**滑动窗口的最大值**

问题描述

给出一个可能包含重复的整数数组，和一个大小为 k 的滑动窗口。从左到右在数组中滑动这个窗口，编程找到数组中每个窗口内的最大值。

输入

有多组测试数据。每组有2行，第1行有两个整数n和k；第2行有n个整数。

输出

对每组测试数据，依次输出每一个滑动窗口内的最大值的数组。

输入样例

5 3

1 2 7 7 8

输出样例

7 7 8

代码实现

#include <deque>

#include <iostream>

using namespace std;

//Get the max in A[0] - A[size-1]

int getMax(int \*A, int size){

int mx = A[0];

for(int i=1; i<size; i++){

if(A[i] > mx)

mx = A[i];

}

return mx;

}

//Order solution get the max in windows move

void maxInWindows(int \*A, int n, int k, int \*B){

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

B[i] = getMax(A+i,k);

}

// Bidirectional Queue Implementation

void maxSlidingWindow(int \*A, int n, int k, int \*B){

deque<int> Q;

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

while (!Q.empty() && A[i] >= A[Q.back()])

Q.pop\_back();

Q.push\_back(i);

}

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

B[i-k] = A[Q.front()];

while (!Q.empty() && A[i] >= A[Q.back()])

Q.pop\_back();

while (!Q.empty() && Q.front() <= i-k)

Q.pop\_front();

Q.push\_back(i);

}

B[n-k] = A[Q.front()];

}

int main(int argc, char \*argv[]){

int i,array\_size,window\_size;

cout<< "Please Enter your array size and window length: ";

cin>> array\_size >> window\_size;

int \*A = new int[array\_size];

int \*B = new int[array\_size - window\_size + 1];

for (i=0; i<array\_size; i++)

cin >> A[i];

maxInWindows(A, array\_size, window\_size, B);

for (i=0; i<array\_size - window\_size + 1; i++)

cout<< B[i];

cout<< endl;

maxSlidingWindow(A, array\_size, window\_size, B);

for (i=0; i<array\_size - window\_size + 1; i++)

cout<< B[i];

cout<< endl;

}

**最小差**

问题描述

若给定两个有序整数数组（第一个是数组 A，第二个是数组 B），在数组 A 中找 A[ i ]，数组 B 中找B[ j ]，使A[ i ] 和 B[ j ] 两者的差取得最小。

编程求最小差。

要求：时间复杂度 O(n log n)。

输入

有多组测试数据，每组有3行。

每组的第1行是一个整数，（n<10000）。接着有2行，每行上有个数为n的整数序列，分别表示数组A和B。

输出

对每组测试数据中的两个整数数组 A 和 B，输出所要求的最小差。

输入样例

4

3 4 6 7

2 3 8 9

输出样例

0

代码实现

#include <math.h>

#include <iostream>

using namespace std;

// A[],B[] are two ordered arrays. n is the length of arrays.

int min\_dist(int \*A, int \*B, int n){

int min\_distance = fabs(A[0] - B[0]);

int low, high, middle, target;

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

low = 0;

high = n - 1;

middle = 0;

target = A[i];

while (low <= high){

middle = low + (high - low) / 2;

if (B[middle] == target){

min\_distance = 0;

break;

}

else if (B[middle] < target){

low = middle + 1;

}

else{

high = middle - 1;

}

}

if(min\_distance == 0)

break;

if(fabs(A[i] - B[middle]) < min\_distance)

min\_distance = fabs(A[i] - B[middle]);

if(middle > 0)

if (fabs(A[i] - B[middle - 1]) < min\_distance)

min\_distance = fabs(A[i] - B[middle - 1]);

if(middle < n - 1)

if (fabs(A[i] - B[middle + 1]) < min\_distance)

min\_distance = fabs(A[i] - B[middle + 1]);

}

return min\_distance;

}

int main(int argc, char \*argv[]){

int i,array\_size,min\_distance;

cout<< "Please Enter your array size: ";

cin>> array\_size;

int \*A = new int[array\_size];

int \*B = new int[array\_size];

for (i=0; i<array\_size; i++)

cin >> A[i];

for (i=0; i<array\_size; i++)

cin >> B[i];

min\_distance = min\_dist(A,B,array\_size);

cout << "The min distance between array A[] and B[] is " << min\_distance << " !"<<endl;

}

**新的区间列表**

问题描述

给出一个无重叠的按照区间起始端点排序的区间列表。在列表中插入一个新的区间，你要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。编程实现这个功能。

输入

有多组测试数据，每组有3行。每组的第1行是一个整数n，表示原始区间数目，第2行有2n个整数，表示是原始区间列表，其中每两个相邻的整数表示一个区间。第3行是2个整数，表示是待插入的区间端点。

输出

对每组测试数据，输出插入后形成的新的区间列表，每两个相邻的整数表示一个区间。

输入样例

2

1 2 5 9

2 5

输出样例

1 9

代码实现

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

int main(){

vector <pair<int,int>> A;

vector <pair<int,int>> C;

pair <int,int> B;

int n;

int start\_tmp,end\_tmp;

int insert\_pos;

while(cin >> n){

insert\_pos = 0;

A.clear();

C.clear();

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

cin>>start\_tmp>>end\_tmp;

A.push\_back(make\_pair(start\_tmp,end\_tmp));

//cout<<start\_tmp<<" "<<end\_tmp<<endl;

}

cin>>B.first>>B.second;

//cout<<B.first<<" "<<B.second<<endl;

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

//cout<<A[i].first<<" "<<A[i].second<<endl;

if(A[i].second < B.first){

C.push\_back(A[i]);

insert\_pos++;

}

else if(A[i].first > B.second){

C.push\_back(A[i]);

}

else{

B.first = min(A[i].first,B.first);

B.second = max(A[i].second,B.second);

}

}

C.insert(C.begin()+insert\_pos,B);

for(int i=0; i<C.size(); i++){

if(i) cout<<" ";

cout << C[i].first <<" "<< C[i].second;

}

cout<<endl;

}

return 0;

}

**区间最小数**

问题描述

给定一个整数数组A，及对于一个宽带长的查询区间[ start, end ]，求在该查询区间范围中的最小数。这里start、end表示对A查询的开始与结束的序号（A的元素序号从0开始）。

输入

有多组测试数据。每组是一个带查询的整数数组和要查询的区间数组。

每组的第一行是2个正整数n和q，第2行有n个整数，构成序列A，第3行有2q个整数，每2个整数q2i、q2i+1表示一个查询区间[q2i、q2i+1]。

输出

对于每组测试数据，及q个查询列表，输出一行，即依次输出序列A在q个查询区间中数的最小值序列。

挑战：每次查询在O(log n)的时间内完成。

输入样例

5 3

1 2 7 8 5

1 2 0 4 2 4

输出样例

2 1 5

代码实现

#include <iostream>

using namespace std;

int search(int \*A, int start, int end){

int min = A[start];

for(int i=start+1; i<=end; i++){

if(min>A[i])

min = A[i];

}

return min;

}

int main(){

int n,q;

while(cin>>n>>q){

int\* A = new int[n];

int\* B = new int[2\*q];

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

cin>>A[i];

for(int i=0; i<2\*q; i++)

cin>>B[i];

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

int start = B[2\*i];

int end = B[2\*i+1];

//cout << start << " " <<end <<endl;

int result = search(A,start,end);

if(i) cout<<" ";

cout<<result;

}

cout<<endl;

delete[] A;

delete[] B;

}

return 0;

}

**能否得到0**

问题描述

给定3个数，请判定能否用加减乘除运算符号（不用括号）连起来得到结果0。

输入

第一行是一个整数T，表示测试数据的个数。

接着有T行，每行有3个整数，表示一组测试数据。

输出

对每组测试数据，输出一个判定结果。如果有这样的表达式，那么输出“Yes”，否则输出“No”。

输入样例

3

2 3 1

5 4 3

输出样例

Yes

No

代码实现

在一个二维数组中，每一行都按照从左到右递增的顺序排序，每一列都按照从上到下递增的顺序排序。请完成一个函数，输入这样的一个二维数组和一个整数，判断数组中是否含有该整数。

//从右上角开始搜索

class Solution {

public:

bool Find(int target, vector<vector<int> > array) {

int rowCount = array.size();//行

int colCount = array[0].size();//列

int i,j;

for(i=0,j=colCount-1;i<rowCount&&j>=0;)

{

if(target == array[i][j])

return true;

if(target > array[i][j])

{

i++;

continue;//continue：终止当前循环中的一次操作，继续执行循环；break：直接跳出当前循环，执行循环外的语句。

}

if(target < array[i][j])

{

j--;

continue;

}

}

return false;

}

};

请实现一个函数，将一个字符串中的空格替换成“%20”。例如，当字符串为We Are Happy.则经过替换之后的字符串为We%20Are%20Happy。

//采用从后向前的替换方式。

//若采用从前往后的替换方式，则找到的第一个空格位置，则其后所有的字符均要后移。

//比如，若第一个空格在第5个位置，指针p值向它，若要将其替换成“%20”，则需要：

// \*p = "%";

// \*(p +1) = "2";

// \*(p +2) = "0";

//但这里，\*(p +1)和\*(p +2)原本就指向字符串中的后续字符，直接按照上述代码操作会丢失这部分的信息，则需要先保存p位置所有的字符（将这些字符后移）。

//若采用从后往前的替换方式，则找到最后一个空格后，只需将该空格后的字符移动到最终位置。故而采取这种方案。

//思路：先获取空格数。再从后往前遍历寻找空格进行替换。

class Solution {

public:

void replaceSpace(char \*str,int length) {

int count = 0;

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

if(str[i] == ' ')

count++;

}

for(int i = length-1; i>=0; i--){

if(str[i] != ' ')

str[i+count\*2] = str[i];

else{

count --;

str[i+count\*2] = '%';

str[i+count\*2+1] = '2';

str[i+count\*2+2] = '0';

}

}

}

};

输入一个链表，从尾到头打印链表每个节点的值。

/\*\*

\* struct ListNode {

\* int val;

\* struct ListNode \*next;

\* ListNode(int x) :

\* val(x), next(NULL) {

\* }

\* };

\*/

class Solution {

public:

vector<int> printListFromTailToHead(ListNode\* head) {

vector<int> value;

if(head != NULL){

value.insert(value.begin(),head->val);

if(head->next != NULL){

vector<int> tmpVec = printListFromTailToHead(head->next);

if(tmpVec.size() > 0)

value.insert(value.begin(),tmpVec.begin(),tmpVec.end());

}

}

return value;

}

};

**硬币问题：**

给予不同面值的硬币若干种，如何利用若干种硬币组合为某种面额的钱，使硬币个数最少。

代码实现

#include<iostream>

using namespace std;

//money需要找零的钱

//coin可用的硬币硬币种类

void FindMin(int money,int \*coin, int n)

{

int \*coinNum=new int[money+1]();//存储1...money找零最少需要的硬币的个数

int \*coinValue=new int[money+1]();//最后加入的硬币，方便后面输出是哪几个硬币

coinNum[0]=0;

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

{

int minNum=i;//i面值钱，需要最少硬币个数

int usedMoney=0;//这次找零，在原来的基础上需要的硬币

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

{

if(i>=coin[j])//找零的钱大于这个硬币的面值

{

//if(coinNum[i-coin[j]]+1<=minNum)//所需硬币个数减少了

/\*

上面的判断语句有问题，在更新时，需要判断i-coin[j]是否能找的开，如果找不开，就不需要更新。

多谢zywscq 指正

\*/

if(coinNum[i-coin[j]]+1<=minNum&&(i==coin[j]||coinValue[i-coin[j]]!=0))//所需硬币个数减少了

{

minNum=coinNum[i-coin[j]]+1;//更新

usedMoney=coin[j];//更新

}

}

}

coinNum[i]=minNum;

coinValue[i]=usedMoney;

}

//输出结果

if(coinValue[money]==0)

cout<<"找不开零钱"<<endl;

else

{

cout<<"需要最少硬币个数为："<<coinNum[money]<<endl;

cout<<"硬币分别为:";

while(money>0)

{

cout<<coinValue[money]<<",";

money-=coinValue[money];

}

}

delete []coinNum;

delete []coinValue;

}

int main()

{

int Money=18;

int coin[]={1,2,5,9,10};

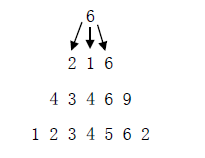
FindMin(Money,coin,5);

}

**数字三角形-3支**

问题描述

给定每行由数字组成的数字三角形，寻找从顶至低某处的一条路径，使该路径所经过的数字的总和最大，每一步只能向左下、向下或向右下走，如图所示。



输入

输入有若干测试数据。每一组测试数据的第一行为一个整数H，表示有高度为H的数字三角形，接下来的H行描述一个数字三角形，其中的第i行有2i-1个整数，整数之间用一个空格分开。

输出

对每组测试数据输出对应的数字三角形从顶至低边某处的一条最优路径所经过的最大数字之和。

输入样例

4

6

2 1 6

4 3 4 6 9

1 2 3 4 5 6 2

5

7

3 8 6

8 1 0 4 6

2 7 4 4 3 5 3

4 5 2 6 5 8 3 9 3

输出样例

27

33

算法实现

#include <iostream>

using namespace std;

const int H = 1000;

int A[H][H];

int main(){

int h;

while(cin>>h){

for(int i=1; i<=h; i++){

for(int j=1; j<=2\*i-1; j++){

cin>>A[i][j];

}

}

//动态规划问题

for(int i=h-1; i>=1; i--){

for(int j=1; j<=2\*i-1; j++){

A[i][j] += max(A[i+1][j],max(A[i+1][j+1],A[i+1][j+2]));

//cout<<A[i][j]<<endl;

}

}

cout<<A[1][1]<<endl;

}

return 0;

}

**点与三角形**

问题描述

给定点P和三角形ABC，确定点P与三角形的位置关系，并求三角形面积。

输入

输入的第一行上有正整数n，表示有n组测试数据，接下来的2n行是n组测试数据的描述。每组测试数据有2行：第一行上有6个整数x1 y1 x2 y2 x3 y3，整数之间用一个空格隔开，他们分别表示三角形ABC的三个顶点坐标A(x1，y1)、B(x2，y2)、C(x3，y3)；第二行是2个整数u，v，整数之间用一个空格隔开，它们表示点P的坐标P(u,v)。

输出

对输入的每组测试数据，在输出文件中先输出一行，内容是“Case i:”，其中i是数组的编号（从1开始）。在下一行上输出三角形的面积（四舍五入保留1位小数），空1格，接着对给定的三角形ABC和点P，确定点P与三角形ABC的位置关系。如果点P在三角形内部（应在真正内部），那么输出“Yes”；否则输出“No”。

输入样例

3

0 0 3 4 0 4

0 2

0 0 3 4 0 4

0 -3

0 0 3 4 0 4

1 2

输出样例

Case 1:

6.0 No

Case 2:

6.0 No

Case 3:

6.0 Yes

代码实现：

#include <iostream>

#include <iomanip>

#include <vector>

#include <utility>

#include <math.h>

using namespace std;

void cal(vector<pair<int,int>> triangle, pair<int,int> point){

pair<int,int>a,b,c,p;//向量AB、AC、BC、AP

float u,v,area,al,bl,cl,l;

string inside;

a.first = triangle[1].first - triangle[0].first;

a.second = triangle[1].second - triangle[0].second;

b.first = triangle[2].first - triangle[0].first;

b.second = triangle[2].second - triangle[0].second;

c.first = triangle[2].first - triangle[1].first;

c.second = triangle[2].second - triangle[1].second;

p.first = point.first - triangle[0].first;

p.second = point.second - triangle[0].second;

//海伦公式求面积

al = sqrt(a.first\*a.first + a.second\*a.second);

bl = sqrt(b.first\*b.first + b.second\*b.second);

cl = sqrt(c.first\*c.first + c.second\*c.second);

l = (al+bl+cl)/2;

area = sqrt(l\*(l-al)\*(l-bl)\*(l-cl));

//盘点点与三角形的位置关系

u = (p.first\*a.second - a.first\*p.second)/(float)(b.first\*a.second - a.first\*b.second);

v = (p.first\*b.second - b.first\*p.second)/(float)(a.first\*b.second - b.first\*a.second);

if(u>0 && v>0 && u+v<1)

inside = "Yes";

else

inside = "No";

cout<< setiosflags(ios::fixed) <<setprecision(1) << area << " " << inside << endl;

}

int main(){

int n,ver\_x,ver\_y,point\_x,point\_y;

int i = 1;

vector<pair<int,int>> triangle;

pair<int,int> point;

cin >> n;

while(i<=n){

for(int i=0; i<3; i++){

cin >> ver\_x >> ver\_y;

triangle.push\_back(make\_pair(ver\_x,ver\_y));

}

cin>>point.first>>point.second;

cout<<"Case " << i << ":" <<endl;

cal(triangle,point);

i++;

}

return 0;

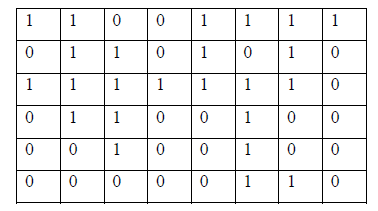
}

**是否一个海岛**

问题描述

输入一个长方形表示海域G，判定G中是否只有一个海岛。区域中每个方格位置用0或1表示，其中1表示是高于海平面的小区域，0表示低于海平面的小区域。水平或垂直直接相邻的两个方格称为区域相邻，其他位置的方格不是直接相邻的，高出海平面的连通区域构成海岛。

现在要求你编写一个程序，判断该长方形区域中是否只有一个海岛。如下图所示是一个6×8的海域，有一个海岛。



输入

有多个海域图数据。

每个海域图描述的第1行是两个整数m和n，（0<m，n<100）,分别表示海域的行数和列数。接着有m行，每行有n个数字，每个数字是0或1，之间用一个空格分开。

输出

对每个海域图，如果是里面只有一个海岛，那么输出“Yes”，否则输出“No”。

输入样例

6 8

1 1 0 0 1 1 1 1

0 1 1 0 1 0 1 0

1 1 1 1 1 1 1 0

0 1 1 0 0 1 0 0

0 0 1 0 0 1 0 0

0 0 0 0 0 1 1 0

输出样例

Yes

代码实现

#include <iostream>

using namespace std;

const int N = 100;

int mp[N][N];

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

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

int n,m;

bool vis[N][N];

void dfs(int r, int c){

vis[r][c] = true;

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

int nr = r+dx[i];

int nc = c+dy[i];

if(nr>=0 && nr<m && nc>=0 && nc<n && !vis[nr][nc] && mp[nr][nc]==1){

dfs(nr,nc);

}

}

}

int main(){

while(cin >> m >> n){

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

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

cin >> mp[i][j];

vis[i][j] = false;

}

}

int res = 0;

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

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

if(mp[i][j]==1 && !vis[i][j]){

res++;

dfs(i,j);

}

}

}

if(res == 1)

cout<< "Yes" << endl;

else

cout << "No" << endl;

}

return 0;

}

**卷积最大值子采样**

问题描述

给出一个可能包含重复整数的m\*n数组，和一个大小为k\*k卷积核，现用该核进行最大值采样，即滑动大小为k\*k的窗口，从左上角开始，自左至右，自上而下，在m\*n数组中滑动这个窗口，取出这个窗口中的最大值。

编程找到m\*n数组中每个窗口内的最大值构成的矩阵。

输入

有多组测试数据。每组有2行，第1行有3个整数m、n和k；第2行开始有m行，每行n个整数。（0<m,n<=100, k<=m, k<=n）。

输出

对每组测试数据，先输出“Case #”，下一行上依次输出m\*n数组中每个窗口内的最大值构成的矩阵。

输入样例

5 5 3

7 1 1 0 0

0 1 1 1 0

0 0 4 1 1

0 0 1 1 8

6 1 1 0 0

3 4 2

94 48 91 16

35 95 89 92

98 99 74 31

输出样例

Case 1

7 4 4

4 4 8

6 4 8

Case 2

70 99 99

92 99 99

92 99 99

代码实现

#include <iostream>

using namespace std;

//获取0-m,0-n内的最大值

//int getMax(vector<vector<int>> A, int n){

int getMax(int \*A, int k){

int mx = A[0];

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

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

if(A[i\*k+j]>mx)

mx = A[i\*k+j];

}

}

return mx;

}

//获取k\*k的数组

void getArray(int \*A, int index\_i, int index\_j, int n, int k, int \*B){

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

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

B[i\*k+j] = A[index\_i\*n+index\_j+i\*n+j];

}

}

}

int main(){

int m,n,k;

int mx;

int i=1;

while(cin >> m >> n >> k){

int \*A = new int[m\*n];

int \*B = new int[k\*k];

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

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

cin>>A[i\*n+j];

}

}

cout<< "Case " << i << endl;

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

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

getArray(A,i,j,n,k,B);

mx = getMax(B,k);

if(j!=0 && j!=n-k+1) cout<<" ";

cout<< mx;

}

cout << endl;

}

i++;

}

return 0;

}

**特殊物品装包**

问题描述

现有n个特殊的物品需要装入容量为c的背包，但这n个物品重量越轻价值越大，即设这n个物品的第i个的体积为wi，价值为vi（i=1,2,……,n），假如按重量排序，不妨设，w1≤w2≤...≤wn,w2，那么此时有v1≥ v2≥…≥vn\。如“金、银、铜、铁”等物品就是这种情况。

现在您的任务是根据物品的重量和价值，给出一个将物品放入背包的方案使得包中物品的价值最大。注意：物品不能切割，也即要么整块放入、要么不放。

输入

输入有若干组测试数据(不超过20组)。

每组测试数据有3行：其第1行上有2个整数n和c，分别是物品个数n和包所能容纳物品的体积，（n<=50,c<=500），第2行上有n个整数v1、v2、…、vn，依次是n个物品的价值，第3行上有n个整数w1、w2、…、wn,，分别是n个物品的重量。诸整数之间用一个空格分开。

输出

现对输入中的每组测试数据，输出1行：先输出“Case #:”，其中“#”是测试数据的组号（从1开始）。接着输出包能装的物品的最大价值。

输入样例

3 4

1 3 4

3 2 1

5 20

6 3 5 4 2

2 6 3 5 8

输出样例

Case 1: 7

Case 2: 18

代码实现

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

vector<pair<int,int>> getLightest(vector<pair<int,int>> v){

pair<int,int> c;

for(int i=0; i<v.size()-1; i++ ){

for(int j=0; j<v.size()-i-1; j++){

if(v[j].first>v[j+1].first){

c = v[j];

v[j] = v[j+1];

v[j+1] = c;

}

}

}

return v;

}

int main(){

int n,c,wei,val,k;

int j = 1;

while(cin >> n >> c){

wei = 0;

val = 0;

int \*value = new int[n];

int \*weight = new int[n];

vector<pair<int,int>> mat,order\_mat;

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

cin >> value[i];

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

cin >> weight[i];

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

mat.push\_back(make\_pair(weight[i],value[i]));

order\_mat = getLightest(mat);

k = 0;

while(wei < c){

wei = wei + order\_mat[k].first;

if(wei > c) break;

val = val + order\_mat[k].second;

k++;

}

cout << "Case " << j << ": " << val << endl;

j++;

}

return 0;

}

**点与四边形**

问题描述

给定点P和四边形ABCD，确定点P与四边形的位置关系。

输入

输入的第一行上有正整数n，表示有n组测试数据，每组数据有2行。

每组的第一行上有8个整数x1、y1、x2、y2、x3、y3、x4、y4，整数之间用一个空格隔开，他们分别表示四边形ABCD的四个顶点坐标A(x1，y1)、B(x2，y2)、C(x3，y3)、D(x4、y4)；第二行是2个整数u、v，整数之间用一个空格隔开，他们表示一个点的坐标P(u,v)。

输出

对输入的每组测试数据，先输出一行，内容是“Case i:”，其中i是数组的编号（从1开始）。接着对给定的四边形ABCD和点P，确定点P与四边形ABCD的位置关系。如果点P在四边形内部或边上（包括端点），那么输出“In”；否则输出“Out”。

输入样例：

2

0 0 3 4 0 4 -1 1

7 4

0 0 3 4 0 4 -1 1

0 2

输出样例：

Case 1:

Out

Case 2:

In

代码实现

#include <iostream>

#include <vector>

#include <utility>

using namespace std;

void cal(vector< pair<int,int> > triangle, pair<int,int> point){

pair<int,int>ab,ap,bc,bp,cd,cp,da,dp;

int a,b,c,d;

string inside;

//AB

ab.first = triangle[1].first - triangle[0].first;

ab.second = triangle[1].second - triangle[0].second;

//AP

ap.first = point.first - triangle[0].first;

ap.second = point.second - triangle[0].second;

//BC

bc.first = triangle[2].first - triangle[1].first;

bc.second = triangle[2].second - triangle[1].second;

//BP

bp.first = point.first - triangle[1].first;

bp.second = point.second - triangle[1].second;

//CD

cd.first = triangle[3].first - triangle[2].first;

cd.second = triangle[3].second - triangle[2].second;

//CP

cp.first = point.first - triangle[2].first;

cp.second = point.second - triangle[2].second;

//DA

da.first = triangle[0].first - triangle[3].first;

da.second = triangle[0].second - triangle[3].second;

//DP

dp.first = point.first - triangle[3].first;

dp.second = point.second - triangle[3].second;

a = (ab.first\*ap.second - ab.second\*ap.first);

b = (bc.first\*bp.second - bc.second\*bp.first);

c = (cd.first\*cp.second - cd.second\*cp.first);

d = (da.first\*dp.second - da.second\*dp.first);

//cout<< "a= " << a << " b= " << b << " c= " << c << " d= " << d <<endl;

if((a >= 0 && b >= 0 && c >= 0 && d >= 0) || (a <= 0 && b <= 0 && c <= 0 && d <= 0))

cout<< "In" << endl;

else

cout<< "Out" << endl;

}

int main(){

int n,ver\_x,ver\_y,point\_x,point\_y;

int i = 1;

vector< pair<int,int> > rectangle;

pair<int,int> point;

cin >> n;

while(i<=n){

rectangle.clear();

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

cin >> ver\_x >> ver\_y;

rectangle.push\_back(make\_pair(ver\_x,ver\_y));

}

cin>>point.first>>point.second;

cout<<"Case " << i << ":" <<endl;

cal(rectangle,point);

i++;

rectangle.clear();

}

return 0;

}