Nuclee

1. Preliminarii

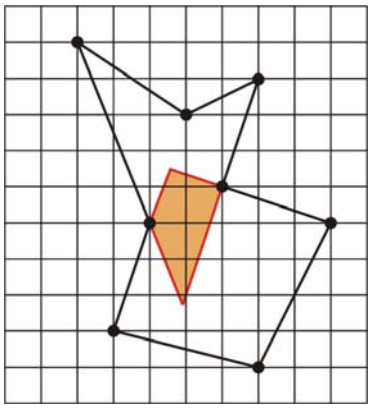
Printre problemele geometrice de calcul se numără şi problema *nucleului* poligonului simplu. Nuclueul unui poligon simplu P este, în general, format din mulţimea de puncte Q ($Q \subseteq P$), astfel încît:

$$\forall x \in Q, \forall y \in P, [x, y] \in P$$

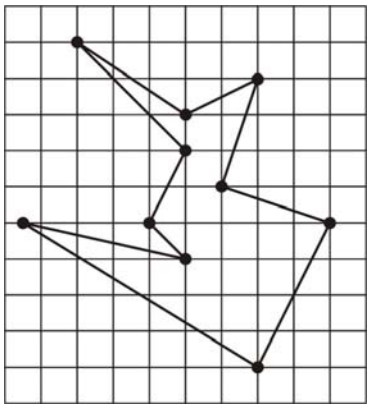
Cu alte cuvinte, nucleul poligonului este mulţimea de puncte ale poligonului, care, fiind unite cu orice alt punct al poligonului, formează un segment, ce aparţine în întregime domeniului interior al poligonului (des. 1a).

Nu întotdeauna un poligon simplu are nucleu nevid(des. 1.b)

Nucleul poligonului (dacă el există) este şi el un poligon, dar, spre deosebire de poligonul iniţial, este întotdeauna convex.



Des. 1a Nucleul poligonului simplu este figura interioară, colorată



Des. 1b Poligon simplu fără nucleu

2. Algoritm

Fie dat un poligon simplu P cu N vârfuri. Se va considera că poligonul este descris prin lista vârfurilor sale (p_1, p_2, p_3, ... ,p_{N-1}, p_N), parcurse în direcţia mişcării acelor de ceasornic. Fiecare vârf p_i este descris de coordonatele sale (x_i, y_i).

Algoritmul de determinare a nucleului necesită o construcţie secundară: un poligon convex Q, care, iniţial, conţine în sine poligonul P. Pentru determinarea poligonului Q pot fi abordate două metode:

a. Construirea înfăşurătorii convexe pentru P[1]
b. Construirea unui dreptunghi, care conţine P în interiorul său.

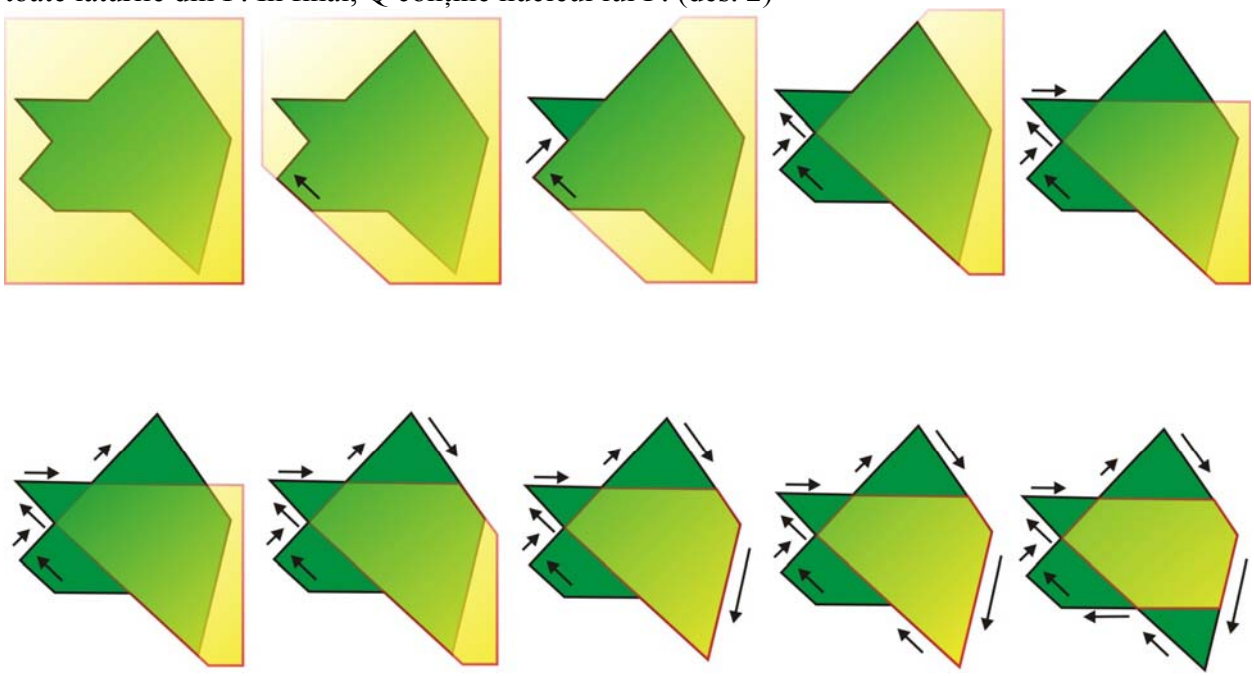
Cea de a doua metodă este mult mai simplă, şi se reduce la determinarea valorilor minime şi maxime ale abscisei şi ordonatei dintre coordonatele vârfurilor. În calitate de Q poate fi luat dreptunghiul cu vârfurile (x_{max}+1, y_{max}+1) (x_{max}+1, y_{min} -1) (x_{min} -1, y_{min}-1) (x_{min}-1, y_{max}+1). Extinderea valorilor extreme nu este o condiţie obligatorie.

După construcţia poligonului Q poate fi realizată nemijlocit conctrucţia nucleului. Metoda (cel puţin descrierea ei în limbaj uman) este foarte simplă:

Se analizează consecutiv laturile poligonului P parcurse în direcţia mişcării acelor de ceasornic. Latura curentă (p_i, p_{i+1})se cercetează ca o dreaptă l. Se determină partea poligonului Q, care se

[1] Algoritmul respectiv a fost descris în revista Delta, nr 4., pp 24 – 25.

află în stânga vectorului $\overrightarrow{(p_i p_{i+1})}$ şi se exclude din Q. Procedeul se repetă până nu sunt analizate toate laturile din P. În final, Q conține nucleul lui P. (des. 2)



Des. 2 Formarea consecutivă a nucleului poligonului

Pseudocod

Pasul 1. La sfârțitul listei de vârfuri (după p_N) se adaugă vârful p_1 (el va avea indicele N+1); Se construieşte poligonul Q (nucleul inițial), conform uneia din metodele descrise anterior.

Pasul 2. Fie la pasul ***i*** Q=(q_1, q_2, ... q_k). Pentru dreapta ce conține latura *i* a poligonului P definită de vârfurile (p_i, p_{i+1}) se determină punctele de intersecție cu poligonul Q şi laturiile intersectate.

Fie punctele de intersecție cu Q d_1, d_2, iar laturile intersectate - (q_j, q_{j+1}) şi (q_l, q_{l+1})[2].

a. Se formează Q_1= (d_1, q_{j+1},..., q_l,d_2) şi Q_2= (d_2, q_{l+1},...q_1,q_2, ... q_j,d_1)
b. Se determină Q^* care se află în dreapta vectorului $\overrightarrow{(p_i p_{i+1})}$.
c. $Q \leftarrow Q^*$

Pasul 3. Pasul 2 se repetă pentru valorile *i* de la 1 la N.

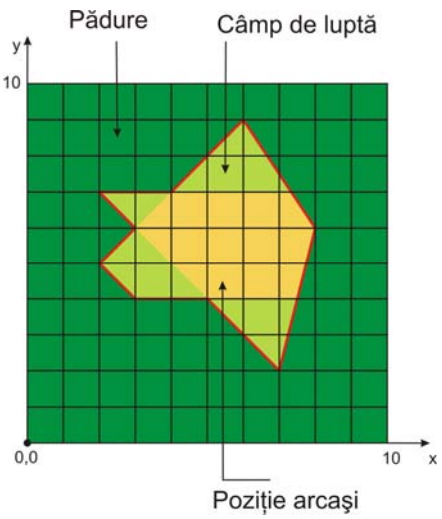
Complexitatea algoritmului este de O(N^2). Se prelucrează N laturi a poligonului P. Pentru fiecare latură se verifică intersecția cu maximum N laturi ale poligonului Q. Fiecare intersecție este determinată într-un număr constant de operații.

[2] dacă dreapta nu intersectează poligonul Q, se verifică doar poziția Q față de $\overrightarrow{(p_i p_{i+1})}$. Dacă e poziționat în stânga, nucleul final este vid, altfel Q nu se modifică la acest pas.

3. Exemplu implementare

Problema Arcaş (finala .campion, 2006)

Secretul victoriilor faimosului comandant de oşti MegaFlop este strategia lui de alegere a poziției arcaşilor pe câmpul de luptă. Câmpul de luptă are forma unui poligon simplu şi e înconjurat de păduri. MegaFlop plasează arcaşii doar pe poziții din care este **văzut** tot câmpul de luptă. Se consideră că arcaşii văd tot câmpul, dacă din orice punct care aparține poziției lor de tragere se poate trage cu săgeata în orice alt punct al câmpului. Traiectoria săgeții este liniară. Nimerind în pădure, săgeata se pierde. Pentru tragere, fiecare arcaş are nevoie de o unitate de suprafață. Astfel, numărul maxim de arcaşi, care pot fi plasați pe poziții este determinat de aria poligonului din care este văzută toată câmpia. (vezi figura alăturată).



Cerință

Scrieți un program, care determină numărul maxim de arcaşi care pot fi plasați pe poziții pe câmpul de luptă.

Date de intrare

Fişierul de intrare `arcas.in` va conține pe prima linie un număr întreg `N` – numărul de vârfuri ale poligonului simplu, care descrie perimetrul câmpului de luptă. Urmează `N` linii care conțin coordonatele vârfurilor poligonului parcurse în sensul acelor de ceasornic, câte un vârf pe linie. Linia `i+1` conține două numere întregi x_i, y_i, separate prin spațiu – coordonatele vârfului `i`.

Date de ieşire

Fişierul de ieşire `arcas.out` va conține un singur număr întreg: numărul maxim de arcaşi, care pot fi plasați pe poziții.

Restricții

$3 \le N \le 1\ 000$

$0 < x_i, y_i \le 10000$

Exemplu

arcas.in (desen)	arcas.out
9 2 5 3 6 2 7 4 7 6 9 8 6 7 2 5 4 3 4	11
arcas.in	**arcas.out**
5 1 3 2 6 2 3 5 6 3 1	0

Timp maxim de execuție/test: 0.1 secunde.
Cod:

```
program arcas;
 type
   lat=record x,y:real; end;
   pol=array[0..1001] of lat;
 var nuc,camp:pol;
     i,nnuc,ncamp:integer;
     xint,yint:real;

procedure init;
  var square : array[1..5,1..2] of integer;
      i: integer;
  begin {initializare nucleu}
  nuc[1].x:=0;  nuc[1].y:=0;
  nuc[2].x:=0;  nuc[2].y:=10001;
  nuc[3].x:=10001; nuc[3].y:=10001;
  nuc[4].x:=10001; nuc[4].y:= 0;
  nuc[5].x:=nuc[1].x; nuc[5].y:= nuc[1].y;
  nnuc:=4;
  {initializare câmp de lupta}
  readln(ncamp);
  for i:=1 to ncamp do readln(camp[i].x,camp[i].y);
  camp[ncamp+1].x:=camp[1].x; camp[ncamp+1].y:=camp[1].y; {repetare varf 1}
end;

function intersect(al,bl,cl,ad,bd,cd:real;i,j:integer): boolean;
 begin {determină intersecția dreptei și laturii + coordonate pt intersecție}
     if Ad*Bl=Bd*Al then begin intersect:=false; exit; end;
     {intersectii cu laturi adiacente}
     if (camp[j+1].x=nuc[i].x) and (camp[j+1].y=nuc[i].y) then
       begin intersect:=true; xint:=nuc[i].x; yint:=nuc[i].y; exit; end;
     if (camp[j].x=nuc[i+1].x) and (camp[j].y=nuc[i+1].y) then
       begin intersect:=true; xint:=nuc[i+1].x; yint:=nuc[i+1].y;exit; end;
    {nu sunt paralele}
     if (Ad*Bl<>Bd*Al) then
          begin
            if Al<>0 then
                              begin
                               yint:=(Ad*Cl-Cd*Al)/(Bd*Al-Ad*Bl);
                               xint:=(-Bl*yint-Cl)/Al;
                              end
                        else
                              begin
                               yint:=-Cl/Bl;
                               xint:=(-Bd*yint-Cd)/Ad;
                              end;
          end;
     if (((xint>=nuc[i].x) and (xint<=nuc[i+1].x)) or ((xint>=nuc[i+1].x)
         and (xint<=nuc[i].x))) and(((yint>=nuc[i].y) and (yint<=nuc[i+1].y))
         or ((yint>=nuc[i+1].y) and (yint<=nuc[i].y)))
     then intersect:=true else intersect:=false;
 end;

function sarrus(a,b,c:lat):real;
  begin
      sarrus:=a.x*b.y+b.x*c.y+a.y*c.x-c.x*b.y-b.x*a.y-c.y*a.x;
  end;

procedure cut(j:integer);
```

```
 var al,bl,cl,ad,bd,cd:real;
     inter: array[1..4] of lat;
     index: array[1..4] of integer;
     k,ii,i,nr:integer;
     copy: pol;
begin
  Ad:=camp[j+1].y - camp[j].y; Bd:=camp[j].x-camp[j+1].x;
  Cd:=camp[j+1].x*camp[j].y-camp[j].x*camp[j+1].y;
  nr:=0;
for i:=1 to nnuc do
 begin
  Al:=nuc[i+1].y -nuc[i].y; Bl:=nuc[i].x-nuc[i+1].x;
  Cl:=nuc[i+1].x*nuc[i].y-nuc[i].x*nuc[i+1].y;
  if intersect(al,bl,cl,ad,bd,cd,i,j) then
                  begin nr:=nr+1;
                        inter[nr].x:=xint;
                        inter[nr].y:=yint;
                        index[nr]:=i;
                  end;
 end;
 if nr>=2 then
    begin
        if sarrus(camp[j],camp[j+1],nuc[index[1]])>=0 then
           begin
             ii:=1;
             copy[1].x:=inter[1].x; copy[1].y:=inter[1].y;
             for k:=index[1]+1 to index[2] do
                                          begin
                                            inc(ii); copy[ii]:=nuc[k];
                                          end;
             inc(ii); copy[ii].x:=inter[2].x; copy[ii].y:=inter[2].y;
             nnuc:=ii;
             inc(ii); copy[ii].x:=inter[1].x; copy[ii].y:=inter[1].y;
           end
         else
           begin
             ii:=0;
             for k:=1 to index[1] do begin inc(ii); copy[ii]:=nuc[k];end;
             inc(ii); copy[ii].x:=inter[1].x; copy[ii].y:=inter[1].y;
             inc(ii); copy[ii].x:=inter[2].x; copy[ii].y:=inter[2].y;
             for k:=index[2]+1 to nnuc do
                          begin inc(ii); copy[ii]:=nuc[k]; end;
             nnuc:=ii;
             inc(ii); copy[ii]:=nuc[1]
           end;
           nuc:=copy;
    end
     else
       if sarrus(camp[j],camp[j+1],nuc[1])>=0 then
              begin writeln(0); close(output); halt; end;
 end;

function area(a:pol;n:integer):integer; {aria poligonului simplu}
 var s:real; i:integer;
 begin
     s:=0;
     for i:=1 to n do s:=s+(a[i+1].x-a[i].x)*(a[i+1].y+a[i].y)/2;
     area:=trunc(s);
 end;

begin
```

```
  assign(input,'arcas.in');  reset(input);
  assign(output,'arcas.out');  rewrite(output);
  init;
  for i:=1 to ncamp do cut(i);
  writeln(area(nuc,nnuc));  close(output);
end.
```