**Визначення площі довільного многокутника**

За заданими координатами вершин многокутника визначити його площу.

Для обчислення площі можна використати формулу:



Обґрунтування:

1) Для трикутника:



(x1,y1)

(x2,y2)

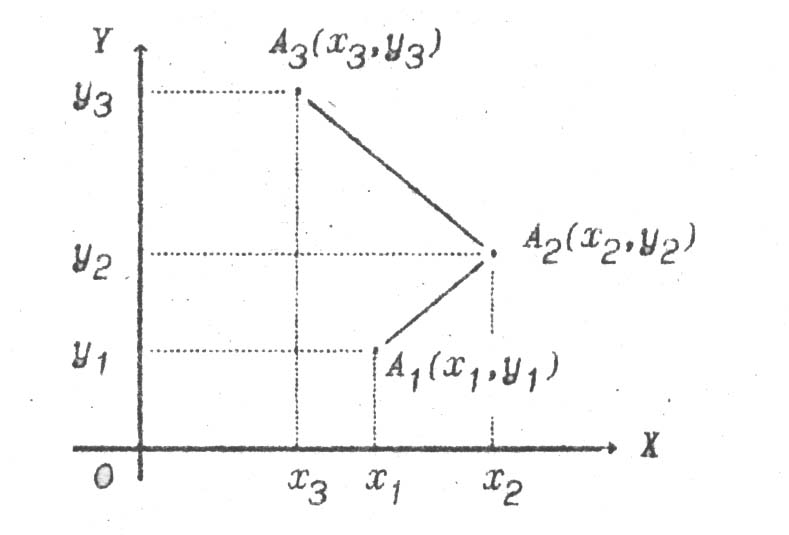
(x3,y3)

Координати векторів (x2- x1,y2-y1), (x3- x1,y3-y1).

Модуль векторного добутку рівний площі паралелограма, а ½ модуля векторного добутку - площі трикутника.

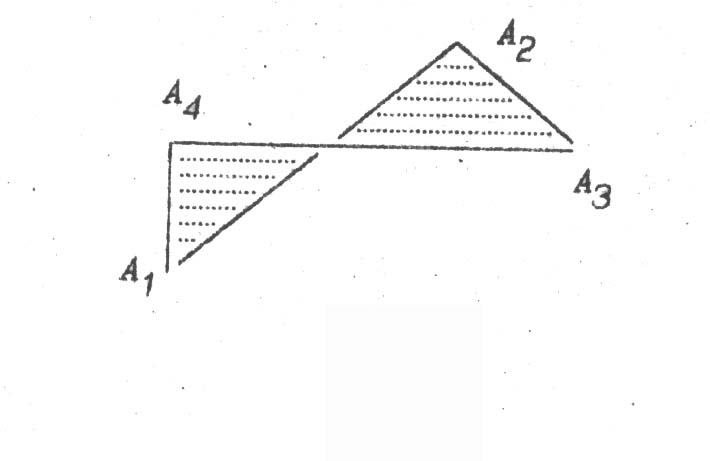


2) Структуру формули можна зрозуміти на прикладі трикутника.



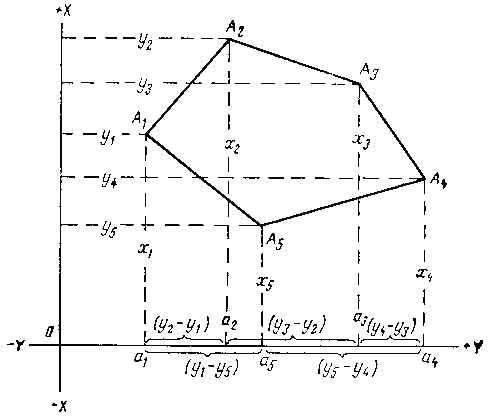
Справді, а наприклад, площа трапеції  дорівнює 

Зазначимо, площа фігури, заданої точками А1, … , Аn, n≥4, залежить від порядку, в якому задано вершини. Наприклад, якщо вершини розміщені на площині так, як показано на малюнку, то дістанемо фігуру, яка називається чотиривершинником (його площа заштрихована).



3) Структуру формули можна зрозуміти на прикладі многокутника.

Нехай потрібно визначити площу многокутника A1, A2, A3, A4, A5 з координатами вершин x1,y1; x2,y2; x3,y3; x4,y4; x5,y5. Площа многокутника S можна преставити трапеціями, у яких абсциси є основними, а різниця ординат сусідніх точок висотами.



S = a1A1A2a2 + a2A2A3a3 + a3A3A4a4 - a5A5A4a4 - a1A1A5a5.  
2S = (x1 + x2)(y2 - y1) + (x2 + x3)(y3 - y2) + (x3 + x4)(y4 - y3) +

+(x4 + x5) (y5 - y4) + (x5 + x1)(y1 - y5).

Після розкриття дужок і приведення подібних членів отримаємо

2S = x1y2 - x2y1 + x2y3- x3y2 + x3y4 - x4y3 + x4y5 - x5y4 + x5y1 - x1y5

Після винесення за дужки x1, x2, x3, x4, x5 будемо мати

2S = x1(y2-y5) + x2(y3-y1) + x3(y4-y2) + x4(y5-y3) + x5(y1-y4)

а якщо з винести за дужки y1, y2, y3, y4, y5. то будемо мати

2S = y1(x5-x2) + y2(x1-x3) + y3(x2-x4) + y4(x3-x5) + y5(x4-x1).

В скороченому вигляді ці формули можна записати так:

http://algolist.manual.ru/maths/geom/polygon/gif/s1.gif  
http://algolist.manual.ru/maths/geom/polygon/gif/s2.gif

Після перетворення отримаємо формулу в її нормальному вигляді.

|  |  |
| --- | --- |
| http://algolist.manual.ru/maths/geom/polygon/gif/sall.gif | , де X0,Y0 = Xn+1,Yn+1 |

У програмі використовуються змінні:

n – кількість вершин;

s – площа многокутника:

x[1..n+1], y[1..n+1] – масиви координати вершин;

і - змінна циклу.

На n+1 позицію заносимо координати першої вершини.

Програмний код:

**Program Prog 3;**

var x,y:array[1..100] of real;

i,n:integer;

s:real;

begin

write('n=');

readln(n);

for i:=1 to n do

readln(x[i],y[i]);

x[n+1]:=x[1];

y[n+1]:=y[1];

s:=0;

for i:=1 to n do

s:=s+(x[i]\*y[i+1]-x[i+1]\*y[i]);

s:=1/2\*abs(s);

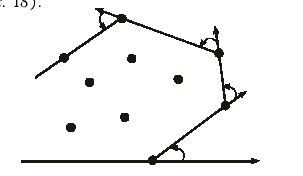
writeln('s=',s:2:2);

end.

***Побудова опуклої оболонки для множини з N точок площини.***

Опуклою оболонкою деякої заданої множини точок називається перетин всіх опуклих множин, що містять задану множину. Для скінченої множини точок опуклою оболонкою буде завжди опуклий багатокутник, всі вершини якого є точками заданої множини.

Задача полягає в тому, щоб для заданої скінченої множини точок знайти вершини опуклої оболонки цієї множини. Будемо перераховувати вершини в порядку перегляду проти годинникової стрілки. Для ефективного розв’язування цієї задачі існує декілька різних алгоритмів. Наведемо найбільш просту реалізацію одного з них – алгоритму Джарвіса. Цей алгоритм інколи називають «загортання подарунка» (див. мал. 18).



Мал. 18

Перерахунок точок шуканої межі опуклого багатокутника почнемо з правої нижньої точки *Р1*, яка належить межі опуклої оболонки. Позначимо її координати *(х1, у1).* Наступною при заданому обході буде точка *Р2(х2, у2)*. Вона має таку властивість, яку мають всі інші точки , що лежать зліва від вектора , тобто орієнтований кут між векторами  і  невід’ємний для будь-якої іншої точки *М* нашої множини. Для претендента на роль точки *Р2* перевіримо виконання умови [, ] ? 0 зі всіма точками *М*. Якщо точок, що задовольняють цю умову, декілька, то вершиною шуканого багатокутника буде та з них, для якої довжина вектора =(х2-х1, у2-у1) максимальна.

Аналогічно будемо шукати інші вершини. Припустимо, що вже знайдена *і*-та вершина *Рі(хі, уі)* опуклої оболонки. Для наступної точки *Рі+1(хі+1, уі+1)* косі добутки [] невід’ємні для всіх точок М. Якщо таких точок декілька, то вибираємо ту, для якої вектор  має найбільшу довжину. Пошук такої точки можна здійснювати так: спочатку ми можемо вважати наступною, *(і+1)*-шою, будь-яку точку. Потім знаходимо значення [], де М- всі інші точки. Якщо для однієї з них даний вираз буде менше нуля, то вважатимемо її наступною і продовжимо перевірку інших точок (аналогічно алгоритму пошуку мінімального елемента в масиві). Якщо значення виразу дорівнює нулю, то порівнюємо квадрати довжин векторів. В результаті за *О(N)* операцію ми знайдемо наступну вершину опуклої оболонки. Продовжуючи процес далі, ми повернемося до точки *Р1*. А це означає, що опукла оболонка побудована.

При розв’язуванні даної задачі у випадку цілочисельних координат ми повністю можемо уникнути використання дійсної арифметики, а тому, уникнути від втрати точності обчислень. В противному випадку в розв’язках можуть бути отримані «лишні» точки, близькі до межі опуклої оболонки , або не враховані деякі з «потрібних» точок. Складність даного алгоритму *О(kN)*, де *k* – кількість точок опуклої оболонки, в гіршому випадку дорівнює *N.*

Наведемо варіант реалізації даного алгоритму. Множина даних точок знаходиться в масиві, а всі точки, що належать опуклій оболонці, будемо записувати в масив b.

**type** vv=**record**

x, y: **longint**;

end;

**var** a, b: **array** [1..100] **of** vv;

min, m, i, j, k, n: **integer**;

**function** vect (a1, a2, b1,b2: vv): **longint**;

*{косий добуток векторів a1a2 і b1b2}*

**begin**

vect:=(a2.x-a1.x)\*(b2.y-b1.y)-(b2.x-b1.x)\*(a2.y-a1.y)

**end**;

**function** dist2(a1, a2: vv): **longint**;

*{квадрат довжини вектора a1a2}*

**begin**

dist2:=sqr(a2.x-a1.x)+sqr(a2.y-a1.y)

**end**;

**begin** {Main}

readln (n); *{кількість точок}*

**for** i:=1 **to** n **do**

read (a[i].x, a[i].y);

*{шукаємо праву нижню точку}*

m:=1;

**for** i:=2 **to** n **do**

**if** a[i].y<a[m].y **then** m:=i **else**

**if** (a[i].y=a[m].y) **and** (a[i].y>a[m].y) **then** m:=i;

*{запишемо її в масив опуклої оболонки b і переставимо на перше місце в масиві a}*

b[1]:=a[m];

a[m]:=a[1];

a[1]:=b[1];

k:=1;

min:=2;

**repeat**

*{шукаємо наступну вершину опуклої оболонки}*

**for** j:=2 **to** n **do**

**if** (vect(b[k], a[min], b[k],a[j])<0 **or** ((vect(b[k], a[min], b[k],a[j])=0) **and**

(dist2(b[k], a[min])< (dist2(b[k], a[j])))

**then** min:=j;

k:=k+1;

*{записана наступна вершина}*

b[k]:=a[min];

min:=1;

**until** (b[k].x=b[1].x) and (b[k].y=b[1].y);

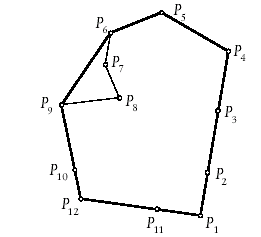
*{поки ламана не замкнеться}*

**for** j:=1 **to** k-1 **do** *{друк результатів}*

writeln (b[j].x,’’,b[j].y)

**end**.

Існує інший алгоритм розв’язання цієї задачі (алгоритм Грехема) з обчислювальною складністю *О(NlogN)*, оснований на попередньому сортуванні точок даної множини за значенням кута в полярній системі координат з центром в одній із точок опуклої оболонки. Тобто найбільш складним буде сортування заданих точок. Сортування точок можна виконувати за знаком косого добутку [], де *Р1 –* будь-яка вершина опуклої оболонки (наприклад, права нижня точка). У відсортованому масиві точок всі дані добутки повинні бути невід’ємними. Точки з рівними кутами ([]=0) розміщуються в порядку збільшення довжин відповідних векторів  (мал. 19).



Мал.19. Вершини опуклої оболонки множини точок *{Pi}: P1, P4, P5, P6, P9, P12.* Номери всіх точок відповідають сортуванню за полярним кутом.

Далі перегляд Грехема використовує стек, в якому зберігаються точки, що є претендентами в опуклу оболонку. Спочатку в стек заносять першу з відсортованих точок. Потім – сусідню з нею вершину опуклої оболонки. Якщо на першому із променів точок декілька, то ця точка променя *Pi,* найбільш віддалена від *P1*. Третя – точка  *Pі+1*. Нехай на вершині стека знаходиться точка *Pk*. Розглянемо наступну в порядку збільшення полярного кута точку заданої множини *Pi*. Поки частина ламаної *Pk-1, Pk, Pi* не є опуклим (див. задачу 4.2.), із стеку вилучається наступна точка *Pk*. Потім *Pi* заноситься до стеку. В момент закінчення перегляду всіх точок в стеці будуть знаходитись всі вершини опуклої оболонки. Так як будь-яка точка заноситься в стек не більше одного разу, тому час перегляду складає *О(N)*.

Наведемо приклад реалізації саме цього перегляду. Для наочності сортування зробимо методом «бульбашки». В якості стека використовується масив b. Описи змінних і функцій співпадають з наведеними вище.

**begin**

readln (n);

**for** i:=1 **to** n **do**

read(a[i].x, a[i],y);

*{шукаємо праву нижню точку}*

m:=1;

**for** i:=2 to n do

**if** a[i].y< a[m].y **then** m:=i **else**

**if** (a[i].y= a[m].y) **and**

(a[i].x> a[m].x) **then** m:=i;

*{запишемо її в масив опуклої оболонки b і переставимо на перше місце в масиві а }*

b[1]:=a[m];

a[m]:=a[1];

a[1]:=b[1];

*{інші точки сортуємо методом «бульбашки» за значенням полярного кута}*

**for** i:=n **downto** 3 **do**

**for** j:=2 **to** i-1 **do**

**if** (vect (a[1], a[j], a[1], a[j+1])<0 **or**

((vect (a[1], a[j], a[1], a[j+1])=0) **and**

(dist2(a[1],a[j]>dist2(a[1], a[j+1])))

**then**

**begin** *{b[n]-допоміжна змінна}*

b[n]:=a[j];

a[j]:=a[j+1];

a[j+1]:=b[n]

**end**;

*{шукаємо другу вершину опуклої оболонки}*

i:=2;

**while** vect(a[1], a[i+1], a[1], a[2])=0 **do**

i:=i+1;

b[2]:=a[i];

b[3]:=a[i+1];

k:=3;

**for** i:=i+2 **to** n **do**

**begin**

**while** vect(b[k-1], b[k], b[k], a[i])<=0 **do**

k:=k-1; *{видаляємо точку з стека }*

k:=k+1;

b[k]:=a[i] *{додаємо точку в стек}*

**end**;

**for** j:=1 **to** k **do** *{друк розв’язку}*

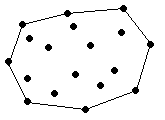
writeln (b[j].x,’’,b[j].y)

**end.**

На цьому ми закінчуємо огляд розв’язувань елементарних задач обчислювальної геометрії на площині. Їх можна використовувати при розв’язуванні більш складних, наприклад олімпіадних задач.

**Визначення опуклої оболонки**

Нехай задано деяка множина точок на площині. Опукла оболонка - найменший випуклий многокутник, який містить дані точки.



Уявимо, що площина - це дерев’яна дошка з гвіздками в кожній точці з даної множини. Тепер натягнемо навколо гвіздків резинове кільце. Воно стягнеться і утворить опуклу оболонку, як показано на малюнку вище.

Алгоритм містить наступні операції:

- визначення лівої нижньої точки;

- належність всіх точок одній півплощині відносно прямої проведеної через дві точки;

- визначення точок опуклої оболонки.

Цей алгоритм обводить точки лінією, «завертаючи» в опуклу оболонку. На малюнку дана детальна ілюстрація його дії.

Знаходимо нижню-ліву точку.

Проводимо прямі через дану точку і всі інші точки. Перевіряємо належність всіх точок одній півплощині відносно даної прямої. Якщо це справджується, то знайдена точка є вершиною многокутника і з неї шукаємо аналогічно наступну точку.

В масиві T під номером MIN відмічаємо, знайдену точку як 1.

І координати занести в масиви XN,YN під номером 1.

Повторюємо поки пошук наступної точки дає результат :

- Для кожної точки крім відмічених точок в масиві T[i],

шукаємо наступну точку, провівши через яку пряму всі інші точки лежать в одній півплощині;

* + для цього перевіряємо, чи є хоча б одна точка лежить в правій площині, якщо немає то дана точка є вершиною опуклої оболонки;
  + якщо є точки, то перевіряємо аналогічно для лівої півплощини;

- знайдену точку заносимо в масиви XN,YN під номером k+1;

- в масиві T під номером І відмічаємо, знайдену точку як k+1;

* + відмічаємо дану точку як першу і шукаємо наступну;

Виводимо послідовно точки з XN,YN.

Програмний код:

PROGRAM OBOLONKA;

VAR X,Y,XN,YN,T:ARRAY[1..100] OF INTEGER;

N,I,J,K,MIN,MAX,LICH:INTEGER;

X1,Y1,X2,Y2,XMIN,XMAX,YMIN:INTEGER;

POISK,PT:BOOLEAN;

BEGIN

WRITE('N=');

readln(n);

FOR I:=1 TO N DO READLN(X[I],Y[I]);

FOR I:=1 TO N DO T[I]:=0;

MIN:=1;

FOR I:=2 TO N DO IF X[I]<X[MIN] THEN MIN:=I;

XMIN:=X[MIN];

K:=1;

X1:=X[MIN]; Y1:=Y[MIN];

XN[K]:=X1; YN[K]:=Y1;

T[MIN]:=K;

POISK:=TRUE;

WHILE POISK DO BEGIN

POISK:=FALSE;

FOR I:=1 TO N DO

IF T[I]=0 THEN BEGIN

X2:=X[I];

Y2:=Y[I];

PT:=TRUE;

{}

FOR J:=1 TO N DO

IF (X[J]-X1)\*(Y2-Y1)-(Y[J]-Y1)\*(X2-X1)<0 THEN

PT:=FALSE;

IF NOT (PT) THEN

FOR J:=1 TO N DO

IF (X[J]-X1)\*(Y2-Y1)-(Y[J]-Y1)\*(X2-X1)>0 THEN

PT:=FALSE;

IF PT THEN BEGIN K:=K+1;

X1:=X2;

Y1:=Y2;

T[I]:=K;

XN[K]:=X1;

YN[K]:=Y1;

POISK:=TRUE;

END;

END;

END;

WRITELN;

FOR I:=1 TO K DO WRITELN(XN[I],' ',YN[I]);

END.

**Задача 10. Task10 .** Багатокутник на площині задано цілочисельними координатами своїх N вершин у декартовій системі координат. Потрібно знайти площу многокутника. Сторони багатокутника не стикаються (за винятком сусідніх - у вершинах) і не перетинаються.

введення

У першому рядку знаходиться число N. У наступних N рядках знаходяться пари чисел - координати точок. Якщо з'єднати точки в даному порядку, а також першу і останню точки, вийде заданий многокутник.

висновок

Вивести одне число - площа багатокутника. Його слід округлити до найближчого числа з однією цифрою після десяткової крапки.

обмеження

3 ≤ N ≤ 50 000, координати вершин цілі і по модулю не перевищують 20000.

|  |  |
| --- | --- |
| **task10.in** | **task10.in** |
| 4 5 0 0 5 -5 0 0 -5 | 4 0 4 0 0 3 0 1 1 |
| **task10.out** | **task10.out** |
| 50.0 | 3.5 |

**Задача11.task11. Нова держава.**

Іваном Річкоплавцем і його командою був відкритий новий багатий континент, який назвали Іванія, де вони вирішили оселитися. Кожним членом команди було засноване нове поселення і назване його іменем. Їхні нащадки вирішили утворити нову державу і побудувати кордон, таким чином щоб границя була мінімальної довжини у вигляді прямих відрізків, які з’єднають поселення, біля яких пройде кордон при умові, що в середині будуть всі інші поселення і площа країни буде найбільшою. Яка довжина границі утвореної держави і її площа?

**Вхідні дані**

В першому рядку знаходиться натуральне N (3<=N<=50) – число поселень. В наступних рядках координати кожного поселення X,Y, які записуються через пропуск. X,Y – цілі числа (-1000<=X<=1000; -1000<=Y<=1000).

**Вихідні дані**

В першому рядку Р, а в другому S. P i S – дійсні числа із виведеними двома розрядами після коми.

Максимальний час роботи 15 секунд

|  |  |
| --- | --- |
| Приклад вхідного файлу task11.in | Приклад вихідного файлу task11.out |
| 5  0 0  0 2  1 1  2 2  2 0 | 8.00  4.00 |