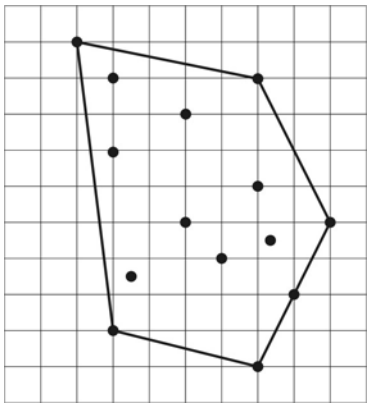
Înfăşurători convexe

1. **Preliminarii**



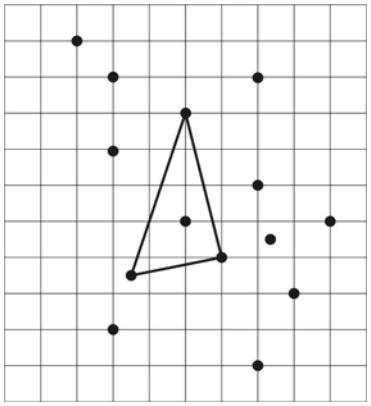
Problema determinării (calculării) înfăşurătorii convexe este una dintre problemele centrale ale geometriei computaționale. Ea apare atît în cadrul unor aplicații economice, financiare, arhitecturale, cît şi în unele probleme analitice de geometrie.

Noțiunea de înfăşurătoare convexă în plan este intuitiv simplă: pentru o mulțime S de puncte ale planului, înfăşurătoarea convexă este mulțimea de vârfuri ale poligonului convex[1] cu cea mai mică arie, care conține toate punctele mulțimii S. Înfăşurătoarea convexă poate fi modelată cu ajutorul inei benzi elastice, întinse în jurul mulțimii S. La eliberare, banda elastică va repeta conturul înfăşurătorii convexe (des. 1).

2. Algoritmi

Vor fi cercetați doi algoritmi de construire a înfăşurătorii convexe în plan. Primul algoritm permite o realizare extrem de simplă, dar cere efectuarea unui număr mare de operații (complexitatea[2] algoritmului este $O(n^4)$). Cel de al doilea este un algoritm eficient, cu complexitatea $O(n \log n)$, dar cu o implementare ceva mai voluminoasă (Algoritmul Graham).

Algoritmul elementar.



Algoritmul se bazează pe o două afirmații simple:

a. Înfăşurătoarea convexă a unei mulțimi de puncte S este formată din punctele extreme ale mulțimii S
b. Un punct p al mulțimii S **nu** este un punct extrem atunci şi numai atunci, când există cel puțin un triunghi, format din punctele din S, astfel încât p să se afle în interiorul lui (des 2.).

În baza acestor afirmații, pseudocodul algoritmului capătă forma:

a) $C \leftarrow S$
b) Pentru toți $p_1 \in S$
  Pentru toți $p_2 \in S, p_2 \neq p_1$
    Pentru toți $p_3 \in S, p_3 \neq p_1, p_3 \neq p_2$
      Pentru toți $p_4 \in S, p_4 \neq p_1, p_4 \neq p_2, p_4 \neq p_3$
        Dacă $p_4 \in \Delta p_1 p_2 p_3$, atunci $C \leftarrow C / p_4$

Verificarea apartenenței punctului la triunghi poate fi realizată într-un număr constant de operații.

Realizarea acestui algoritm este propusă cititorilor.

---

[1] Poligonul P - convex, dacă $\forall x_1, x_2 \in P, [x_1, x_2] \in P$

[2] Modul de calcul al complexității unui algoritm poate fi găsit în: A. Gremalschi. Informatica. Tehnici de programare. Manual pentru clasa a 11. Ştiința, 2003.

Algoritmul Graham. Varianta Andrew.

Algoritmul se bazează pe următorul principiu: partea superioară (după y) a înfăşurătorii convexe este bombată (în sus) pentru oricare 3 puncte consecutive ale sale, cea inferioară – bobmbată (în jos). Pentru a realiza acest algoritm este suficient să:

a. se determine două puncte p_1(p_{fin}) din S cu abscisa minimă (maximă).
b. se determinne segmentul de dreaptă, care le uneşte
c. se separe S în S_{sup} şi S_{inf} după poziţia punctelor faţă de segmentul [p_1, p_{fin}]. S_{sup} – punctele extreme şi cele din stânga vectorului p_1p_{fin}, S_{inf} – punctele extreme şi cele din dreapta vectorului p_1p_{fin}.
d. se sorteze S_{sup}, S_{inf} după creşterea abscisei.
e. se verifice toate tripletele consecutive p_i, p_{i+1}, $p_{i+2} \in S_{sup}$. Dacă p_{i+1} se află în stânga vectorului p_ip_{i+2} se va trece la tripletul p_{i+1}, p_{i+2}, p_{i+3}. Dacă p_{i+1} se află în dreapta vectorului p_ip_{i+2} se va elimina p_{i+1} şi se va cerceta tripletul p_i, p_{i+2}, p_{i+3}. Dacă la o parcurgere de la p_1 la p_{fin} nu se elimină nici un punct, cele rămase formează partea superioară a înfăşurătoarei convexe.
f. se verifice toate tripletele consecutive p_i,p_{i+1}, $p_{i+2} \in S_{inf}$ Dacă p_{i+1}se află în dreapta vectorului p_ip_{i+2} se va trece la tripletul pi p_{i+1},p_{i+2}, p_{i+3}. Dacă p_{i+1} se află în stânga vectorului p_ip_{i+2} se va elimina p_{i+1} şi se va cerceta tripletul p_i, p_{i+2}, p_{i+3}. Dacă la o parcurgere de la p_1 la p_{fin} nu se elimină nici un punct, cele rămase formează partea inferioară a înfăşurătoarei convexe.
g. $C \leftarrow S_{sup} \cup S_{inf}$

Procedura care realizează acest algoritm este propusă mai jos. Se presupune că punctele sunt date prin coordonatele lor – numere întregi. Numărul de puncte nu depăşeşte 1000. Sortarea este realizată de procedura qsort, care nu este descrisă în paragraful respectiv, fiind considerată elementară. Structurile de date:

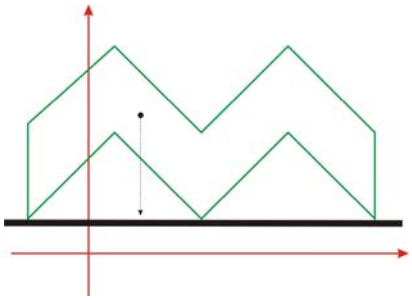
```
type  nod = record x,y :longint; end;
      pol = array [1..1001] of nod;
var   sus,jos, drag: pol;
      count,cmx, cmy,n,i: integer;

procedure conv;
var   minx,maxx:longint;
      j, imin, imax,i, nsus, njos: integer;
      rem: boolean;
      p1,p2,p3:nod;
begin
{1. determinarea extremelor dupa x}
minx:=drag[1].x; maxx:=drag[1].x;imin:=1; imax:=1;
      for i:=2 to n do
          if drag[i].x< minx then
              begin  minx:=drag[i].x; imin:=i; end
          else if drag[i].x>maxx then
              begin  maxx:=drag[i].x; imax:=i; end;
{2. separarea in submultimi}
nsus:=1; njos:=1; sus[1]:=drag[imin]; jos[1]:=drag[imin];
for i:=1 to n do
  if not (i in [imin, imax])then
       begin
         if sarrus(drag[imin],drag[imax],drag[i])<0
            then begin inc(njos); jos[njos]:=drag[i]; end;
         if sarrus(drag[imin],drag[imax],drag[i])>0
            then begin inc(nsus); sus[nsus]:=drag[i]; end;
       end;
inc(nsus); sus[nsus]:=drag[imax]; inc(njos); jos[njos]:=drag[imax];
```

```
{3. sortarea subseturilor }
qsort(sus,2,nsus-1);
qsort(jos,2,njos-1);
{crearea invelisului convex}
{4. sus}
repeat
      rem:=false; i:=2;
      while i<nsus do
          begin
            p1:=sus[i-1]; p2:=sus[i]; p3:=sus[i+1];
            if sarrus(p1,p3,p2)>0 then  i:=i+1
                else begin
                       rem:=true;
                       for j:=i to nsus-1 do sus[j]:=sus[j+1];
                       dec(nsus);
                     end;
          end;
until not rem;
{5. si jos}
repeat
      rem:=false; i:=2;
      while i<njos do
          begin
            p1:=jos[i-1]; p2:=jos[i]; p3:=jos[i+1];
            if sarrus(p1,p3,p2)<0 then  i:=i+1
                else begin
                       rem:=true;
                       for j:=i to njos-1 do jos[j]:=jos[j+1];
                       dec(njos);
                     end;
          end;
until not rem;
{6. asamblarea}
       for i:= nsus-1 downto 2 do
         begin inc(njos); jos[njos]:=sus[i]; end;
       drag:=jos; n:=njos;
end;{conv}
```

3. Probleme rezolvate

Dragon (.campion 2006 - 2007)

Enunț



Dănuț este pasionat de modelarea diferitor carcase din sârmă. După modelare el dă carcaselor numele unor dragoni, în dependență de proprietățile acestora. Recent el a modelat dragonul Hotgar. Dragonul Hotgar e vestit prin faptul că doarme permanent. Având forma unui poligon simplu, el nu se culcă pentru a dormi, ci doarme vertical pe suprafața mesei pe care a fost construit. Pentru somn dragonul are nevoie de o poziție, în care centrul de masă se află strict între 2 puncte de contact cu suprafața mesei. În timpul somnului dragonul nu-şi schimbă poziția. Centrul de masă este întotdeauna un punct interior al dragonului, şi nu coincide cu nici un vârf al carcasei.

Cerință

Scrieți un program, care va determina numărul pozițiilor în care poate dormi dragonul Hotgar

Date de intrare

Prima linie a fişierului de intrare conţine trei numere întregi separate prin spaţiu: N ($3 \le N \le 1000$) - numărul de vârfuri în carcasa dragonului şi x_c,y_c ($-1000 \le x_c,y_c \le 1000$) – coordonatele centrului de masă a dragonului.

Urmează N linii ce conţin câte 2 numere întregi x_i,y_i ($-1000 \le x_i,y_i \le 1000$), separate prin spaţiu – coordonatele vârfurilor poligonului în ordinea parcurgerii lor.

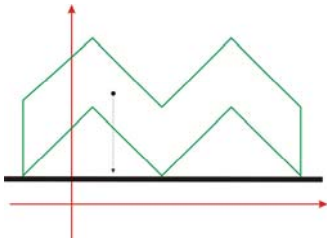
Date de ieşire

Fişierul de ieţire va conţine un număr întreg - numărul poziţiilor „de dormit” ale dragonului

Exemplu

drag.in	drag.out
10 6 16 3 14 13 4 23 14 33 4 33 14 23 24 13 14 3 24 -7 14 -7 4	3

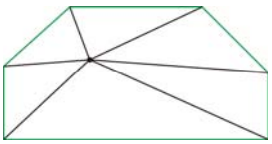
Rezolvare



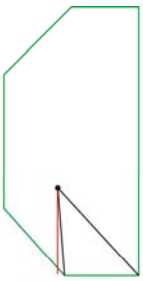
Deoarece dragonul se sprijină pe careva puncte extreme ale poligonului, problema poate fi divizată în 2 subprobleme relativ simple:

1. Determinarea înfăşurătoarei convexe a vârfurilor poligonului.
2. Determinarea prezenţei unui unchi optuz într-un triunghi

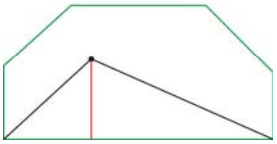
1. Prima subproblemă este rezolvată cu ajutorul algoritmului descris anterior.
2. După determinarea înfăşurătorii convexe, problema poate fi reformulată în felul următor:



Este dat un poligon convex P şi un punct interior C, unit prin linii drepte cu cu vârfurile poligonului. În câte dintre triunghiurile formate înălţimea dusă din punctul C se proiectează într-un punct interior al laturei poligonului care aparţine triunghiului?



Evident, înălţimea dusă din C cade în interiorul laturei dacă nici unul dintre unghiurile, alăturate laturei poligonului nu este optuz. Verificarea unghiurilor se realizează elementar cu ajutorul teoremei cosinusurilor. Complexitatea etapei este liniară după numărul de vârfuri.



Cod

Soluţia problemei în limbajul de programare Pascal poate fi găsită pe adresa web: `campion.edu.ro`

4. Probleme pentru rezolvare

Turnuri de veghe

Enunț

Flatlandia este o țară, care se extinde permanent. Pentru a apăra frontierele sale, după fiecare extindere, în noile puncte extreme ale frontierei se construiesc turnuri de veghe. Există N turnuri de veghe, date prin coordonatele lor (x_i,y_i) – numere întregi. Regele Flatlandiei, Bytezar a decis sa trimită la vatră garnizoanele turnurilor, care nu se mai află la hotarele țării.

Cerință

Scrieți un program, care va determina câte turnuri vor fi lipsite de garnizoane.

Date de intrare

Prima linie a fişierului de intrare conține un număr întreg: N $(1 \le N \le 10000)$ - numărul de turnuri de veghe în Flatlandia.

Urmează N linii ce conțin câte 2 numere întregi x_i,y_i $(-1000 \le x_i,y_i \le 1000)$, separate prin spațiu – coordonatele turnurilor de veghe.

Date de ieşire

Fişierul de ieșire va conține un număr întreg - numărul de turnuri care pot fi lipsite de garnizoane

Exemplu

turn.in	Turn.out
10 2 1 3 4 6 3 6 6 8 5 10 3 11 7 12 6 9 11 15 8	5

Bibliografie

1. F.P.Preparata, M.I.Shamos. Computational Geometry. An Introduction. Springer-Verlag, 1985
2. A. Gremalschi. Informatica. Tehnici de programare. Manual pentru clasa a 11. Ştiința, 2003.
3. E. Cerchez, M.Şerban. Programarea în limbajul C/C++ pentru liceu. Vol. 3. Iaşi, Polirom, 2006