanf("%d",&n);//下一轮测试

}

return 0;

}

### Problem J

### 花生米（五）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

五一长假第六天，Tom在QQ上遇到了Kitty。呵呵，Kitty，在离散数学课上认识的PPMM……等等！Tom恍然大悟：自己这一生除了看帖不回之外最大的错误就是离散数学没学好！   
五一长假第七天，Tom和Jerry在仓库散步的时候发现了一堆花生米（仓库，呵呵，仓库…）。这次Tom制定分花生米规则如下：   
???????1、首先选出最苦的一粒花生米，放到一个瓶子里；   
???????2、把剩下的花生米做成花生酱，Tom和Jerry轮流取一些花生酱吃掉；   
???????3、第一个取的人只能取1.0克，以后取花生酱的数量不能少于两个人已经取过的总数量且不能超过两个人已经取过的总数量的三倍；   
?????? 4、不能按规则3取花生酱的人必须吃掉瓶子里的花生米；   
?????? 5、为显示规则的公平性，Jerry可以选择先取或者后取。   
Jerry当然希望瓶子里的花生米被Tom吃掉。请计算，Jerry为了达到目的应该先取还是后取。

输入：

本题有多个测例，每个测例的输入是一个浮点数w，w大于1.0小于等于1000.0，w最多只有一位小数，代表花生酱的数量，单位为克。  
w小于0表示输入结束，不需要处理。

输出：

每个测例在单独的一行内输出一个整数：Jerry先取输出1；Tom先取输出0。

输入样例：

1.5  
7.9  
-1

输出样例：

1  
0

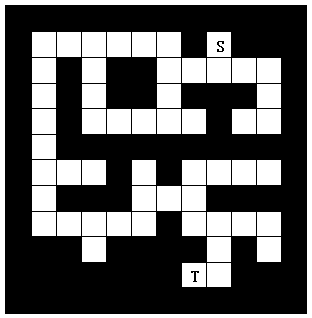
# 实验四

### Problem A

### 电子老鼠闯迷宫

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

有一只电子老鼠被困在如下图所示的迷宫中。这是一个12\*12单元的正方形迷宫，黑色部分表示建筑物，白色部分是路。电子老鼠可以在路上向上、下、左、右行走，每一步走一个格子。现给定一个起点S和一个终点T，求出电子老鼠最少要几步从起点走到终点。  


输入：

本题包含一个测例。在测例的第一行有四个由空格分隔的整数，分别表示起点的坐标S（x.y）和终点的坐标T（x,y）。从第二行开始的12行中，每行有12个字符，描述迷宫的情况，其中'X'表示建筑物，'.'表示路.

输出：

输出一个整数，即电子老鼠走出迷宫至少需要的步数。

输入样例：

2 9 11 8  
XXXXXXXXXXXX  
X......X.XXX  
X.X.XX.....X  
X.X.XX.XXX.X  
X.X.....X..X  
X.XXXXXXXXXX  
X...X.X....X  
X.XXX...XXXX  
X.....X....X  
XXX.XXXX.X.X  
XXXXXXX..XXX  
XXXXXXXXXXXX

输出样例：

28

#include<queue>

#include<iostream>

#define N 12

using namespace std;

char map[N][N];

int mark[N][N];

int startX,startY,endX,endY;

int isCanMove(int x, int y, int& newX, int& newY, int direction) {

int tempX = x;

int tempY = y;

switch(direction) {

case 0:

tempX--;break;

case 1:

tempX++;break;

case 2:

tempY--;break;

case 3:

tempY++;break;

}

newX = tempX;

newY = tempY;

if(tempX < 0 || tempX >= N || tempY < 0 || tempY >= N) {

return 0;

}

if(map[tempX][tempY] == '.') {

return 1;

}

return 0;

}

int isUesed(int x, int y) {

if(mark[x][y] == 0) {

return 0;

}

return 1;

}

int isAim(int x, int y) {

if( x == (endX-1) && y == (endY-1)) {

return 1;

}

return 0;

}

int search(char map[N][N], int start\_node) {

int x;

int y;

int newX;

int newY;

int num = 0;

queue<int> not\_yet\_explored;

not\_yet\_explored.push(start\_node);

mark[startX - 1][startY - 1] = 1;

while(!not\_yet\_explored.empty()) {

int node\_to\_explored = not\_yet\_explored.front();

not\_yet\_explored.pop();

x = node\_to\_explored / N;

y = node\_to\_explored % N;

num = mark[x][y];

for(int i = 0;i < 4;i++) {

if(isCanMove(x, y, newX, newY, i)) {

if(isAim(newX, newY)) {

return num;

}

if(!isUesed(newX, newY)) {

mark[newX][newY] = num + 1;

not\_yet\_explored.push(newX\*N + newY);

}

}

}

}

}

int main() {

cin >> startX >> startY >> endX >> endY;

for(int i = 0;i < N;i++) {

for(int j = 0;j < N;j++) {

cin >> map[i][j];

}

}

for(int row = 0;row < N;row++) {

for(int col = 0;col < N;col++) {

if(map[row][col] == '.') {

mark[row][col] = 0;

} else {

mark[row][col] = -2;

}

}

}

cout << search(map, (startX-1)\*N + startY-1) << endl;

return 0;

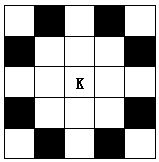
}

### Problem B

### 跳马

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

在国际象棋中，马的走法与中车象棋类似，即俗话说的“马走日”，下图所示即国际象棋中马（K）在一步能到达的格子（其中黑色的格子是能到达的位置）。  
  
现有一200\*200大小的国际象棋棋盘，棋盘中仅有一个马，给定马的当前位置（S）和目标位置（T），求出马最少需要多少跳才能从当前位置到达目标位置。

输入：

本题包含多个测例。输入数据的第一行有一个整数N（1<=N<=1000），表示测例的个数，接下来的每一行有四个以空格分隔的整数，分别表示马当前位置及目标位置的横、纵坐标C（x,y）和G（x,y）。坐标由1开始。

输出：

对于每个测例，在单独的一行内输出一个整数，即马从当前位置跳到目标位置最少的跳数。

输入样例：

2  
1 1 2 1  
1 5 5 1

输出样例：

3  
4

#include <stdio.h>

int f[8]={-409,-407,-206,-202,202,206,407,409};

int step(int s,int e)

{

int map[42000]={0};

int d[42000]={0},ds[42000]={0};

int start,endd,x,y,i;

for(i=0;i<204;i++)

{

map[i]=1;

map[i+204]=1;

map[i\*204]=1;

map[i\*204+1]=1;

map[i\*204+203]=1;

map[i\*204+202]=1;

map[204\*202+i]=1;

map[204\*203+i]=1;

}

start=0;

endd=1;

d[start]=s;

map[s]=1;

ds[start]=0;

while(start<endd)

{

x=d[start];

y=ds[start];

if(x==e)

break;

for(i=0;i<8;i++)

{

if(map[x+f[i]]==0)

{

map[x+f[i]]=1;

d[endd]=x+f[i];

ds[endd]=y+1;

endd++;

}

}

start++;

}

return(y);

}

int main()

{

int n,a,b,c,d,i;

int e[100]={0};

scanf("%d",&n);

for(i=0;i<n;i++)

{

scanf("%d%d%d%d",&a,&b,&c,&d);

e[i]=step((a+1)\*204+b+1,(c+1)\*204+d+1);

}

for(i=0;i<n;i++)

{

printf("%d\n",e[i]);

}

return(0);

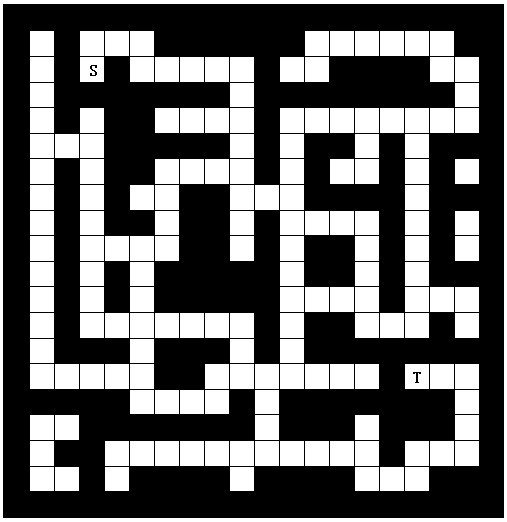
}

### Problem C

### 独轮车

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

独轮车的轮子上有红、黄、蓝、白、绿（依顺时针序）5种颜色，在一个如下图所示的20\*20的迷宫内每走一个格子，轮子上的颜色变化一次。独轮车只能向前推或在原地转向。每走一格或原地转向90度均消耗一个单位时间。现给定一个起点（S）和一个终点（T），求独轮车以轮子上的指定颜色到达终点所需的最短时间。  


输入：

本题包含一个测例。测例中分别用一个大写字母表示方向和轮子的颜色，其对应关系为：E-东、S-南、W-西、N-北；R-红、Y-黄、B-蓝、W-白、G-绿。在测试数据的第一行有以空格分隔的两个整数和两个大写字母，分别表示起点的坐标S（x,y）、轮子的颜色和开始的方向，第二行有以空格分隔的两个整数和一个大写字母，表示终点的坐标T（x,y）和到达终点时轮子的颜色，从第三行开始的20行每行内包含20个字符，表示迷宫的状态。其中'X'表示建筑物，'.'表示路.

输出：

在单独的一行内输出一个整数，即满足题目要求的最短时间。

输入样例：

3 4 R N  
15 17 Y  
XXXXXXXXXXXXXXXXXXXX  
X.X...XXXXXX......XX  
X.X.X.....X..XXXX..X  
X.XXXXXXX.XXXXXXXX.X  
X.X.XX....X........X  
X...XXXXX.X.XX.X.XXX  
X.X.XX....X.X..X.X.X  
X.X.X..XX...XXXX.XXX  
X.X.XX.XX.X....X.X.X  
X.X....XX.X.XX.X.X.X  
X.X.X.XXXXX.XX.X.XXX  
X.X.X.XXXXX....X...X  
X.X.......X.XX...X.X  
X.XXX.XXX.X.XXXXXXXX  
X.....XX.......X...X  
XXXXX....X.XXXXXXX.X  
X..XXXXXXX.XXX.XXX.X  
X.XX...........X...X  
X..X.XXXX.XXXX...XXX  
XXXXXXXXXXXXXXXXXXXX

输出样例：

56

#include<stdio.h>

struct colornode

{

int row;//该状态的行

int col;//该状态的列

int color;//该状态的颜色(五种颜色，0-R,1-Y,2-B,3-W,4-G)

int direction;//该状态的方向(四个方向，0-W,1-S,2-E,3-N,西南东北)

int num;//该状态的最小步数

};

struct colornode s,t;//起点，终点

struct colornode open[2000];//队列

int head,tail,openlen=2000;

int direct[4][2]={{0,-1},{1,0},{0,1},{-1,0}};//向左，下，右，上四个方向转时，行列的增加值

int a[20][20]={0},n=20;//a数组表示迷宫，n为迷宫边长

int b[20][20][5][4] = {0};//b数组表示搜索时的所有状态，0为未访问,1为已访问

void readdata();

void init();

int search();

struct colornode moveahead(struct colornode u);//u向前走一格得到的结点v

struct colornode turntoleft(struct colornode u);//u向左转得到新结点v

struct colornode turntoright(struct colornode u);//u向右转得到新结点v

int islegal(struct colornode v);

int isaim(struct colornode v);

int used(struct colornode v);//判断该点是否是已经到达过的结点

void addtoopen(struct colornode v);//将节点v加入队列

struct colornode takeoutofopen();//从队列头取节点数据

///////////////////////////////////////////////////////////////////////////////////////////////////////

int main()

{

readdata();

init();

int number=search();

printf("%d\n",number);

return 0;

}

///////////////////////////////////////////////////////////////////////////////////////

int search()

{

struct colornode u,v;

while(head != tail)

{

u = takeoutofopen();//从队列头取数据

//////////////////////////////向前走可能到目标

v = moveahead(u);//向前走

if(islegal(v))

{

if(isaim(v))

return(v.num);

else if(!used(v))

addtoopen(v);

}

//////////////////////////////向左，向右转都不会到目标点

v=turntoleft(u);//左转

if(!used(v))

addtoopen(v);

/////////////////////////////

v=turntoright(u);//右转

if(!used(v))

addtoopen(v);

}

return 0;//起始点没有通路时

}

struct colornode turntoright(struct colornode u)

{

struct colornode v;

v=u;

v.direction = (u.direction+3)%4;//只要改变方向

v.num = u.num+1;

return(v);

}

struct colornode turntoleft(struct colornode u)

{

struct colornode v;

v=u;

v.direction = (u.direction+1)%4;//只要改变方向

v.num = u.num+1;

return(v);

}

int used(struct colornode v)

{

if(b[v.row][v.col][v.color][v.direction]==0)

return 0;

else

return 1;

}

int isaim(struct colornode v)

{

if(v.row==t.row &&v.col==t.col &&v.color==t.color)

return 1;

else

return 0;

}

int islegal(struct colornode v)

{

if(v.row>=0 &&v.row<n &&v.col>=0 &&v.col<n &&a[v.row][v.col]==0)

return 1;

else return 0;

}

struct colornode moveahead(struct colornode u)

{

struct colornode v;

v.row=u.row+direct[u.direction][0]; //由方向决定行列增减值

v.col=u.col+direct[u.direction][1];

v.color = (u.color+1)%5;//颜色循环

v.direction = u.direction;//直走方向不变

v.num = u.num+1;//新的位置步长加1

return(v);

}

struct colornode takeoutofopen()

{

struct colornode v;

v=open[head++];

head=head%openlen;

return v;

}

void addtoopen(struct colornode v)

{

open[tail++]=v;

tail = tail%openlen;

b[v.row][v.col][v.color][v.direction]=1;//走过的点状态记为1

}

//////////////////////////////////////////////////////////////////////

void readdata()

{

char str[50];

int Berow,Becol;

char Becolor,Bedir;

scanf("%d %d %c %c",&Berow,&Becol,&Becolor,&Bedir);//注意%c之前一定要加空格

s.row = (Berow-1);

s.col = (Becol-1);

switch(Becolor)

{ case 'R':s.color=0;break;//红色块压地，值为0

case 'Y':s.color=1;break;

case 'B':s.color=2;break;

case 'W':s.color=3;break;

case 'G':s.color=4;break;

};

switch(Bedir)

{ case 'W':s.direction=0;break;//向左，方向值为0

case 'S':s.direction=1;break;

case 'E':s.direction=2;break;

case 'N':s.direction=3;break;

};

int Enrow,Encol;

char Encolor;

scanf("%d %d %c",&Enrow,&Encol,&Encolor);//注意%c之前一定要加空格

t.row = (Enrow-1);

t.col = (Encol-1);

switch(Encolor)

{ case 'R':t.color=0;break;

case 'Y':t.color=1;break;

case 'B':t.color=2;break;

case 'W':t.color=3;break;

case 'G':t.color=4;break;

};

int i=0,j=0;

for(i=0;i<n;i++)

{

scanf("%s",str);//输入一行迷宫字符，再转换成迷宫数字

for(j=0;j<n;j++)

{ if(str[j]=='X') a[i][j]=1;//1表示墙

else a[i][j]=0;//0表示空格

}

}

}

void init()

{

int i=0,j=0,k=0,l=0;

for(i=0;i<n;i++)//行

for(j=0;j<n;j++)//列

for(k=0;k<5;k++)//颜色

for(l=0;l<4;l++)//方向

b[i][j][k][l]=0;//状态

head=0; tail=0;

s.num=0;//在初始状态走的步数为0

addtoopen(s);

}

### Problem D

### 六数码问题

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

现有一两行三列的表格如下：  
  
A B C  
D E F  
  
把1、2、3、4、5、6六个数字分别填入A、B、C、D、E、F格子中，每个格子一个数字且各不相同。每种不同的填法称为一种布局。如下：  
  
1 3 5  
2 4 6  
布局1  
  
2 5 6  
4 3 1  
布局2  
  
定义α变换如下：把A格中的数字放入B格，把B格中的数字放入E格，把E格中的数字放入D格，把D格中的数字放入A格。  
定义β变换如下：把B格中的数字放入C格，把C格中的数字放入F格，把F格中的数字放入E格，把E格中的数字放入B格。  
  
问：对于给定的布局，可否通过有限次的α变换和β变换变成下面的目标布局：  
  
1 2 3  
4 5 6  
目标布局

输入：

本题有多个测例，每行一个，以EOF为输入结束标志。每个测例的输入是1到6这六个数字的一个排列，空格隔开，表示初始布局ABCDEF格中依次填入的数字。

输出：

每个输出占一行。可以转换的，打印Yes；不可以转换的，打印No。

输入样例：

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

输出样例：

No  
Yes

#include <stdio.h>

long turna(long n)

{

int m=0,i=1;

int a[7]={0};

while(n>0)

{

a[i]=n%10;

i++;

n=n/10;

}

m=a[3]\*100000+a[6]\*10000+a[4]\*1000+a[2]\*100+a[5]\*10+a[1];

return(m);

}

long turnb(long n)

{

int m=0,i=1;

int a[7]={0};

while(n>0)

{

a[i]=n%10;

i++;

n=n/10;

}

m=a[6]\*100000+a[2]\*10000+a[5]\*1000+a[3]\*100+a[1]\*10+a[4];

return(m);

}

int find(int a[10])

{

int t[66666]={0};

long n=0,m=1,i;

long d[10000]={0};

int s,e;

for(i=1;i<=6;i++)

{

n=n+a[i]\*m;

m=m\*10;

}

s=0;e=1;

t[n/10]=1;

d[s]=n;

i=0;

while(s<e)

{

n=d[s];

if(n==123456)

{

i=1;

break;

}

m=turna(n);

if(t[m/10]==0)

{

t[m/10]=1;

d[e]=m;

e++;

}

m=turnb(n);

if(t[m/10]==0)

{

t[m/10]=1;

d[e]=m;

e++;

}

s++;

}

return(i);

}

int main()

{

int a[10]={0};

while(scanf("%d%d%d%d%d%d",&a[6],&a[5],&a[4],&a[3],&a[2],&a[1])!=EOF)

{

if(find(a)==1)

printf("Yes\n");

else

printf("No\n");

}

return(0);

}

### Problem E

### 找倍数

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

对于每个输入的数字（如：2），则要求 给出一个由1，0构成的十进制整数，且该整数为输入数字的某个倍数，且是满足该条件的最小数（如2对应的10）。

输入：

数字n，n等于0时停止。

输出：

n的一个满足条件的最小倍数。

输入样例：

2  
0

输出样例：

10

#include<stdio.h>

#include<math.h>

int n,t,d,num[30];

void hjz(int m){

int i=0,j,x=0,a[30];

while(m!=0){

a[i]=m%2;

m=m/2;

i++;

}

d=i;

for(j=0;j<d;j++){

num[i-1]=a[j];

i--;

}

}

int check(){

int i,j,k,q;

k=0;

q=0;

for(i=0;i<d;i++){

k=num[i]+q\*10;

q=k%n;

}

if(q==0)

return 1;

else

return 0;

}

void search(int m){

int i;

hjz(m);

if(check()==1){

for(i=0;i<d;i++)

printf("%d",num[i]);

}

else

search(m+1);

}

int main()

{

scanf("%d",&n);

while(n!=0){

search(1);

printf("\n");

scanf("%d",&n);

}

return 0;

}

### Problem F

### 木乃伊迷宫

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

木乃伊地下宫殿是一个6行6列的迷宫。作为敢到木乃伊地下宫殿里去探险的你，有没有跟木乃伊抓迷藏的心理准备呵！游戏在木乃伊所在的迷宫里展开，任务就是尽快赶到出口。你一次只能走一步，而木乃伊可以走两步，但木乃伊是很笨的，他总是先尽量跟你达到同一列，如果已经是同一列了，他才会像你走来，有墙的地方人和木乃伊都不能过，你可以利用障碍物牵制住木乃伊。

输入：

先输入墙的数量n，然后在后续的n行里每行有3个数表示一堵墙，3个数分别为格子的行、列和墙的位置（0表示这个格子的下方是墙，1表示这个格子的右方是墙），再下来的3行每行2个数，分别表示木乃伊、人还有出口的位置。

输出：

    如果能安全逃生则输出Yes,否则输出No,答案占一行。

输入样例：

5  
0 0 0  
1 1 1  
1 4 1  
3 4 1  
4 3 0  
3 3  
3 1  
5 5

输出样例：

No

#include <iostream>

#include <math.h>

#include <queue>

int ma[6][6],rr=3,rc=1,mr=3,mc=3,cr=5,cc=5,s=0,f=0,num[6][6][6][6],b[4][2]={

0,1,

0,-1,

1,0,

-1,0

};

using namespace std;

struct mu{

int rrow;

int rcol;

int mrow;

int mcol;

};

int isGo(mu m,int i){

if(i==0){

if(ma[m.rrow][m.rcol]==2)

return 0;

}

if(i==2){

if(ma[m.rrow][m.rcol]==1){

return 0;

}

}

return 1;

}

mu go(mu m,int i){

m.rrow+=b[i][0];

m.rcol+=b[i][1];

return m;

}

mu mgo(mu m){

int n=2;

while(n!=0){

n--;

if(m.rcol>m.mcol){

if(ma[m.mrow][m.mcol]!=2)

m.mcol++;

}

else{

if(m.rcol<m.mcol){

m.mcol--;

}

else {

if(m.rcol==m.mcol){

if(m.mrow<m.rrow)

{ if(ma[m.mrow][m.mcol]!=1)

m.mrow++;

}

else{

if(m.mrow>m.rrow){

m.mrow--;

}

else{

if(m.mrow==m.rrow)

s=1;

}

}

}

}

}

}

return m;

}

int isAim(mu m){

if(m.rrow==cr && m.rcol==cc)

return 1;

else

return 0;

}

int iskeyi(mu m){

if(m.rrow<6 && m.rrow>=0 && m.rcol<6 && m.rcol>=0)

return 1;

else

return 0;

}

int isSafe(mu m){

if(m.mcol==m.rcol && m.mrow==m.rrow)

return 0;

else

return 1;

}

int isNew(mu m){

if(num[m.rrow][m.rcol][m.mrow][m.mcol]==1)

return 0;

else

return 1;

}

void fd(){

mu fir,m,newm;

queue<mu> que;

int i;

fir.rcol=rc;

fir.rrow=rr;

fir.mrow=mr;

fir.mcol=mc;

que.push(fir);

while(!que.empty()){

m=que.front();

que.pop();

for(i=0;i<4;i++){

if(isGo(m,i)==0)

continue;

newm=go(m,i);

if(isAim(newm)==1){

f=1;

return;

}

if(iskeyi(newm)==1){

newm=mgo(newm);

if(isSafe(newm)==1){

if(isNew(newm)==1){

num[newm.rrow][newm.rcol][newm.mrow][newm.mcol]=1;

que.push(newm);

}

}

}

}

}

}

int main()

{

int i,n,a,b,c;

cin>>n;

for(i=0;i<n;i++){

cin>>a>>b>>c;

if(c==0)

ma[a][b]=2;

else

ma[a][b]=1;

}

cin>>mr>>mc;

cin>>rr>>rc;

cin>>cr>>cc;

fd();

if(rr==cr && rc==cc)

f=1;

if(mr==rr && mc==rc)

f=0;

if(f==1)

cout<<"Yes"<<endl;

else

cout<<"No"<<endl;

return 0;

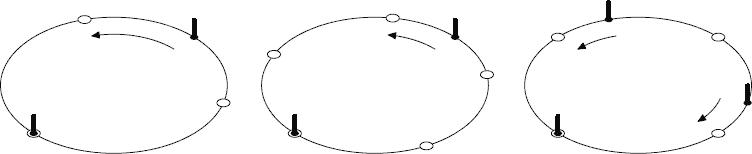
}

### Problem G

### polygon

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

在一个周长为10000的圆上等距分布着n个点，即这n个点是一个正n边形的顶点。现在要另加m个点到圆上，新加的m个点可以任意选择位置（可以与原有的点重合）。然后将这n+m个点中的一些点延圆周移动，最终使n+m个点均匀分布，即在一个正n+m边形的顶点上。输出最小总移动距离。

输入：

输入两个整数 n, m。 (2≤n≤1000, 1≤m≤1000).

输出：

输出最小总移动距离，保留4位小数。

输入样例：

sample input #1

2 1

sample input #2

2 3

sample input #3

3 1

sample input #4

10 10

输出样例：

sample output #1

1666.6667

sample output #2

1000.0

sample output #3

1666.6667

sample output #4

0.0

图对应前3个样例

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

int n, m;

double Min(double a, double b)

{

return (a < b ? a : b);

}

double FindSimilarOrEqual(double mod[], double ori[])

{

int begin = 0, end = n + m - 1, mid;

double sum = 0.0;

for (int i = 1; i < n; ++i)

{

while (begin <= end)

{

mid = (begin + end) / 2;

if (mod[mid] == ori[i])

break;

else if (mod[mid] > ori[i])

end = mid - 1;

else

begin = mid + 1;

}

if (begin > end)

sum += Min(fabs(mod[begin] - ori[i]), fabs(mod[end] - ori[i]));

begin = 0;

end = n + m - 1;

}

return sum;

}

int main(void)

{

int i;

double ori\_part, mod\_part;

cin >> n >> m;

double \*ori = new double[n];

double \*mod = new double[n + m];

ori\_part = 10000 / (double)n;

mod\_part = 10000 / (double)(n + m);

for (i = 0; i < n; ++i)

ori[i] = i \* ori\_part;

for (i = 0; i < n + m; ++i)

mod[i] = i \* mod\_part;

cout << setiosflags(ios::fixed) << setprecision(4);

cout << FindSimilarOrEqual(mod, ori) << endl;

delete []ori;

delete []mod;

return 0;

}

### Problem H

### 推箱子

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

绝大多数人都玩过推箱子的游戏，控制一个人将箱子推动到目标位置即获得胜利。现请你编写一个程序，判断将箱子推到目标位置至少需要多少步。

输入：

推箱子的平面区域为固定大小（10\*10），使用10行10列输入推箱子的初始局面。其中，0代表空格，1代表墙，2代表箱子，3代表目标位置，4代表人。  
注：游戏中只有一个箱子，一个目标位置，一个人。

输出：

输出将箱子推到目标位置的最小步数；若箱子不可能被推到目标位置，输出-1。

输入样例：

0000000000  
0000000300  
0100000000  
0100000000  
0101111100  
0000010000  
0000010000  
0020010040  
0000010000  
0000010000

输出样例：

34

### Problem I

### 特殊的二阶魔方

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

魔方大家应该都玩过。现在有一个特殊的二阶魔方，它只有一面是白色，其余五个面全是黑色。玩这个魔方当然也有特殊的规则，玩家只能通过六种方式去改变它，底层向左转一格（称为DL），底层向右转一格（称为DR），右侧向上转一格（称为RU），右侧向下转一格（称为RD），内侧顺时针转一格（称为C），内侧逆时针转一格（称为CC）。现给一魔方的状态，请在最少的步骤内把魔方还原

输入：

按照上下左右前后的顺序给出各面的具体情况，0表示白色，1表示黑色。上下、左右、前后分别是以俯视图、左视图、正视图看到的

输出：

输出令一面全为白色的最小步数。

输入样例：

00  
00  
11  
11  
11  
11  
11  
11  
11  
11  
11  
11

输出样例：

0