**АЛГОРИТМИЧНА ГЕОМЕТРИЯ**

Б. Банчев /обучение в Хасково/, Г. Момчева, Пл. Христова, К. Григорова,   
Г. Панайотова, Стоян Христов, stancho.roncho.net, judge.openfmi.net, Уикипедия и др.

**КООРДИНАТНИ СИСТЕМИ. КООРДИНАТИ НА ТОЧКИ.**

**1. Координатни системи в равнината**

**а/ правоъгълна /декартова, ортогонална/ координатна система**

Две взаимно перпендикулярни оси, като се посочва коя то тях е първа и коя – втора, пресичащи се в т. О, наречена координатно начало, и линейна единица за измерване на дължина.

**О**

Оx

Оy

**M**

**M2**

**M1**

Всяка точка М в равнината се определя от наредена двойка числа, наречени декартови координати – абсциса и ордината, записва се т. М (х, у).

Абсцисата х на т. М е алгебричната мярка на насочената отсечка ОМ1, т.е х = 

Ординатата у на т. М е алгебричната мярка на насочената отсечка ОМ2, т.е у = 

Ох

Оy

**M**

На екрана координатна система се представя по малко по-различен начин – центърът е в горния ляв ъгъл на екрана и оста Оу е надолу, тъй като така се разполага текста.

**б/полярна координатна система** – определя се от една фиксирана точка О и ос р, минаваща през точка О. Точка О се нарича полюс на координатната система, а оста р – полярна ос.

**О**

р

**M**

**ϕ**

**ρ**

Всяка точка М в равнината се определя от наредена двойка числа ρ и ϕ, наречени полярни координати, където ρ е дължината на вектора  , а ϕ - ъгълът, между pи , отчитан в положителна посока в интервала [0, 2π) и измерван в радиани. Означаваме т. М (ρ, ϕ).

**2.Преминаване от една координатна система в друга**

**а/ от полярна в декартова координатна система** – през т.О построяваме ос Оу ⊥ Ох. Тогава координатите на М са алгебричните стойности на проекциите на вектор  върху тези оси.

т.е. ако М (ρ, ϕ), тогава х = ρ.cosϕ, ay = ρ.sinϕ,

**б/ преминаване от декратова в полярна координатна система** – т. О е полюс, а Ох е полярната ос и ако е дадена т. М (х, у), тогава , а  радиана – ъгълът φ∈[0, 2π].

πrad = 180º 🡪 1 rad = (180/π)º, а 1º = (π/180) rad.

В STL има специална функция за пресмятане на => hypot (x,y), а за функция аркус-тангенс се задава така: atan2(y,x). Изискват включване на файл cmath.

**3. Пресмятане на координати на точка, деляща отсечка в дадено отношение**

Нека М(x,y) е вътрешна точка за отсечката АВ и АМ:МВ = λ, то М ще има координати:

Ако М(x,y) е среда на отсечката АВ, то М ще има координати:.

**Пример 1:** Преминаване от декартови в полярни координати:

#include<iostream>

#include<cmath>

using namespace std;

int main()

{ double x,y, ro,fi;

cin>>x>>y;

ro=hypot(x,y);

fi=atan2(y,x);

cout << ro <<" "<<fi<<endl;

return 0;

}

**Тест: Вход:** 1 1 **Изход:**1.41421 0.785398

**Пример 2:** Преминаване от полярни в декартови координати:

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

double x,y, ro,fi;

cin>>ro>>fi;

x=ro\*cos(fi);

y=ro\*sin(fi);

cout << x <<" "<<y<<endl;

return 0;

}

**Тест: Вход:** 1.41421 0.785398 **Изход:**0.999998 0.999997

**Зад. 1.** Да се пресметне на отсечка по зададени координати на 2-те крайни точки.

sqrt((x1-x2)\*(x1-x2)+(y1-y2)\*(y1-y2))

**Зад. 2.** Дадени са координати на 2 точки. Да се пресметне ъгъла, който сключва правата определена от тези две точки с абсцисната ос.

atan2(y1-y0, x1-x0)

**Зад. 3.** Дадени са координатите на 2 точки А и В. Да се пресметнат координатите на средата на отсечката АВ.

**Зад. 4.** Дадена е отсечка с координати на краищата A (x1,y1) и B(x2,y2) иточка М, която дели отсечката АВ в отношение AM:BM=λ. Да се намерят координатите на точка М.

**Зад. 5.**Даден е триъгълникът ABC с координати на върховете A(x1 , y1), B(x2 , y2) и C(x3 , y3). Намерете координатите на медицентъра на триъгълника ABC.

Решение:

Нека M е среда на страната AB.Тогава координатите на M са:.

Медианата CM се дели от медицентъра G(x, y) на триъгълника в отношение 2:1 считано от върха. Тогава координатите на медицентъра са: , .

**ВЕКТОРИ**

**1. Определение за вектор**

Отсечка, на която единият край е избран за първи (начало), а другият за втори, наричаме насочена отсечка или свободен вектор.

Векторът има начало /т.А/, край /т.В/, посока – посоката на лъча , и дължина – дължината на отсечката АВ – означаваме с |AB|. Множеството от всички насочени отсечки, равни на наричаме свободен вектор, а се нарича негов представител.

Ако относно декартова координатна система Oxy в равнината е дадена точка А (x,y),то означаваме вектора с

Ox

Oy

O

А(x, y)

-

x

y

Неговият **противоположен** вектор е:   
-

**Перпендикулярният** на вектор означаваме с и той ще има координати .

Два вектора и са **колинеарни** /лежат една права или на успоредни прави/ тогава и само тогава, когато съответните им координати са пропорционални, т.е. || ⬄, където λ е коефициент на пропорционалност. Ако някоя от координатите на единия вектор е 0, то съответната координата на другия вектор също е 0. За да избегнем деление на 0, можем да запишем израза така: , т.е. || ⬄.

А(x1, y1)

B(x2, y2)

Ако относно декартова координатна система Oxy в равнината са дадени точките A (*x*1, *y*1) и B (*x*2, *y*2) , то векторът ще има координати:(х2 – х1, y2 – y1).

**2. Ориентирани тройки вектори**

Нека са дадени три вектора , и , които не лежат в една равнина /некомпланарни/ и имат общо начало точка О. Представяме си, че сме в края на вектор и гледаме към равнината, определена от векторите и . Ако по-малкият от ъглите между векторите и е положителен /завъртането към е срещу часовниковата стрелка/, тройката , , се нарича дясна, а по часовниковата стрелка – лява.

Лява тройка

Дясна тройка

a

b

c

a

b

c

**3. Операции с вектори**

**а/ сума** – Ако и , то сумата на двата вектора е вектор, чийто координати са сума от координатите на дадените вектори:

Графично представяне:

I начин: правило на триъгълника – в края на първия вектор нанасяме втория и свързваме началото на Iя векторс края на IIя.

II начин: правило на успоредника – пренасяме втория вектор в началото на първия, допълваме успоредника и свързваме общото им начало със срещуположния връх на успореденика.

Полученият вектор е сума на дадените вектори.

a

b

a

b

a+b

a

b

a+b

Правило на триъгълника

Правило на успоредника

**б/ разлика** - Ако и , то разликата на двата вектора е вектор, чийто координати са разлика от координатите на дадените вектори:

Графично представяне – пренасяме втория вектор в началото на първия и свързваме края на IIя с края на Iя вектор.

a

b

a

b

a-b

**в/ произведение на вектор с число:**Ако и λе реално число, то произведението λ е вектор, с дължина λ\*| | т.е. λ

**г/ скаларно произведение** – Скаларното произведение на два вектора и е число, равно на произведението от дължините на векторите и косинуса на ъгъла заключен между тях:   
.

Когато векторите са зададени в координатна форма, т.е. и , то скаларното произведение е сума от произведенията от съответните им координати: .

Следствие 1:

⬄⬄ остър

⬄⬄ тъп

⬄⬄ 90°⬄

Следствие 2:

**д/ векторно произведение –** Векторно произведение на два вектора и е вектор, който е равен на произведението от големините им и синуса на ъгъла между тях:  
х=. Ако = х, то е ⊥ на равнината, определена от векторите и .

Ъгълът между двата вектора приема стойности от 0° до 180°, следователно векторното им произведение може да приема само неотрицателни стойности.

Свойства:

a

b

c

х= 0 ⬄a || b

х= - х

|х|= .b

/От.b = |.|b|cos (b, ) =

= |.|b|cos (90°-<(a, )) =

|.|b|. sin < (a, ) = = х //тъй като | = |a|/

|х| =

/Тъй като и , то следва, че: = .

И тъй като |х| = .b, следва че и |х| = /

Геометричен смисъл:

Дължината на вектор **c**е равна на лицето на успоредника, определен от векторите и :

Sусп = || =х.

От тук следва, че лицето на триъгълника, определен от векторите и ,е = х.

Векторите a, b и c образуват дясна тройка. Когато векторите са зададени в координатна форма, т.е. и , то векторното произведение може да се запише така:

х

*Обяснение*: Записваме координатите на векторите **a** и **b** една под друга. Първата координата на векторното произведение се получава като “зачеркнем” стълба, образуван от първите координати на векторите a и b и от останалите координати образуваме детерминанта от втори ред. Втората координата се получава чрез “зачеркнем” вторите координати, но получената детерминанта отчетем със знак минус. Третата координата се получава аналогично на първата.

**За всеки два вектораи е в сила: (2 + (х)2 = а2.b2, където a2 = |a|2.**

Доказателство: (2 + (х)2 = **(**)2 + ()2 = ) =

**е/ смесено произведение** – Смесеното произведение на три вектора **a**, **b** и **с** е число, равно на векторното произведение на a и b, умножен скаларно по c: (**a**,**b**,**c**) = (**a** x **b**).**c**.

a

b

c

Смесеното произведение (**abc**) е равно на обема на паралелепипеда, образуван от трите вектора: (**abc**) = V(**abc**).Смесеното произведение е равно на 0 ⬄векторите a, b и c са в една равнина /компланарни/.

Когато векторите са зададени в координатна форма, т.е. , и , то смесеното произведение може да се запише така:

(a,b,c) =

**4. Ориентирана тройка точки**

Ориентацията на три точки е реда, по който ги обхождаме, като значение има посоката на обхождане, а не от коя точка сме тръгнали.

Например 3 точки А, В и С могат да бъдат обходени по 6 начина: АВС, ВСА, САВ, АСВ, СВА и ВАС. От тях първите три са в една и съща посока, разликата е само началната точка. Вторите три начина обхождат точките в противоположната посока.

Така съществено различни се оказват два. За конкретност избираме начална точка А. Тогава различните обходи са АВС и АСВ. Както и да са разположени точките А, В и С, в първия случай ги обхождаме в едната посока, а във II случай – в противоположната посока.

Когато посоката на обхождане на точките АВС е положителна (обратна на часовниковата стрелка), казваме че имаме положително ориентирана тройка точки.

По зададените координати на точките трябва да разберем в каква посока е обхождането.

За да установим това, определяме координатите на векторите и . Означаваме с x1=, y1=, x2=, у2=. Проверяваме стойността на израза x1\*y2 - y1\*x2. Ако x1\*y2 - y1\*x2>=0 точките са положително ориентирани.

Програмна реализация:

#include <iostream>

#include <algorithm>

using namespace std;

struct point { double x,y;};

int ccw(point p0, point p1, point p2)

{ int dx1=p1.x-p0.x;

int dy1=p1.y-p0.y;

int dx2=p2.x-p0.x;

int dy2=p2.y-p0.y;

return dx1\*dy2-dy1\*dx2;

}

int main()

{

point p0, p1, p2;

cin>>p0.x>>p0.y>>p1.x>>p1.y>>p2.x>>p2.y;

if (ccw(p0,p1,p2)>=0) cout << "+" << endl;

else cout << "-" << endl;

return 0;

}

**5. Трансформации**

**I начин – чрез точка**

* Мащабиране (kx, ky)

P

P(x,y)

* Преместване в координатното начало (P-C), мащабиране k(P-C), връщаме на старото място +c. Получаваме k(P-C)+C=kP+(1-k)C

C

* Ротация /въртене на ъгъл φ около т.О/ - за Р използваме полярни координати P(ρ, α) => P(, α) => x = ρ\*cos α, y= ρ\*sin α.

P(x,y)

P’(x’,y’)

За точка Р’ получаваме:

x' = ρ\*cos (α + φ) = ρ\*cos α\*cos φ - ρ\*sin α\*sin φ = x\*cos φ – y\*sin φ

y' = ρ\*sin (α + φ) = ρ\*sin α\*cos φ + ρ\*cos α\*sin φ = y\*cos φ + x\*sin φ

За завъртане спрямо произволна точка: транслация –С, ротация, връщане +С.

**II начин – чрез вектор**

* Ротация – нека (x, y) е дадения вектор и (x’, y’) – новия вектор, получен след ротация на ъгъл φ. Тогава = Rφ., където Rφ = /числата в матрицата се вземат от горния запис на х’ и у’.

= Rφ.

* Мащабиране = s., s=. Позволява различен мащаб по Ох и Оу, ако мащабът е един и същ, k1=k2
* Транлация – не може да се изрази чрез вектори

**Зад. 1.**Дадени са 3 точки A, B и C. Определете, лежат ли те на една права.

Решение:

Трябва да изясним, успоредни ли са векторите AB и AC. Ние можем да изчислим координатите на тези вектори – това са просто разликите от координатите на точките. За да бъдат векторите с координати (x1, y1) и (x2, y2) успоредни, трябва да съществува такова число a, че x2 = a \* x1 и y2 = a \* y1. Затова първата идея се състои в това, да проверим равенството x2 / x1 = y2 / y1. Все пак това не е желателно, тъй като x1 или y1 могат да се  окажат равни на 0. Вместо това ще проверяваме равенството  x1 \* y2 = x2 \* y1.

struct point{int x; int y;};

bool isStraightLine(point A, point B, point C)

{ int x1=B.x-A.x;

int y1=B.y-A.y;

int x2=C.x-A.x;

int y2=C.y-A.y;

return (x1\*y2==y1\*x2);

}

**Зад. 2.** Дадени са 3 точки A, B и C, не лежащи на една права. Определете, явява ли се обходът A→B→C  обход по часовниковата стрелка или обратен на часовниковата стрелка.

Решение:

#include <iostream>

#include <algorithm>

using namespace std;

struct point { double x,y;};

bool isClockWise(point A, point B, point C)

{ int x1=B.x-A.x;

int y1=B.y-A.y;

int x2=C.x-A.x;

int y2=C.y-A.y;

return (x1\*y2 < y1\*x2);

}

int main()

{

point p0, p1, p2;

cin>>p0.x>>p0.y>>p1.x>>p1.y>>p2.x>>p2.y;

if (isClockWise(p0,p1,p2)) cout << "CW" << endl;

else cout << "CCW" << endl;

return 0;

}

**Зад. 3.** Даден е тетраедър с върхове: A(2,1,3) B(3,2,5) , C(3,3,6) , D(4,4,2).

а/ Намерете обема на тетраедъра

б/ Изчислете дължината на височината му през върха C - hC

Решение:

а) Определяме координатите на векторите **AB**, **AC**, **AD**

Определяме обема на тетраедъра чрез смесеното произведение на векторите **AB**, **AC**, **AD**.

б) Определяме векторите **AB** и **AD**.

Пресмятаме лицето на триъгълника ABD.

От формулата за обема на тетраедър определяме височината през върха C.

**Зад. 4.** Дадени са координатите на 4 точки A, B, C и D. Да се намери лицето на четириъгълника ABCD.

A

B

C

D

P

SABCD = SABP + SBCP + SCDP + SDAP =

=

/групираме I и II събираемо, разменяме местата на векторите в произведението/

=

=

=

=

**Зад. 5.** Дадени са координати на 5 точки, които лежат на страните на правоъгълник ABCD съответно: P1и P2∈AB, R∈BC, U∈CD и L∈AD. Да се определи лицето на правоъгълника.

Решение:

U

L

R

P1

P2

V

Проекцията на LR върху P1P2 e AB.

=>

От друга страна намираме и чрез векторното произведение:

От (1) и (2) =>

S = AB\*AD = =

**Зад. 6.** Дадени са векторите , и . Вектор . Да се намерят стойностите на числата αи β.

Решение:

Ще умножим двете страни на даденото равенство векторно с , като по този начин ще неутрализираме, тъй като

a

b

p

P

b'

a'

=>

Аналогично, но този път умножаваме по отляво /за да се получи същия знаменател/, и получаваме

=>

За да избегнем делението на , за което се изисква да е ≠0, можем да запишем израза и по следния начин: .

Ако съвпада с , тогава:

*=>*

**Зад. 7.** Дадени са три точки Q, P1и P2.Да се намери ориентирано разстояние от Q до правата l, определена от точките P1и P2.

P1

P2

Q

Q1

Ориентираното разстояние е положително, когато Q е отляво на вектора.

Решение:

Търсеното разстояние е равно на височината на QQ1 към странaта P1P2 на ΔP1P2Q. Ще намерим лицето на триъгълника чрез векторно произведение и ще го разделим на дължината на странaта P1P2.

, но от друга страна =>

Ако правата е определена от точка Р и вектор u, тогава .

Ако u се нормализира /превърне се в единичен/, то .

P1

P2

Q

Q1

**Зад. 8.** Дадена е права g: (P1, P2) и точка Q. Търсим т. Q1.

Решение:

Q1∈g =>∃t: Q1=P1+t.

Проектираме P1Q:

=>

=>

**Зад. 9.** Дадена е права g: (P1, P2) и точка Q. Да се намери правата g', перпендикулярна на g.

Упътване: g': P = Q + t

**Зад. 10.** Дадени са две прави определени от точка и вектор. Да се намери пресечната им точка.

Решение:

P1

P2

Q1

P1

P2

u

v

g1: P = P1 + s., g2: P = P2 + t.

Пресечната точка удовлетворява и двете условия:

P1 + s. = P2 + t.

s. = P2 - P1 + t.

s. = /

s. = = =

=>s = //Аналогично с може да се пресметне t, ако е необходимо.

=> Q = P1 +

**Зад. 11.** Дадени са две отсечки определени от по две точки /I права – P1 и P2, II права - Q1 и Q2/. Да се намери пресечната им точка.

M

Q2

Решение:

I начин:

* Намираме пресечната точка на правите
* Проверяваме дали лежи върху отсечките /ако параметъра λ ∈ [0, 1]

II начин – спестява пресмятания:

* Проверяваме дали имат пресечна точка

За да се пресичат правите, трябва Q1 и Q2 да са от различни страни на правата, определенеа от P1 и P2. Можем да установим това чрез ориентирани лица на Δ P1P2Q1 и Δ P1P2Q2.

* Ако имат – намираме я

Координатите на точка М намираме от отношението P1M/MP2, или P1M/P1P2, или чрез лицата SΔ P1Q1Q2 / (SΔ P1Q1Q2 + SΔ P2Q1Q2).

**МНОГОЪГЪЛНИК**

**1. Основни понятия**

**а/ многоъгълник** - Всяка затворена начупена линия се нарича многоъгълник. Многоъгълникът се задава с множество от точки в равнината и последователност на обхождане.

**б/ прост многоъгълник** – един многоъгълник се нарича прост, ако никои две несъседни страни нямат обща точка.

А1

А2

А3

А4

В1

В2

В3

В4

Прост многоъгълник

Непрост многоъгълник

**в/ изпъкнал многоъгълник** – един многоъгълник се нарича изпъкнал, ако за всеки негов връх съществува съседен, така че, останалите върхове на многоъгълника са от една и съща страна на правата, която свързва тези два върха.

А1

А2

А3

А4

В1

В2

В3

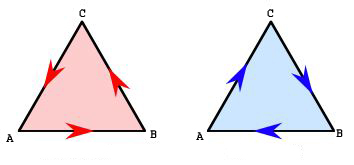
В4

Изпъкнал многоъгълник

Не изпъкнал многоъгълник

Ориентацията на А1, А2 А3 съвпада с ориентацията на контура, ако А2 е изпъкнал.

**2. Ориентирано лице на триъгълник по зададени координати на върховете му**

Ориентираното лице на триъгълника ABC e положително число ако точките A, B и C са наредени в положителна (обратно на часовниковата стрелка) посока и отрицателно число, ако са по часовниковата стрелка. Следователно има значение в какъв ред взимаме точките при ориентираните лица.

Дадени са три точки А(х1, у1), В(х2, у2) и С(х3, у3). Ориентираното лице на триъгълника, определен от тези три точки се намира по формулата:



Ако S < 0, то трите точки А, В и С са наредени по посока на часовниковата стрелка.

Ако S > 0, то точките А, В и С са наредени по посока обрата на часовниковата стрелка.

Ако S = 0, то трите точки лежат на една права.

За да намерим лицето на триъгълника вземаме абсолютната стойност на насоченото лице.

II начин: Ориентираното лице може да се пресметне и по формулата:

А

В

C

А’

В’

C’

X1

X3

X2

У3

У2

У1

Извеждането на формулата става като проектираме върховете на триъгълника върху оста Ох и представим лицето на триъгълника чрез лицата на получените трапци: S = SA’C’CA+SC’B’BC-SA’B’BA = , като за височини вземаме ориентираните разстояния .

III начин: Независимо какви координати за въведени за първия връх условно ще преместим триъгълника, така че връх А да съвпада с началото на координатната система. Новите координати на върховете се получават като се изваждат стойностите на координатите на първия връх.

Същото се получава, ако в първоначалната матрица от I ред извадим останалите и развием по III стълб:



IV начин: Да разгледаме триъгълник, за който т.А е началото на координатната система, а т.В и т.С лежат съответно върху осите Ox и Oy. В израза x1\*y2 - y1\*x2, произведението y1\*x2 е 0. Останалото x1\*y2 представлява удвоеното лице. Може да се докаже, че за всеки триъгълник можем да пресметнем лицето по тази формула. Само трябва да вземем абсолютната стойност и да разделим на две.

   int Surface(point A, point B, point C)

{ int x1=B.x-A.x;

int y1=B.y-A.y;

int x2=C.x-A.x;

int y2=C.y-A.y;

return abs(x1\*y2-y1\*x2)/2;

}

Програмна реализация – по първата формула:

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x1, y1, x2, y2, x3, y3;

cin>>x1>>y1>>x2>>y2>>x3>>y3;

double S=abs((x2\*y3-y2\*x3)-(x1\*y3-y1\*x3)+(x1\*y2-y1\*x2))/2;

cout<< S << endl;

return 0;

}

Тест: Вход: 0 0 0 3 4 0 Изход: 6

Вход: 1 0 5 0 4 7 Изход : 14

**3. Лице на изпъкнал многоъгълник**

Даден е изпъкнал многоъгълник с n върха – точки М1(x1, y1), М2(x2, y2), ..., Мn(xn, yn). Лицето му се пресмята по формулата:



Представяме лицето на многоъгълника като сума от лицата на триъгълниците с върхове т.О и 2 съседни точки от многоъгълника. Тогава можем да запишем формулата



Тъй като лицата са ориентирани, частите които са вътре в многоъгълника се добавят, а тези вън от многоъгълника се вадят.

Програмна реализация:

I начин – масив от point, където point е структура с координатите на точка

#include <iostream>

#include <cmath>

using namespace std;

struct point

{ int x;

int y;

};

int main()

{ int n, i;

cin>>n;

point M[100];

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

cin>>M[i].x>>M[i].y;

double S=0;

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

S = S + M[i].x\*M[i+1].y - M[i+1].x\*M[i].y;

S = S + M[n].x\*M[1].y - M[n].y\*M[1].x;

cout << "S = " << abs(S)/2 << endl;

return 0;

}

Тест: n = 3

1 0 5 0 4 7 Изход : S = 14

n = 4

0 0 5 0 5 10 0 5 Изход : S = 37.5

*Проверете какво ще се случи, ако въвеждаме точките разбъркано, а не се спазва посоката на обхождане.*

II начин – координатите се пазят в двумерен масив

double Area()

{ double S = 0, x1, y1, x2, y2;

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

{ x1 = A[i][0] - A[0][0];

y1 = A[i][1] - A[0][1];

x2 = A[i+1][0] - A[0][0];

y2 = A[i+1][1] - A[0][1];

S += x1\*y2 - x2\*y1;

}

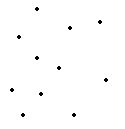
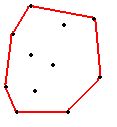
return S/2;

}

**4. Изпъкнала обвивка. Сканиране по Graham.**

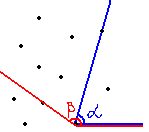
Да се намери обвивката от точки (техните координати), която обхваща всички крайни точки

(т.е. единственият многоъгълник, който съдържа дадените точки).

[](http://judge.openfmi.net/wiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Pict001.jpeg) [](http://judge.openfmi.net/wiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Pict002.jpeg)

Aлгоритъм Graham Scan /сложност O(N\*log(*N*))/

1.Намираме точката, която е с най-малки х и у координати.Нека тя да е т.D.(в програмата p0)

2.Точката се премахва от множеството и след това сортираме всички останали точки във възходящ ред по ъгъла, който се получава между правата, свързваща т.D и текущата точка и правата през т.D, успоредна на абсцисата.

3.Взимаме т.D и първата от сортираните – т.В (защото те винаги са част от обвивката).

4.Взимаме следващата точка А от сортираните и проверяме дали заедно с В е в положителната ориентация. Ако е, от обвивката премахваме В, иначе добавяме А.

5.Повтаряме стъпка 4, докато не обходим всички точки.

Следва пълната програмна реализация:

#include <iostream>

#include <cmath>

#include <algorithm>

using namespace std;

struct point { double x,y;};

point p[100];

bool cmp(point p1, point p2)

{return atan2((double)p1.y-p[0].y, (double)p1.x-p[0].x)< atan2((double)p2.y-p[0].y, (double)p2.x-p[0].x);

}

void swap(int i, int j)

{point w; w=p[i]; p[i]=p[j]; p[j]=w;}

int ccw(point p0, point p1, point p2)

{ int dx1=p1.x-p0.x;

int dy1=p1.y-p0.y;

int dx2=p2.x-p0.x;

int dy2=p2.y-p0.y;

return dx1\*dy2-dy1\*dx2;

}

int main()

{int i,n;

cin >> n;

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

cin >> p[i].x >> p[i].y ;

int min=0;

for(i=1; i<n; i++) if(p[i].y<p[min].y) min=i;

for(i=0; i<n; i++) if(p[i].y==p[min].y) if(p[i].x > p[min].x) min=i;

swap(0,min);

sort(p+1,p+n,cmp);

p[n]=p[0];

int m=2;

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

{

while(ccw(p[m],p[m-1],p[i])>=0) m--;

m++; swap(i,m);

}

n=m;

cout << n << endl;

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

cout << p[i].x << " " << p[i].y << " " << endl;

}

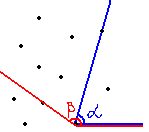
Тест: Вход: 5 Изход: 3

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

II начин – чрез vector /стека също се представя чрез vector/

1.Намираме точката, която ни е с най-малки х и у координати.Нека тя да е т.D.

2.Точката се премахва от множеството и след това сортираме в низходящ ред всички останали точки по косинус от ъгъла м/у правата, която се получава м/у т.D и точката която разглеждаме и правата успоредна на абцисата минаваща през т.D.

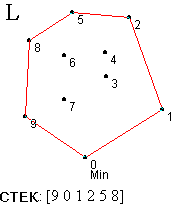
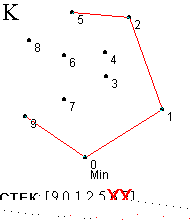
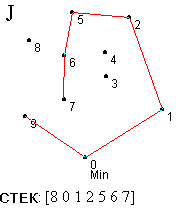
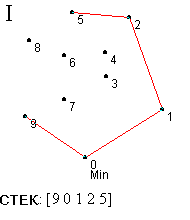
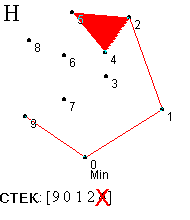
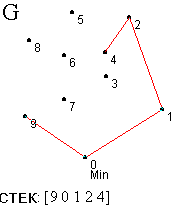
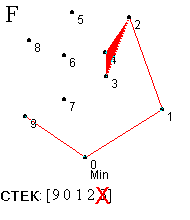
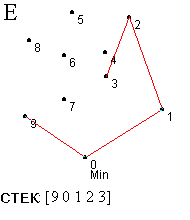
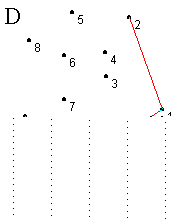
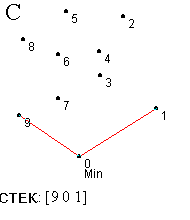
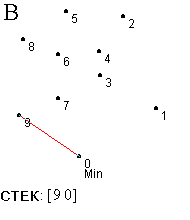
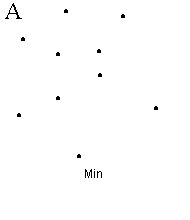
[](http://judge.openfmi.net/mediawiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Pict004.jpeg)

3.Поставяме в стека с точките от обвивката т.D и първата от сортираните (защото те винаги са част от обвивката).

4.Нека т.В - последна добавена точка в стека, т.С - предпоследна добавена точка в стека

Взимаме следващата точка А от сортираните и проверяваме дали т.В(последна точка в стека) е в положителна полуравнина спрямо правата, минаваща през т.С(предпоследна в стека) и т.А: ако е в положителна полуравнина добавяме т.А към стека, иначе премахваме т.В от обвивката и продължаваме със стъпка 4, докато не обходим всички точки.

Бележка: Не трябва да имаме в списъка две еднакви точки. Ако имаме такъв случай алгоритъма не работи коректно и не получаваме правилната обвивка.



Описание на алоргитъма на псевдо език:

(1) Сортираме точките в А спрямо техните X координати по увеличаващ

ред.Ако имаме две точки с равни X координати, гледаме техните Y координати.

(2) Push(A[1]).

(3) Push(A[2]).

(4) Увеличавайки I от 3 до N прави {

Take(P1,P2). // връща последните две точки от стекa без да ги изважда от него, като P2 е последната, а P1 предпоследната точка.

Докато ((InStek >= 2) и ( S(P1,A[i],P2) >0 )) - Del.

Push(A[i]).

}

(5) К := InStek. //броя на точките в стека

(6) Увеличавайки I от 1 до K прави Pop(B[I]).

(7) Push(A[n]).

(8) Push(A[n-1]).

(9) Намалявайки I от N-2 до 1 прави {

Take(P1,P2).

Докато ((InStek >= 2) и ( S(P1,A[i],P2) >0 )) - Del.

Push(A[i]).

}

(10) Del.

(11) L := InStek.

(12) Увеличавайки I от K+1 до К+L-1 прави Pop(B[i]).

Минималната обвивка е записана в масива B, който се състой от K+L-1 точки.

#include <iostream>

#include <cmath>

#include <algorithm>

#include <vector>

using namespace std;

struct Point{int x;int y;};

int N;//брой точки

int sx=10000,sy=10000;//координатите на най-малката точка

vector <Point> masiv;

int used[100];

bool CMP(Point i,Point j)

{ //проверка коя от двете точки е по-голяма по синус...

if((i.x-sx)/(sqrt(pow(i.x-sx,2)+pow(i.y-sy,2)))>(j.x-sx)/(sqrt(pow(j.x-sx,2)+pow(j.y-sy,2))))

return 1;

else return 0;

}

int main()

{ cin>>N;

int it, i;

Point temp;

for(i=1;i<=N;i++)//въвежда координатите

{ cin>>temp.x>>temp.y;

masiv.push\_back(temp);

if(sy>masiv[masiv.size()-1].y)//проверява дали нямаме точка с по-малка ордината

{

it=masiv.size()-1;

sy=masiv[masiv.size()-1].y;

sx=masiv[masiv.size()-1].x;

}

if(masiv[masiv.size()-1].y==sy&&masiv[masiv.size()-1].x<sx)//проверяра дали нямаме точка с по-малка абциса

{

it=masiv.size()-1;

sx=masiv[masiv.size()-1].x;

}

}

masiv.erase(masiv.begin()+it);//изтрива намерената точка

sort(masiv.begin(),masiv.end(),CMP);//сортира по синус-а

vector<Point> stack;

temp.x=sx;

temp.y=sy;

stack.push\_back(temp);//вкарва в обвивката най-малката точка

stack.push\_back(masiv[0]);//вкарва и първата точка от сортираното множество

for(i=1;i<masiv.size();i++)//обхожда всички останали точки

{ int x1,x2,x3,y1,y2,y3;

int status=0;

while(status==0)//докато не вкараме последната точка

{

x3=masiv[i].x;

y3=masiv[i].y;

x1=stack[stack.size()-2].x;

y1=stack[stack.size()-2].y;

x2=stack[stack.size()-1].x;

y2=stack[stack.size()-1].y;

if((x1\*y3+y1\*x2+x3\*y2)-(x2\*y3+y2\*x1+x3\*y1)>0)//ако е в положителната полуравнина

stack.pop\_back();//маха B от обвивката

else {

stack.push\_back(masiv[i]);//вкарва А в обвивката

status=1;}

}

}

for(i=0;i<stack.size();i++)

cout<<stack[i].x<<" "<<stack[i].y<<endl;

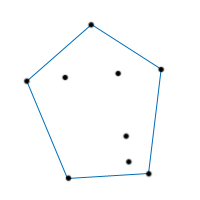
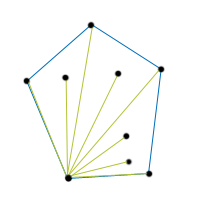
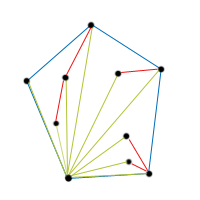
return 0;

}

Тест: Вход: 5 Изход: 4 0 3 1 2 2 1 1 /??? 3 1 е на правата през 4 0 и 2 2/

1 1 2 1 3 1 2 2 4 0

III описание judge.openfmi.net/mediawiki:

[](http://judge.openfmi.net/mediawiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Hull1.jpg)[](http://judge.openfmi.net/mediawiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Hull2.jpg)[](http://judge.openfmi.net/mediawiki/index.php/%D0%9A%D0%B0%D1%80%D1%82%D0%B8%D0%BD%D0%BA%D0%B0:Hull3_copy.jpg)

В началото на алгоритъма Т е празен стек. Червените линии показват коя точка вадим от стека.

1. Намираме най-долната и най-дясно разположена точка от множеството. Нека я означим с Р. Тя със сигурност принадлежи на изпъкналата обвивка
2. Сортираме останалите точки по големината на ъгъла ,който сключва правата минаваща през съответната точка и Р, с абцисата. По този нчин наредената последователност от соритраните точки и Р образуват многоъгълник, в който всички точки са видими от Р Не е нужно да намираме стойноста на ъгъла. Можем просто да сортираме точките по котангенса от ъгъла(който е намаляваща функция в интевала (0, 180) ). Нпраимер ако имаме точката C с координати (x2, y2), a P има координати (x1, y1), то котангенса на ъгъла който вектора PC сключва с абцисата е (x2 - x1)/(y2 - y1). Е трябва да внимаваме ако y1=y2.
3. Пъхаме в Т точката Р и първата точка от сортираната поседователност. Те със сигурност влизат в изпъкналата обвивка.
4. Обхождаме последователно всики точки от соритрантата последователност (без първата защото вече е в стека) и за всчка pовтаряме следното:
   * 1. Нека С е точката кочто обхождаме, В е точката на върха на стека ,а А е точката "под" върха на стека.
     2. Ако ъгълът АВС е по-малък от 180 (в положителна посока) вадим В от стека и отиваме към 4.1 в противен случай добавяме С в стека и преминаваме към следващата точка. Проверката става лесно само като проверим знака на векторното призведение AB x BC. Ако получим отрицателен резулата значи махаме B от стека ;)
5. В стека Т вече имаме изпъкнала обвивка на множеството.

**5. Взаимно положение на точка и многоъгълник**

Даден е многоъгълник и точка А, за която трябва да проверим лежи ли вътре в многоъгълника.

**а/ Принадележност на точка в изпъкнал многоъгълник**

**I начин:** аналогичен на проверката за принадлежност на точка в триъгълник – ако Sabc..xy < Soab + Sobc + .. + Soxy + Soya, тогава точката O лежи извън триъгълника. В противен случай е вътре или на някоя страна. Този подход има линейна по броя на върховете реализация.

**II начин:** чрез двоично търсене: Избираме един връх от многоъгълника за начален (например върха Ax). Разглеждаме ъгъла с връх Ax и рамена минаващи през Ax-1 и Ax+1.

O

Ax-1

Ax

Ax+1

O

Ако точката не принадлежи на ъгъла, значи е извън многоъгълника. /Ако OAxAx+1 са – ориентирани или OAxAx-1 са + ориентирани -> О външна./

Ако точката е вътре в ъгъла избираме върха A[n/2-x] (т.е. този, който разделя другите върхове на две равни части) и построяваме лъч с начало Ax през него. Този лъч разделя първоначалния ъгъл на две части и точката със сигурност е в някоя от тях. Проверяваме в коя – ако ОАхА[n/2-x] е (+) ориентирана, значи О е вляво, иначе е вдясно от лъча ОАхА[n/2-x]. Зоната в която О не е се изключва. Разглеждаме другата зона и повтаряме действието с разделянето на ъгъла, в който се намира. Това продължава докато не ограничим точката между лъчи през два съседни върха Ay и Ay+1. Тогава ако точката принадлежи на триъгълника АxAyAy+1, то тя принадлежи и на многоъгълника, иначе е извън него /разглеждаме ориентацията на трите точки О, Ay и Ay+1 и ако е (+) – точката е вътре.

**б/ Принадележност на точка в произволен прост (несамопресичащ се) многоъгълник**

**I начин:** Да си представим, че многоъгълникът представлява ограда и искаме да разберем дали тя загражда дадена точка или не. Нека наречем едната й страна вътрешна, а другата външна. Ако сме били откъм вътрешната страна и прескочим оградата сме отвън, ако пък сме били отвън и прескочим - озоваваме се вътре. Ако застанем в точката и тръгнем в някоя произволна посока, огато срещнем ограда я прескачаме и ако сме били отвън ставаме вътре и обратно. При второ прескачане пак сме се върнали отвън/отвътре. Така на всеки две пресичания на оградата пак сме в тази област, в която е точката. Нека продължим така докато пред нас вече няма стени. Тогава се намираме със сигурност отвън. Ако сме прескочили четен брой пъти стената, значи и точката е била извън многоъгълника, в противен случай е била вътре.

Реализираме тази идея, като си изберем произволен (удобен за реализация) лъч с начало в разглжданата точка А. За всяка страна на многоъгълника проверяваме дали не се пресича с лъча. Сложността на алгоритъма е O(N).

В качеството на права е удобно да се вземе, например, хоризонтална права (Y = A).   
За нея е по-просто, отколкото за произволна права, да проверяваме наличието на точки на пресичане с отсечките, смятайки нейната координата (една!) и проверявайки, от коя страна на A тази точка лежи.

Трябва да обърнем внимание на частните случаи:

* когато лъчът минава през връх на многоъгълника – точката ще бъде преброена два пъти /по веднъж за двете съседни страни/, а всъщност пресичането е едно или може изобщо да няма пресичане /правата може да преминава през връх на многоъгълника по 2 различни начина – ***пресичайки*** границата на многоъгълника или ***допирайки*** се до границата на многоъгълника/.

Едно пресичане

Няма пресичане

* когато лъчът минава през страна на многоъгълника /страната на многоъгълника е хоризонтална и лежи на правата/ – трябва да разгледаме и съседните точки, защото в единия случай има пресичане, в другия – не.

Няма пресичане

Има пресичане

Този проблем най-лесно се решава като изместим лъча в някоя друга посока.

Друг начин /Банчев/ - Правила, с които опростяваме разглеждането:

* ако отсечката е хоризонтална, не я разглеждаме
* ако не е хоризонтална – разглеждаме по-ниския й край, ако тази точка е крайна и лежи на лъча скъсяваме координатите й, при което се получава нова точка и страните се деформират.

0 пресичания

1 пресичанe

Една модификация на този алгоритъм може за бъде чрез използване на отсечка, с начало дадената точка А и край друга точка B за която сме сигурни, че не е в многоъгълника (примерно с координати по-големи от ограниченията за точките на многоъгълника). Тогава аналогично броим пресечните точки на отсечката и страните на многоъгълника.

Ако многоъгълникът е изпъкнал, тогава правата не пресича контура или пресича в 1 или 2 точки.

Описание реализацията на алгоритъма

М е многоъгълник с n върха, А е дадената точка, Count – брояч, L – права.

(1) М[N+1] := М[1].

(2) Count := 0.

(3) L.x = А.x , L.y = А.y , L.a = 1 , L.b = 0.

(4) Увеличавайки I от 1 до N прави {

P1 := М[I].

P2 := М[I+1].

Ако P1.y < P2.y, то разменяме P1 и P2.

Ако S(P1,P2,А) < 0 и L пресича М[I],М[I+1] и А.y=P2.y, то Count++.

}

(5) Ако Count е четно, то P не лежи в многоъгълника. В противен случай P е вътрешна за A.

ЗАБЕЛЕЖКА: Предварително е нужно да се провери дали точката не лежи на някоя от страните на многоъгълника!

**II начин:** Заставаме в точката и започваме да се движим с поглед по многоъгълника. Ако сме вътре в него, за до обходим целия, ще трябва да направим едно пълно завъртане около оста си, защото той ни обгражда от всички страни. Ако пък сме отвън, колкото и да се завъртаме настрани в даден момент посоката ще се сменя и погледът ни пак ще се връща напред. Реализираме тази идея като съберем всички насочени ъгли AxOAx+1, които съответстват на всяка страна от многоъгълника. Ако сборът е 0 градуса, значи точката O е извън многоъгълника. Ако сборът е 360 градуса, точката O е вътре.

**6. Сечение, обединение и разлика на два многоъгълника**

Правила за определяне на броя пресечни точки

вън вътре

вън вътре

вън вътре

вън вътре

Добавяме само края. Началото не се добавя, т.к. е край на някоя друга страна и ще бъде добавен при разглеждането й.

Обхождаме всички прави и по горното правило определяме броя на пресечните точки.

Правим списъци с пресечните точки

Обхождаме + точките по контура на единия многоъгълник. Когато стигнем до пресечна точка, има 2 вектора – избираме левия. Когато се движим в положителна посока и винаги следваме лява посока – намираме сечение.

Ако обиколим в (-) и пак спазваме лява посока – намираме обединение.

Ако единия контур се обхожда в + посока, а другия в отрицателна – получаваме разлика на многоъгълниците.

**Зад. 1.** Да се напише програма, която да познае дали две точки са в една и съща полуравнина спрямо дадена права или не.

Решение:

Ако обходим точките така 1 2 3 после 1 2 4. Ако имат различни знаци, значи са в различни полуравнини.

a= ccw(p1,p2,p3);

b= ccw(p1,p2,p4);

if(a\*b<0) са с различни знаци

II начин: Нека L е права и A,B са точки от нея, а P1,P2 e oтсечка. Тогава:

Ако S(A,B,P1)\*S(A,B,P2) > 0,то L не пресича отсечката.

Ако S(A,B,P1)\*S(A,B,P2) <= 0, то L пресича отсечката.

Ако S(A,B,P1)=0 и S(A,B,P2)=0,то отсечката лежи на изцяло на правата.

**Зад. 2.** Да се напише програма, която да познае дали точка е вътрешна за триъгълник.

Решение:

I начин:

a= ccw(p1,p2,p4);

b= ccw(p2,p3,p4);

c = ccw(p3,p1,p4);

II начин: Нека имаме триъгълник ABC и разглеждана точка O. Ако Sabc < Soab + Sobc + Soca, тогава точката О се намира извън триъгълника. В противен случай, ако някое от лицата Soab, Sobc или Soca е равно на нула, точката лежи на съответната страна. Щом никое от предишните не е изпълнено, точката О принадлежи на вътрешността на триъгълника.

**Зад. 3.** Да се напише програма, която да установи дали две отсечки се пресичат.

Решение:

Р3

Р4

Р1

Р2

a= ccw(p1,p2,p3);

b= ccw(p1,p2,p4);

c = ccw(p3,p4,p1);

d = ccw(p3,p4,p2);

if(a\*b<0&& c\*d<0) се пресичат

II начин: Нека P1,P2 e първата отсечка, а P3,P4 е втората. За да се пресичат двете отсечки е нужно: S(P1,P2,P3)\*S(P1,P2,P4) < 0 и S(P3,P4,P1)\*S(P3,P4,P2) < 0

ЗАБЕЛЕЖКА: Има частни случаи когато три точки лежат на една права и те трябва да се проверят отделно!

III начин:

Отсечките AC и BD се пресичат тогава и само тогава, когато четириъгълника ABCD е изпъкнал. За да бъде четириъгълника ABCD изпъкнал, необходимо и достатъчно е, всички 4 обхода A→B→C, B→C→D, C→D→A, D→A→B да бъдат обходи в една страна: или по часовниковата стрелка, или обратно.   
function isConvex4 (A, B, C, D: point): boolean;

begin

  if (isClockWise (A, B, C)) then

    isConvex4 := isClockWise (B, C, D) and //isClockWise описана в зад. 2 Вектори

                 isClockWise (C, D, A) and

                 isClockWise (D, A, B)

  else

    isConvex4 := (not isClockWise (B, C, D)) and

                 (not isClockWise (C, D, A)) and

                 (not isClockWise (D, A, B))

end;

Тази функция работи в ***не особените*** случаи, т. е няма 3 или 4 точки лежащи на една права. Ако 3 или всички 4 върха се окажат на една права, то четириъгълника се счита за изпъкнал, но тези случаи трябва да се разгледат отделно. Програма, която удовлетворява изискванията на заданието може, например, да проверява отначало, не лежат ли някои 3 точки на една права, извиквайки 4 пъти функция isStraightLine.

bool isStraightLine(point A, point B, point C)

{ int x1=B.x-A.x;

int y1=B.y-A.y;

int x2=C.x-A.x;

int y2=C.y-A.y;

return (x1\*y2==y1\*x2);

}

**Зад. 4.** Дадени са 3 точки A, B и C, лежащи на една права. Определете, коя от тях лежи между двете други.

На първият ред се въвежда число K -- броя тестове във файла. На всеки от следващите K реда се въвеждат 3 двойки цели числа: координатите на точките A, B и C /никои две точки не съвпадат/.

Резултатът се извежда на K реда, като i-тият ред извежда буквата, съответстваща на средната точка.

|  |  |
| --- | --- |
| **ВХОД** | **ИЗХОД** |
| 3  0 0 1 1 2 2  0 0 1 1 -1 -1  0 0 -2 -2 -1 -1 | B  A  C |

Решение:

Ще решаваме задачата не за самите точки, а за техните проекции на оста X. За това трябва да изясним, кое от числата A.x, B.x и C.x лежи между двете други. Следва да имаме предвид, че тези числа могат да съвпаднат (ако точките лежат на права, перпендикулярна на оста X). Тогава ще се наложи да разгледаме проекциите на другата ос.

#include <iostream>

#include <cmath>

#include <algorithm>

#include <vector>

using namespace std;

struct Point{int x;int y;};

int middlePoint (Point A, Point B, Point C)

{ if (A.x > B.x)

if (C.x > A.x) return 1; // B A C

else if (C.x > B.x) return 3; // B C A

else return 2; // C B A

else if (A.x < B.x)

if (C.x < A.x) return 1; // C A B

else if (C.x < B.x) return 3; // A C B

else return 2; // A B C

else if (A.y > B.y) //ако проекциите на оста X съвпадат, гледаме Y

if (C.y > A.y) return 1 ; // B A C

else if (C.y > B.y) return 3; // B C A

else return 2; // C B A

else if (C.y < A.y) return 1; // C A B

else if (C.y < B.y) return 3; // A C B }

else return 2; // A B C

}

int main()

{

Point A, B, C;

int n, i, x;

cin>>n;

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

{

cin>>A.x>>A.y>>B.x>>B.y>>C.x>>C.y;

x = middlePoint(A, B, C);

if (x==1) cout <<"A"<<endl;

if (x==2) cout <<"B"<<endl;

if (x==3) cout <<"C"<<endl;

}

return 0;

}

**Зад. 5.** Намерете лицето на даден прост многоъгълник P1, P2, ...,PN. Предполагаме, че съществува вътрешна точка P , която дели областта на непресичащи се триъгълници с върхове (P, Pi, Pi+1), i = 1, 2,..., N – 1 и (P,PN,P1).

**Зад. 6.** Напишете програма, която проверява дали дадена точка е вътре в многоъгълник.

**Зад. 7.** Напишете програма, която проверява изпъкнал ли е даден многоъгълник.

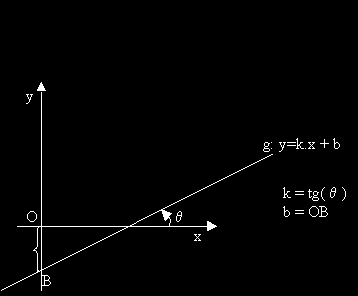
**ПРАВИ В РАВНИНАТА**

**1. Уравнение на права**

Уравнение на правата g се нарича такава връзка между x и y, зададена чрез равенство, която се удовлетворява единствено от координатите на тези точки, които лежат на правата g.

**а/ общо уравнение на права**

Всяка права g в равнината има уравнение от вида Ах + Ву + С = 0, където А, В, С ∈R и поне един от коефициентите А и В е различен от 0, т.е. |A| + |B| ≠ 0.

g: Ax + By + C = 0

Векторът (-В,А) е успореден на правата g, a векторът (А,В) е перпендикулярен на правата.

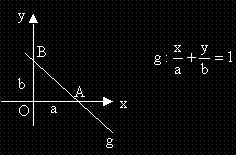
**б/ декартово уравнение на права**

g: y = kx + n , k = tg ϕ,където ϕ е ъгълът межу положителната посока на оста Ox, и правата g. Числото *k* се нарича ъглов коефициент на правата.

**в/ уравнение на права през точка М0 (x0, y0) и ъглов коефициент k**

g: y – y0= k (x – x0)

**г/ уравнение на права през две точки А (х1,y1) и B(x2,y2) /канонично уравнение/**

g: или g:

**д/ отрезово уравнение:**

g: където числата *a* и *b* са алгебричните мерки на отсечките, които правата отсича откоординатните оси.

**е/ уравнение на права определена от точка М0(х0,y0 ) и успореден вектор***(a,b).*

g:

**2. Взаимно положение на точка и права**

Дадена е права g: Ax + By + C = 0 и точка М (х0, у0). За точката М има три възможности – да лежи на правата, да е в горната полуравнина на правата или да е в долната полуравнина.

а/ Точката М (х0, у0) лежи на правата g, т.е. М (х0, у0) ∈g ⇔A\*x0 + B\*y0 + C = 0

б/ Точката М (х0, у0) е над правата ⇔A\*x0 + B\*y0 + C > 0

в/ Точката М (х0, у0) е под правата ⇔A\*x0 + B\*y0 + C < 0

int main()

{

int x1,x2,y1,y2,xd,yd;

cin>>x1>>y1;//Координатите на първата точка от правата

cin>>x2>>y2;//Координатите на втората точка от правата

cin>>xd>>yd;//Координатите на точката, която ще проверяваме

//Основна част...

if((xd\*(y1-y2)+yd\*(x2-x1)+(x1\*y2-x2\*y1))==0) cout<<"Точката е на правата"<<endl;

if((xd\*(y1-y2)+yd\*(x2-x1)+(x1\*y2-x2\*y1))>0) cout<<"Точката е над правата "<<endl;

if((xd\*(y1-y2)+yd\*(x2-x1)+(x1\*y2-x2\*y1))<0) cout<<"Точката е под правата"<<endl;

return 0;

}

**3. Разстояние от точка до права**

Нека имаме правата g: Ax + By + C = 0и точка М(х0, у0). Ориентираното разстояние е D =  и е положително за точките от едната страна на правата и отрицателно – за точките от другата страна. Разстоянието от точката до правата се е равно на|D| = .

**4. Взаимно положение на две прави**

Дадени са прави g: A1x + B1y + C1 = 0 и f: A2x + B2y + C2 = 0. Правите могат да бъдат успоредни, перпендикулярни или да се пресичат. Частен случай е правите да съвпадат.

**а/ успоредност** – Ако е изпълнено A1/A2=B1/B2, или, за да се избегне деление на 0, записваме израза A1\*B2=A2\*B1, тогава двете прави са успоредни.

**б/ съвпадение** – Ако е изпълнено (A1/A2)=(B1/B2)=(C1/C2), или (A1\*B2=A2\*B1) and (A1\*C2=A2\*C1), двете прави съвпадат.

### в/ перпендикулярност– Ако е изпълнено A1\*A2 + B1\*B2 = 0, то двете прави са перпендикулярни

**Забележка**: Ако правите са зададени с техните декартови уравнения, тогава:

* Ако две прави са успоредни, то ъгловите им коефициенти са равни k1=k1.
* Ако две прави са перпендикулярни, то произведението на техните ъглови коефициенти е равно на минус единица k1\*k2 = -1.

**5. Координати на пресечна точка на 2 прави**

Дадени са прави g: A1x + B1y + C1 = 0 и f: A2x + B2y + C2 = 0. Търсим координатите на пресечната им точка, ако има такава.

Ако имат пресечна точка М(х0, у0), то тя ще удовлетворява уравненията и на двете прави, т.е координатите й ще са решение на системата:

A1x0 + B1y0 + C1 = 0

A2x0 + B2y0 + C2 = 0

Преобразуваме я във вида

A1x0 + B1y0 = -C1

A2x0 + B2y0=- C2

Означаваме детерминантите:

∆ = |А1 В1| ∆1= |-С1 В1| ∆2=|А1 -С1|

|А2 В2| |-С2 В2| |А2 -С2|

Ако ∆ = 0, то правите са или успоредни или съвпадат.

Ако ∆ != 0, то координатите на пресечната точка се дават с формулите на Крамер::

x0 = ∆1/∆ и y0 = ∆2/∆

## 6. Ъгъл между две прави

Дадени са правите и g: A1x + B1y + C1 = 0 и f: A2x + B2y + C2 = 0 и t е ъгълът между тях. Косинусът на ъгъла *t* е:

\cos t = \frac{A_1 A_2 + B_1 B_2}{\sqrt{{A_1}^2 + {B_1}^2} \sqrt{{A_2}^2 + {B_2}^2}}

Зад. Да се намери общото уравнение на права по зададени координати на две точки от нея.

Решение:

Общото уравнение на правата има вида: Ax+By+C=0. Трябва да намерим коефициентите A, B и С.Ще запишем уравнение на права през две точки: , след което преобразуваме израза като извадим x и y скоби, а изразите в скобите са стойностите на коефициентите А, В и С.

(x\*y1+y\*x2+x1\*y2)-(x2\*y1+x\*y2+y\*x1)=0 =>x\*(y1-y2)+y\*(x2-x1)+(x1\*y2-x2\*y1)=0=>

A=y1-y2;

B=x2-x1;

C=x1\*y2-x2\*y1

Зад. Дадена е права и 2 точки, нележащи на нея. Да се намери по-голямото от разстоянията от точките до правата.

## 

**ОКРЪЖНОСТ**

## 1. Уравнение на окръжност

Нека центъра О на окръжността е с координати (a, b) и R е радиусът на окръжността, тогава от уравнението на окръжност:

(*x* − *a*)2 + (*y* − *b*)2 = *R*2

**2. Взаимни положения на точка и окръжност**

Дадена е окръжност с център О(a, b) и радиус R и точка А(х0, у0). Получаваме три случая

1. Ако (*x0* − a)2 + (*y0* − b)2 > *R*2 => точката е извън окръжността.

2. Ако (*x0* − a)2 + (*y0* − b)2 = *R*2 => точката принадлежи на окръжността

3. Ако (*x0* − a)2 + (*y0* − b)2 < *R*2 => точката е вътре в окръжността

**3. Взаимни положения на права и окръжност**

Дадена е окръжност с център О(a, b) и радиус R и права g: Ax+By+C=0. Определяме разстоянието от центъра на окръжността О(a, b) до правата g:

* Ако d < R => правата g пресича окръжността /имат две общи точки/
* Ако d = R => правата g допира окръжността /има една обща точки/
* Ако d > R => правата g и окръжността нямат общи точки

**Зад. 1.** Дадена е окръжност с център С и радиус R и точка Р. Да се провери къде лежи точката.

**Зад. 2.** Дадена е окръжност с център С и радиус R и точка Р. Да се намерят допирателните от т.Р към окръжността.

R

C

r

u

P

L

U = CP, P = CR

=> CR = P =

CP.CR = |CP|.|CR> = |CR|2 = r2

= > R =

L =

**Зад. 3.** Дадени са 2 окръжности, да се намерят допирателните към двете оркъжности.

**ОБЩИ ЗАДАЧИ**

**Задача 1. КВАДРАТИ /VII НЛШИ – Габрово, 2008, Зад. за дом. работа/**

Разглеждаме правоъгълна координатна система и редицата **К1, С1, К2, С2**,… от квадрати в нея. Квадратите са дефинирани в съответствие със следните правила:

- Квадратите от тип К имат върхове с координати **(ai,ai), (-ai,ai),   
(-ai,-ai), (ai, -ai),** където **a1** = 1, **ai+1** = 2**ai+1**, **i** = 1, 2, …

- Страните на квадрата Сi, i = 1, 2, … съдържат върховете на квадрата Ki и са успоредни на диагоналите му.

Напишете програма **SQUARE**, която, за зададена точка с координати **(х,у)**, намира квадрата с най-малко лице, съдържащ тази точка.

**Вход:** Входните данни се четат от стандартния вход и включват две цели числа **х** и **у** - координати на точката (**|x|, |y|** ≤106).

**Изход:**Програмата извежда на единствения ред на стандартния изход типа и номера на квадрата, който съдържа точката и има най-малко лице.

**ПРИМЕР: Вход:** 2 4 **Изход:**  K3

**Задача 2. ФИГУРА /VII НЛШИ – Габрово, 2008, Зад. за дом. работа/**

Дадени са N (N>1) правоъгълници, за които се знае, че:

* Всеки от тях е със страни, успоредни на координатните оси и е дефиниран с два от върховете си, които са краища на единия от диагоналите му;
* Всеки правоъгълник има общи вътрешни точки с поне един от останалите и няма общи върхове, страни или части от страни с нито един от останалите правоъгълници.

Да се напише програма **FIGURE**, която чрез последователност от точки определя начупена, затворена фигура F, която е обединение на дадените правоъгълници; определя дали фигурата съдържа „дупки”, т.е. затворени площи във F, които не й принадлежат.

**Задача 1:ТОЧКИ /VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

Дадени са правоъгълник със страни успоредни на координатните оси и *n* точки – *M*1,*M2*, ...,*Mn*. Правоъгълникът е определен чрез координатите на два противоположни върха. Всяка от точките *Mi* също е зададена чрез своите координати. Напишете програма **POINTS**, която намира колко от дадените точки са разположени във вътрешността или по контура на правоъгълника*.*

**Вход:** От първия ред на стандартния вход се въвеждат числата *a*, *c*, *b*, *d*, където (*a,c*) и (*b,d*) са координати на два противоположни върха на правоъгълника. От втория ред се въвежда числото *n* (*n*≤ 100). От всеки от следващите *n* реда се въвеждат координатите *x*i,*y*i  на поредната точка. Всички координати са цели числа от интервала .

**Изход:** На стандартния изход да се изведе търсения брой.

**ПРИМЕР: Вход Изход**

2 -1 -1 2 2

4

-3 –2

1 1

1 2

2 3

**Задача 2: ПРАВОЪГЪЛНИК/VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

Правоъгълникът ABCD е със страни успоредни на координатните оси и е зададен чрез координатите на върховете A и C: A(X1,Y1), C(X2,Y2). Числата X1,Y1, X2,Y2 са цели и 0< X1< X2<10000, 0< Y1< Y2<10000. Дадена е и точка M с целочислени координати: М(X, Y), 0<X<10000, 0<Y,10000. Напишете програма **line**, която намира броя N на точките с целочислени координати, които лежат на правата OM и се намират по страните или във вътрешността на правоъгълника ABCD.

**Вход:** От стандартния вход се въвеждат координатите на А, С и М, разделени с по един интервал.

**Изход:** На единствения ред на стандартния изход се извежда числото N.

**Пример: Вход:** 1 1 3 3 4 4 **Изход:**  3

**Задача 3. ПАРАЛЕЛЕПИПЕДИ/VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

Дадени са два паралелепипеда със страни A1, B1, C1 и A2, B2, C2 – цели положителни числа не по-големи от 1000. Напишете програма **PARA**, която да определя дали някой от двата паралелепипеда може да се помести в другия.

**Вход:** Първият ред от стандартния вход съдържа числата A1, B1, C1, разделени с по един интервал, а вторият - числата A2, B2, C2, разделени с по един интервал.

**Изход**: На единственния ред на стандартния изход се извежда:

0, ако нито първият паралелепипед може да се постави във втория, нито вторият паралелепипед - в първия;

1, ако първият паралелепипед може да се постави във втория;

2, ако вторият паралелепипед може да се постави в първия;

3, ако двата паралелепипеда са еднакви.

**ПРИМЕР 1: Вход Изход**

1 1 1 1

2 2 2

**ПРИМЕР 2:Вход Изход**

10 2 2 2

2 2 3

**ПРИМЕР 3:Вход Изход**

10 2 2 0

3 3 5

**ПРИМЕР 4:Вход Изход**

10 2 2 3

2 10 2

**Задача 4: ПЛОЩАДКИ/VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

Теренът на едно училище е правоъгълник с размери N на M метра. Да приемем, че той е разграфен на квадратчета със страна единица. Координатите на най-долното му ляво квадратче са (1,1), а на най-горното му дясно квадратче (N,M). По различно време върху терена са били асфалтирани К правоъгълни спортни площадки така, че част от терена е асфалтиран, а част – не. На примера тези площадки са защриховани. Разположението на всяка площадка се задава с положението на най-долното си ляво и най-горното си дясно квадратче. Училището иска да асфалтира и останалата част от терена, затова иска да знае колко квадратни метра не са асфалтирани. Напишете програма **SPORT**, която пресмята лицето на неасфалтираната площ.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 1 |  |  |  | 5 |  |  |  |  | 10 |  |  |  |  | 15 |  |  |  |  | 20 |  |  |  | 24 |

**Вход:** От първия ред на стандартния вход се въвеждат целите числа N, M и K (0<N, M≤500, 0≤ K<50). На всеки от следващите K реда са зададени, разделени с по един интервал, целите числа X1, Y1, X2, Y2 – разположението на най-долното ляво и най-горното дясно квадратче на една от площадките.

**Изход:** Резултатът се извежда на единствения ред на стандартния изход.

**Пример**

**Вход Изход**

24 10 4 71

2 2 8 7

20 5 24 9

9 1 20 6

10 5 17 10

**Задача 5: ТРИЪГЪЛНИЦИ /VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

Дадени са N (N≤3000) точки в равнината, никои три от които не лежат на една права. Някои от точките са свързани с отсечки. Напишете програма **TRIAG**, която намира броя и върховете на триъгълниците със страни три от зададените отсечки.

**Вход**

От стандартния вход се въвеждат краен брой тестови случаи. Входните данни от всеки тест са разположени по следния начин:

1. От първия ред се въвежда N;
2. Следват N реда, съдържащи координатите на всяка от точките;
3. Следващите редове задават информация за съществуващите отсечки, като на всеки ред се задава двойка от номера на точки, които са свързани с отсечки.
4. За край на теста се въвежда ред с номера на точки 0 0.

**Изход**

Резултатът се извежда на стандартния изход. На първия ред за всеки тест се извежда броя на триъгълниците. За всеки триъгълник да се изведат на отделен ред координатите на точките, които са негови върхове, подредени по нарастващ ред на номерата им. Числата да са разделени с един интервал да се изведат с точност до четвъртия знак.

**Пример**

**Вход Изход**

6 2

1 1 1.0000 1.0000 3.0000 5.0000 1.0000 6.0000

3 1.5 1.0000 1.0000 3.0000 1.5000 1.0000 6.0000

7 10

3 5

10 7

1 6

3 5

4 6

1 6

1 4

6 4

1 2

2 6

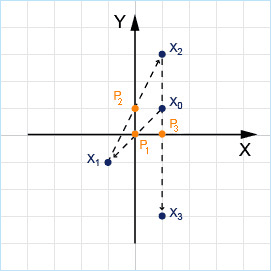
0 0

**Задача 6. СИМЕТРИЯ НА ТОЧКИ /VII НЛШИ – Габрово, 2008, Зад. за сам. работа/**

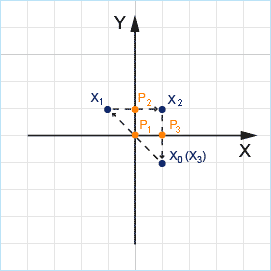
**Ограничениеза време:** 0.1 сек, **Ограничение за памет:** 64 МБ

Нека *A*, *B* и *C* са три точки в равнината (не задължително различни). Ще казваме, че точката *C* е симетрична на точката *A* относно точката *B*, ако са изпълнени следните три условия:

1. Точките *A*, *B* и *C* лежат на една права.
2. Дължините на отсечките *AB* и *BC* съвпадат и са точно два пъти по-малки от дължината на отсечката *AC*.

Преобразуванието *последователно симетрично отражение* на точката *X*0 относно *n* различни точки *P*1, *P*2, ..., *P*n става по следния начин:

* на първоначалната точка *X*0 се строи симетричната ừ точката *X*1 относно P1;
* на точката *X*1 се строи симетричната ừ точка *X*2 относно *P*2;
* ...
* на точкката  се строи симетричната ừ точка *X*n относно *P*n;
* точката Xn е резултатът от преобразуването.

Например, нека *n*=3 и точките *P*1, *P*2 и *P*3 са с координати (0,0), (0,1) и (1,0), съответно. Тогава точката *X*0(1,1), в резултат от последователното симетрично отражение относно *P*1, *P*2, *P*3 преминава в точката *X*3(1, -3) (виж фигурата).

Ще наричаме точката X0 *неподвижна* за описаното по-горе преобразувание, ако координатите ừ са цели числа и в резултат на последователното симетрично отражение относно P1, P2, ..., Pn тя преминава в себе си. За горния пример точката (1, -1) е неподвижна: (виж фигурата).

Напишете програма **SymPoint**, която получава на входа координатите на *n* точки *P*1, *P*2, ..., *P*n и връща координатите на точка, неподвижна за преобразуванието последователно отражение относно точките P1, P2, ..., Pn.

**Вход:**

На първият ред на стандартния вход е зададен броят точките - n . На следващите *n* реда са зададени координатите на точките *P*1, *P*2, ..., *P*n. Всеки такъв ред съдържа координатите *x* и *y* на съответната точка.

**Изход:**

Ако съществува точно една точка (*x*,*y*), неподвижна за преобразуванието последователно отражение относно точките *P*1, *P*2, ..., *P*n, то програмата трябва да изведе нейните координати *x* и *y* на един ред на стандартния изход, разделени с точно един интервал.

Ако съществуват две или повече неподвижни точки, програмата да изведе 0. Ако неподвижни точки не съществуват, програмата да изведе –1.

**Ограничения:**

1. Броят точки *n* е от 1 до 1000 включително.
2. Никои две точки не съвпадат.
3. Координатите на точките са в обхвата на типа int.

**Примери:**

1)

|  |  |  |
| --- | --- | --- |
| Вход | Изход | Примера от условието на задачата. Точката (1, -1) е неподвижна. Други неподвижни точки не съществуват. |
| 3  0 0  0 1  1 0 | 1 -1 |

2)

|  |  |  |
| --- | --- | --- |
| Вход | Изход | Всяка точка от равнината е неподвижна. |
| 4  0 0  0 1  1 1  1 0 | 0 |

3)

|  |  |  |
| --- | --- | --- |
| Вход | Изход | Неподвижни точки не съществуват. |
| 4  0 0  1 0  0 1  100 100 | -1 |

**Зад. 7. ТОЧКИ /НОИ 2012, I кръг, задача A3, автор: Павлин Пеев/**

Откакто написа свой „случаен генератор“ (програма, която извежда цели числа в затворения интервал [-100, 100]), Васко не спира да извършва следния опит:

1. Пуска генератора си 8 пъти и записва получените числа по две на ред;

2. Разглежда двойката числа на всеки ред от ляво надясно като абсциса и ордината на точка (в милиметри) и нанася тези точки на лист хартия;

3. Внимателно свързва всяка отбелязана точка на листа с всяка друга;

4. Прави точни разрези с форматно ножче по начертаните отсечки;

5. Вдига листа от масата. Върху нея най-често остават няколко триъгълни изрезки.

Васко ги преброява и записва получения резултат (естествено, ако на масата не останат триъгълни изрезки, Васко си записва нула). Напишете програма **dots**, която извежда крайния резултат, без да се хаби ценна хартия.

**ВХОД:** От стандартния вход се въвеждат 4 реда, на всеки от които има по две цели числа, разделени с интервал – абсциса и ордината на точка.

**ИЗХОД:** Запишете на стандартния изход един ред с едно цяло число – броя на триъгълните изрезки, които ще останат на масата след реализиране на описания алгоритъм.

**ПРИМЕР 1**

**Вход**

-1 -1

3 2

1 4

-3 3

**Изход**

4

**Обяснение**: На масата ще останат следните триъгълни изрезки: ABE, BCE, CDE и ADE.

**ПРИМЕР 2**

**Вход**

-1 -1

3 2

3 2

-2 2

**Изход**

1

**Забележка**: Тестовите примери са 30 и са групирани по три. Точки се получават при верен отговор и на трите теста в групата

**Зад. 8. НАЧУПЕНА ЛИНИЯ /НОИ 2011 – I кръг, задача В2, автор: Стоян Капралов/**

В равнината са дадени *n* точки. Напишете програма **line**, която намира дължината на най-късата начупена линия, съставена само от хоризонтални и вертикални отсечки, минаваща през всяка от дадените точки. На първия ред на стандартния вход е дадено числото *n* (1< *n* < 10), а на следващите *n* реда са дадени по две цели числа *x* и *y* – координатите на поредната точка (0< *x, y* < 10). Резултатът да се изведе на стандартния изход.

**Пример: Вход:** 3 **Изход:** 9

1 2

2 6

5 2

**Обяснение:** Минималната възможна дължина е 9. Линията (2,6) – (2,2) – (1,2) – (5,2) е едно възможно решение, за което тази дължина се достига.

**Анализ**: Броят на точките *n* е най-много 9, затова е възможно да бъдат разгледани всичките *n*! възможни пермутации на точките. За целта използваме програмата next\_permutation от стандартната библиотека. Вместо да разместваме самите точки, разместваме техните индекси. За всяка подредба пресмятаме дължината на съответната линия. Разстоянието между две последователни точки *A*(*x*1,*y*1) и *B*(*x*2,*y*2) е равно на |*x*2 – *x*1| + |*y*2 – *y*1|, защото се разглеждат само начупени линии, образувани от отсечки, успоредни на координатните оси.

Решение: Радостина Велевска

#include<iostream>

#include <algorithm>

using namespace std;

struct point

{ int x, y;};

int dist(point a, point b)

{

int x=abs(a.x-b.x);

int y=abs(a.y-b.y);

return x+y;

}

point a[10];

int n;

int main()

{ cin>>n;

for(int i=0; i<=n-1; i++)

cin>>a[i].x>>a[i].y;

int p[10];

for(int i=0; i<=n-1; i++)

p[i]=i;

int d=9999;

do

{ int s=0;

for(int i=0; i<=n-2; i++)

s=s+dist(a[p[i]], a[p[i+1]]);

if(d>s) d=s;

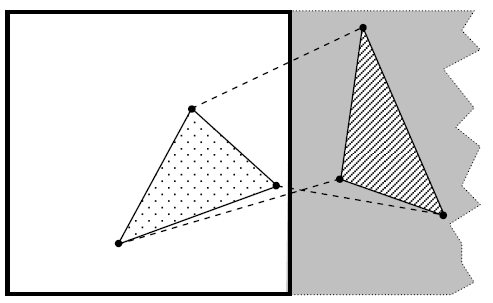
}while(next\_permutation(p, p+n));

cout<<d<<endl;

return 0;

}

**Зад. 9. ПАРЦЕЛ /НОИ 2010, II кръг, задача A2/**

Ама че работа! Точно забихте солидните (и скъпички) колове на наследствената Ви триъгълна част от Широка поляна и – „Натура 2000”! Оказва се, че парцелът попада изцяло в рамките на квадратна защитена територия от десет декара! А сега какво? Как „какво” – заменка, разбира се! Позволяват Ви да си изберете където си искате на изток извън защитеното място друга земя със същата площ.

Вие пък си правите проста сметчица: първо – новата площ пак да е триъгълна (няма да купувате още колове, я!); второ – тези трите да ги местите, естествено, колкото може по-малко. Ако погледнете схемата, става дума за следното: трябва да освободите триъгълната част, която е вътре в квадрата, и да заемете част с равна площ надясно извън него. На Вас пък Ви се иска тя също да е триъгълна и така да пренесете коловете, че сумата от пунктираните линии да е колкото може по-малка. Може да имате и общи точки с границата на защитената територия, но площта да е извън нея. Не е задължително новите места на коловете да са с целочислени координати. Има свободно място само надясно от квадрата (сивата област на схемата, която не е ограничена отдясно).

Ако питате мен, това, което сте замислили на горната схема, е доста далеч от оптималното. По-добре напишете програма **plot**, която да намери решение с колкото може по-малка сума от премествания.

**Вход:** От стандартния вход се въвеждат три реда. Всеки ред съдържа две неотрицателни цели числа, разделени с интервал: координатите на един кол, в цели метри относно югозападния (долния ляв) ъгъл на защитения квадрат. Триъгълникът е със строго положителна площ.

**Изход:** Запишете на стандартния изход три реда с по две реални числа, разделени с интервал и форматирани с точност до третия знак след десетичната точка. Това са новите координати на всяка от съответните точки от входа относно същото координатно начало – югозападния (долния ляв) ъгъл на защитения квадрат.

**Ограничения**

Както е казано по-горе, площта на защитения квадрат е десет декара, а триъгълникът не е изроден и всяка от точките е вътрешна (най-много – контурна) за квадрата.

**Оценка**

Ако новият парцел не е в позволената зона или не е равнолицев на дадения (с точност до третия знак след десетичната точка), тестовият пример получава нула. В противен случай получава от 0 до 10 точки, съобразно близостта на решението до оптималното решение за минимално преместване.

**Пример**

**Вход**

10 20

87 53

33 90

**Изход**

100.000 20.000

166.157 53.000

100.000 90.000

**Зад. 10. Прави.**

Разглеждаме точките с целочислени координати **(*x*,*y*)**, за които **0≤*x*≤*a*** и **0≤*y*≤*b*.** Напишете програма **lines**, която намира колко прави минават през поне две от тези точки.

**Вход:** От един ред на стандартния вход се въвеждат ***a*** и ***b*(0<*a*,*b*<3000).**

**Изход:** На един ред на стандартния изход програмата трябва да изведе броя на търсените прави.

**ПРИМЕР Вход** 2 2 **Изход** 20

**Решение:**

През дадените точки минават ***b*+1** хоризонтални и ***a*+1** вертикални прави. Остава да изброим всички останали прави. В следващите разглеждания ще се интересуваме само от прави, които минават през поне две от дадените точки, но не са хоризонтални или вертикални.

***Решение 1***

Една идея за намиране на този брой е следната: През всеки две точки минава една права. Има опасност обаче тази права да бъде преброена няколко пъти. За да не допуснем това, права през точките точки ***A*(*x*1,*y*1)** и ***B*(*x*2,*y*2)** броим само когато извън отсечката ***AB*** върху правата ***AB*** няма други от дадените точки. При това достатъчно е да намерим колко са правите ***AB***, за които ако ***x*1<*x*2**, то ***y*1<*y*2** (останалите са също толкова на брой).

Сложността на този алгоритъм е ***O*(*n*2)**, където ***n*** е броят на всички точки.

#include <iostream>

**using** **namespace** std;

**int** nod(**int** a, **int** b)

**{** **int** r = a%b;

**while**(r > 0)

**{** a = b; b = r; r = a%b;**}**

**return** b;

**}**

**int** main()

**{** **int** a, b;

cin >> a >> b;

**long** **long** cnt = 0;

**for**(**int** x1=0; x1<a; x1++)

**for**(**int** y1=0; y1<b; y1++)

**for**(**int** x2=x1+1; x2<=a; x2++)

**for**(**int** y2=y1+1; y2<=b; y2++)

**{**

**int** p = x2 - x1, q = y2 - y1;

**int** d = nod(p,q);

**int** dx = p/d, dy = q/d;

**int** x0 = x1 - dx, y0 = y1 - dy;

**int** x3 = x2 + dx, y3 = y2 + dy;

**if**((x0<0||y0<0) && (x3>a||y3>b)) cnt++;

**}**

cnt = cnt \* 2 + a + b + 2;

cout << cnt << endl;

**return** 0;

**}**

***Решение 2***

Нека ***m*** е по-голямото от числата ***a*** и ***b***. Една права може да мине най-много през ***m*+1** от разглежданите точки. Нека с ***x*[2],*x*[3],…,*x*[*m*+1]** означим броя на правите, които минават съответно през **2,3,…,*m*+1** точки. Те могат да бъдат намерени по следния начин:

За всяка отсечка, която не е успоредна на координатните оси и краищата на която са две от дадените точки проверяваме колко от разглежданите точки лежат на нея. Ако краищата на тази отсечка са точките ***A*(*x*1,*y*1)** и ***B*(*x*2,*y*2)**, броят на точките, които лежат на тази отсечка е равен на **НОД(|*x*1–*x*2|,|*y*1–*y*2|)+1**. Интересуващите ни отсечки могат да бъдат разглеждани като диагоналите на всевъзможните правоъгълници със страни, успоредни на координатните оси и върхове – четири от дадените точки. При това броене обаче някои прави са броени по няколко пъти. Така например права, която минава през три точки е броена **2** пъти като права, минаваща през две точки; права, минаваща през четири точки е броена **2** пъти като права, минаваща през три точки и **3** пъти като права, минаваща през две точки и т.н. След извършване на съответните корекции получаваме истинските стойности на с ***x*[2],*x*[3],…,*x*[*m*+1].**

Окончателно търсеният брой прави е равен на ***x*[2]+*x*[3]+…+*x*[*m*+1]+*a*+*b*+2**.

Сложността на този алгоритъм е ***O*(*n*),** където ***n*** е броят на всички точки.

#include <iostream>

**using** **namespace** std;

**const** **int** mm=3100;

**long** **long** x[mm];

**int** nod(**int** a, **int** b)

**{ if** (a == 0) **return** b;

**if** (b == 0) **return** a;

**if** (a >= b) **return** nod(a-b,b);

**else** **return** nod(a,b-a);

}

**int** main()

**{** **int** a, b;

cin >> a >> b;

**int** m;

m = a;

**if** (b > m) m = b;

**for**(**int** p=1; p<=a; p++)

**for**(**int** q=1; q<=b; q++)

**{** **int** s;

s = nod(p,q);

x[s+1] = x[s+1] + 2\*(a-p+1)\*(b-q+1);

**}**

**long** **long** sum = x[m+1];

**for**(**int** i=m+1; i>=3; i--)

**{** **for**(**int** j=i-1; j>=2; j--)

x[j] = x[j] - (i-j+1)\*x[i];

sum = sum + x[i-1];

**}**

sum = sum+a+b+2;

cout << sum << endl;

**return** 0;

**}**

***Решение 3***

Нека правата ***AB*** има уравнение ***y*=*kx*+*n***. Числото ***k*** се нарича ъглов коефициент на правата. Две прави са успоредни тогава и само тогава, когато техните ъглови коефициенти са равни. Ако ***A*(*x*1,*y*1)** и ***B*(*x*2,*y*2)**, то ***k*=(*y*2–*y*1)/(*x*2–*x*1)**. Ъгловият коефициент ***k*** е равен и на тангенса на ъгъла, който правата ***AB*** сключва с положителната абсцисна полуос.

В това решение ще изброим само правите, които имат положителен ъглов коефициент, защото правите с положителен и отрицателен ъглов коефициент, които минават през дадените точки са равен брой. Възможните положителни ъглови коефициенти в разглежданата задача може да определим като броя на всички несъкратими дроби от вида ***p*/*q***, където ***p*** и ***q*** са положителни делители съответно на ***a*** и ***b***. Остава да разберем колко прави имат съответния ъглов коефициент. За целта може да броим най-левите от дадените точки (с най-малката абсциса), които могат да лежат на права със съответния ъглов коефициент. Броят на тези точки е

**(*a*–*p*+1).(*b*–*q*+1)–*max*(*a*–2.*p*+1,0).*max*(*b*–2.*q*+1,0).**

Сложността на този алгоритъм отново е ***O*(*n*),** където ***n*** е броят на всички точки, но програмата работи по-бързо от предложената в ***Решение 2***.

#include <iostream>

**using** **namespace** std;

**int** nod(**int** a, **int** b)

**{** **int** r = a%b;

**while**(r > 0)

**{** a = b; b = r; r = a%b; **}**

**return** b;

**}**

**int** main()

**{** **int** a, b;

cin >> a >> b;

**long** **long** cnt = 0;

**for**(**int** p=1; p <= a; p++)

**for**(**int** q=1; q <= b; q++)

**if**(nod(p,q)==1)

cnt=cnt+(a-p+1)\*(b-q+1)-max(a-2\*p+1,0)\*max(b-2\*q+1,0);

cnt = cnt \* 2 + a + b + 2;

cout << cnt << endl;

**return** 0;

**}**

***Решение 4***

Бързодействието на програмата може да се подобри като генерирането на ъгловите коефициенти на правите се ускори. Такава модификация на ***Решение 3*** е следващата програма. Сложността на алгоритъма отново е ***O*(*n*),** където ***n*** е броят на всички точки.

#include <iostream>

**using** **namespace** std;

**int** nod(**int** a, **int** b)

{ **int** r = a%b;

**while**(r > 0)

{ a = b; b = r; r = a%b; }

**return** b;

}

**int** main()

{ **int** a, b;

cin >> a >> b;

**if**(a>b) { **int** w = a; a = b; b = w; }

**long** **long** cnt = 0;

**for**(**int** p=1; p <= a; p++)

**for**(**int** t=1; t <= p; t++)

**if**(nod(p,t)==1)

**for**(**int** q = t; q<=b; q = q + p)

cnt=cnt+(a-p+1)\*(b-q+1)-max(a-2\*p+1,0)\*max(b-2\*q+1,0);

cnt = cnt \* 2 + a + b + 2;

cout << cnt << endl;

**return** 0;

}