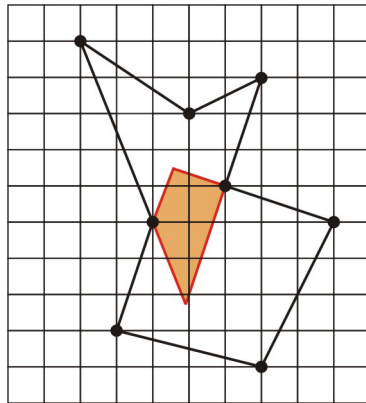


\

Sergiu CORLAT

**Algoritmi și probleme
de geometrie computațională**



Aprobat la ședința Senatului Universității de Stat din
Tiraspol din 26 ianuarie 2009.

Toate drepturile asupra acestei ediții aparțin autorului.

Reproducerea integrală sau parțială a textului sau a
ilustrațiilor din această ediție este permisă doar cu acor-
dul scris al autorului.

Autor: *Sergiu Corlat*, lector superior, UST

Recenzenți: *Andrei Braicov*,
doctor, conferențiar universitar, UST
Emanuela Cerchez,
profesor gradul I, Liceul de Informatică
„Grigore Moisil”, Iași

Redactor: *Tatiana Rusu*

Coperta: *Valentina Stratu*

© Sergiu Corlat, 2009

© Editura *Prut Internațional*, 2009

Editura *Prut Internațional*, str. George Enescu, nr. 6,
bl.1, Chișinău MD 2064
Tel.: 75 18 74, 74 93 18; fax: 50 87 20; e-mail: prut@mtc.md

CZU 519.6+514 C15

ISBN 978-9975-69-377-6

Cuprins

Introducere	5
1. Transformări de coordonate	7
1.1. Deplasări și rotații	7
1.2. Coordonate polare	9
1.3. Implementări	10
2. Intersecții	12
2.1. Intersecția dreptelor	12
2.2. Intersecția dreptei cu un segment	14
2.3. Intersecția segmentelor	16
3. Înfășurătoarea convexă	17
3.1. Algoritmul elementar	17
3.2. Algoritmul Graham	20
4. Triangularizări	26
4.1. Un algoritm greedy pentru mulțimi de puncte	26
4.2. Triangularizarea poligoanelor convexe	28
4.3. Triangularizarea poligoanelor simple	29
5. Aproximare și apartenență	31
5.1. Cea mai apropiată pereche de puncte	31
5.2. Apartenența punctului la un domeniu	35
5.3. Poligonul și diagrama Voronoi	39
5.4. Nuclee	44

6. Probleme geometrice de concurs	48
6.1. Robot	48
6.2. Robot II	50
6.3. Piatra	52
6.4. Carcasa	54
6.5. Turnuri	56
6.6. Atac	57
6.7. Evadare	61
6.8. Arcași	62
6.9. Cetate	68
6.10. Druizi	69
Notații	75
Bibliografie	76

Introducere

Geometria computațională constituie una din ramurile importante ale matematicii aplicate moderne. Pornind de la cercetarea unor elemente geometrice simple (punct, segment, dreaptă), se ajunge la o modelare complexă a figurilor geometrice în plan și a corpurilor în spațiul tridimensional.

Algoritmii geometriei computaționale stau la baza tuturor aplicațiilor de proiectare și modelare grafică (aplicații CAD, procesoare de grafică vectorială, aplicații de modelare 3D). Fără ei ar fi imposibilă proiectarea construcțiilor arhitecturale moderne, realizarea proiectelor GPS, apariția majorității absolute a produselor cinematografice de succes din ultimii ani. Enumerarea domeniilor de aplicare poate fi continuată la nesfârșit.

Domeniul vast de aplicare și multitudinea fenomenelor și situațiilor reale, descrise în termenii geometriei computaționale, au impus apariția unor probleme geometrice în cadrul concursurilor de programare de cele mai diferite nivele.

Prezenta lucrare este concepută ca un curs de inițiere în algoritmii de geometrie computațională pentru studenții facultăților de profil, pentru profesorii de informatică, precum și pentru elevii claselor liceale, participanți la concursurile naționale și internaționale de programare.

Un alt scop este de a-i forma cititorului abilități de analiză și proiectare a algoritmilor. În acest sens, algoritmii și soluțiile problemelor sunt însoțite de descrieri ale structurilor de date, fragmente de cod, de calcule ale complexității.

Copie autorizata pentru [.campion](#)

Cu toate că aparatul matematic necesar pentru rezolvarea problemelor de geometrie computațională este relativ simplu, implementarea acestuia este însoțită de necesitatea cercetării unui număr considerabil de cazuri speciale. Acesta este un motiv pentru care problemele de geometrie computațională se consideră printre cele mai dificile, în special în condiții de concurs.

Cercetarea și optimizarea soluțiilor propuse pentru unele din problemele prezente în lucrare îi vor permite cititorului să capete o experiență necesară la compartimentul rezolvare de probleme și, nu în ultimul rând, la generarea testelor pentru verificarea soluțiilor.

La baza lucrării se află o serie de articole publicate pe parcursul ultimilor ani în revista de matematică și informatică *Delta*, materiale și probleme elaborate în cadrul școlilor de vară la informatică (edițiile anilor 2001, 2002), precum și probleme propuse la concursul de pregătire continuă la informatică „.campion” ([campion.edu.ro](#)).

Țin să aduc sincere mulțumiri tuturor celor care au contribuit la apariția acestei cărți, în special colectivului Catedrei de Informatică și Tehnologii Informaționale a Universității de Stat din Tiraspol.

Ianuarie 2009

Sergiu Corlat

1. Transformări de coordonate

1.1. Deplasări și rotații

Rezolvarea oricărei probleme de geometrie computațională începe (dacă e necesar) cu deplasarea coordonatelor elementelor cercetate (de cele mai dese ori, ale punctelor) într-un domeniu potrivit al sistemului de coordonate (de obicei, în cadranul unu). În unele cazuri, deplasarea poate fi însoțită și de rotația sistemului de coordonate.

Fie într-un sistem cartezian de coordonate un punct p de coordonate (x, y) . Deplasarea de coordonate de-a lungul axei Ox cu o valoare dată a implică o modificare a coordonatelor punctului conform formulelor:

$$\begin{cases} x' = x + a, \\ y' = y. \end{cases}$$

În cazul deplasării de-a lungul axei Oy cu valoarea b , transformarea de coordonate va fi dată de sistemul:

$$\begin{cases} x' = x, \\ y' = y + b. \end{cases}$$

Modificarea „combinată” a coordonatelor cu a după axa Ox și b după Oy va fi dată de sistemul:

$$\begin{cases} x' = x + a, \\ y' = y + b. \end{cases}$$

Următorul tip de transformare este rotația cu un unghi φ a punctului (punctelor) față de originea sistemului de coordonate (fig. 1.1).

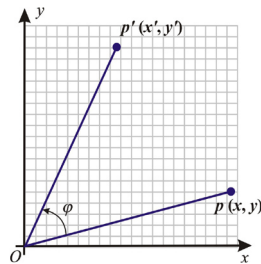


Fig. 1.1. Rotația punctului cu un unghi φ

În unele cazuri, operația poate fi realizată invers: prin rotația originii sistemului de coordonate față de un punct (sau sistem de puncte) dat. Atunci există patru numere a, b, c, d , care permit de a trece univoc coordonatele (x, y) în (x', y') , utilizând sistemul de ecuații:

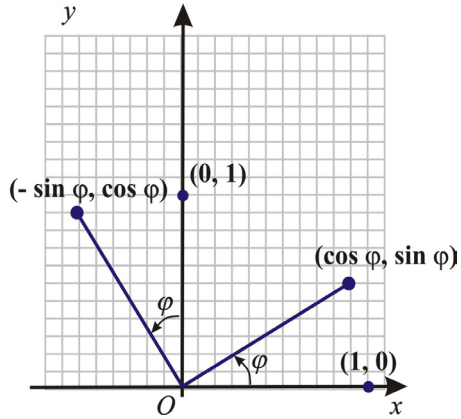


Fig.1.2. Determinarea cuadruplului a, b, c, d folosind punctele $(1, 0)$ și $(0, 1)$

$$\begin{cases} x' = ax + by, \\ y' = cx + dy. \end{cases} \quad (1.1)$$

Pentru a determina acest cuadruplu de numere, pot fi folosite punctele $(1, 0)$ și $(0, 1)$.

Se observă că la rotația cu un unghi φ punctul de coordonate $(1, 0)$ trece în punctul $(\cos \varphi, \sin \varphi)$, iar punctul de coordonate $(0, 1)$ – în $(-\sin \varphi, \cos \varphi)$ (fig. 1.2).

După substituția acestor valori, în calitate de coeficienți ai sistemului (1.1) se obține:

$$\begin{cases} x' = a \\ y' = c \end{cases} \Rightarrow \begin{cases} a = \cos \varphi, \\ c = \sin \varphi. \end{cases}$$

Analog:

$$\begin{cases} x' = b \\ y' = d \end{cases} \Rightarrow \begin{cases} b = -\sin \varphi, \\ d = \cos \varphi. \end{cases}$$

Astfel, sistemul de ecuații pentru rotația cu un unghi φ a unui punct arbitrar în jurul originii sistemului de coordonate ia forma:

$$\begin{cases} x' = x \cos \varphi - y \sin \varphi, \\ y' = x \sin \varphi + y \cos \varphi. \end{cases}$$

În cazul în care rotația cu unghiul φ a punctului de coordonate (x, y) este efectuată față de punctul de coordonate (x_0, y_0) , diferit de originea sistemului de coordonate, întâi se transferă originea sistemului de coordonate în centrul de rotație, apoi se efectuează rotația în jurul noii origini a sistemului de coordonate:

$$\begin{cases} x' - x_0 = (x - x_0) \cos \varphi - (y - y_0) \sin \varphi \\ y' - y_0 = (x - x_0) \sin \varphi + (y - y_0) \cos \varphi \end{cases}$$

sau

$$\begin{cases} x' = x_0 + (x - x_0) \cos \varphi - (y - y_0) \sin \varphi \\ y' = y_0 + (x - x_0) \sin \varphi + (y - y_0) \cos \varphi \end{cases}$$

1.2. Coordonate polare

Problemele în care apare necesitatea parcurgerii unei mulțimi de puncte $p_i, i = \overline{1, N}$ de coordonate (x_i, y_i) după unghiurile formate de vectorii $\overline{Op_i}$ cu axa Ox se rezolvă mai simplu în coordonate polare, unde fiecare punct p este determinat de perechea (r, φ) , unde r este distanța de la punct până la originea sistemului de coordonate, iar φ – unghiul format de vectorul \overline{Op} cu axa Ox (fig. 1.3).

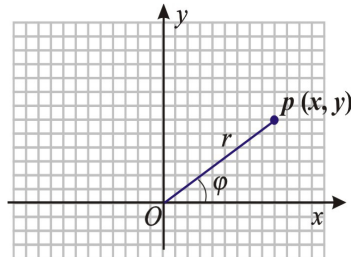


Fig. 1.3. Coordonatele polare ale punctului de coordonate carteziene (x, y)

Trecerea de la coordonatele carteziene (x, y) la cele polare este determinată de următoarele formule [1, p. 77]:

$$r = \sqrt{x^2 + y^2}, \quad \varphi = \begin{cases} \arctan \frac{x}{y} & y > 0 \\ \pi + \arctan \frac{x}{y} & y < 0 \\ \frac{\pi}{2} & x = 0, y > 0 \\ \frac{3\pi}{2} & x = 0, y < 0 \end{cases}$$

1.3. Implementări

În problemele de geometrie computațională pot apărea diferite modele de transformare a coordonatelor: deplasări după o axă, deplasări combinate (după ambele axe), rotații față de un punct, deplasări însoțite de rotații. Toate aceste transformări presupun recalcularea coordonatelor unui punct sau ale unui sistem de puncte.

Pentru rezolvarea eficientă a problemelor de geometrie este foarte important de a alege corect structurile de date. Pentru implementarea transformărilor de coordonate este necesară descrierea unei singure structuri – punctul:

```
type point=record
    x,y : real;
end;
```

Orice obiect geometric determinat de un sistem de puncte poate fi descris cu ajutorul unui tablou unidimensional sau printr-o listă alocată dinamic, cu elemente de tip `point`.

Procedura de deplasare a coordonatelor cu valoarea **vx** după axa Ox și cu valoarea **vy** după axa Oy poate fi realizată în modul următor:

```
procedure move(var P:point; vx,vy:real);
begin
  P.x:=P.x+vx;
  P.y:=P.y+vy;
end;
```

La fel de simplă este și o posibilă implementare a rotației cu un unghi **u** față de un punct de coordonate **vx, vy**:

```
procedure rotate (var P:point; u,vx,vy:real);
var
  old:point;
begin
  old:=P;
  P.x:=vx+(old.x-vx)*cos(u*pi/180)
    -(old.y-vy)*sin(u*pi/180);
  P.y:=vy+(old.x-vx)*sin(u*pi/180)
    +(old.y-vy)*cos(u*pi/180);
end;
```

2. Intersecții

În majoritatea problemelor de geometrie computațională apare în calitate de subproblemă determinarea coordonatelor punctului de intersecție a două drepte, a două segmente sau a unui segment și unei drepte.

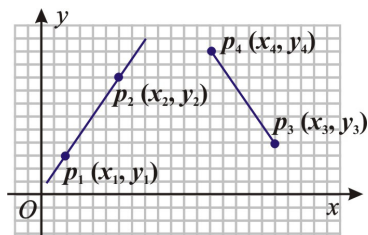


Fig. 2.1. Definirea dreptei (segmentului) prin două puncte

Metoda folosită pentru rezolvarea acestor subprobleme este aceeași, cu diferențe puțin semnificative pentru fiecare caz.

Atât pentru definirea dreptei, cât și pentru definirea segmentului se vor folosi două puncte distincte: arbitrare (ale dreptei) sau extreme (pentru segment)(fig. 2.1).

Prin urmare, atât descrierea dreptei, cât și descrierea segmentului pot fi realizate de aceeași structură de date, care va conține coordonatele a două puncte și, suplimentar, coeficienții A , B , C ai ecuației generale a dreptei. Acești coeficienți vor fi necesari pentru calculul coordonatelor punctului de intersecție. Una din posibilele metode de definire a acestei structuri este:

```
type line=record
    x1,y1,x2,y2 : real;
    A, B, C : real;
end;
```

2.1. Intersecția dreptelor

Ecuatia generală a dreptei

Fie dreapta l determinată de punctele p_1 de coordonate (x_1, y_1) și p_2 de coordonate (x_2, y_2) . Ecuația dreptei

ce trece prin aceste două puncte este:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1}. \quad (2.1)$$

Din această ecuație pot fi calculați coeficienții ecuației generale a dreptei: $Ax + By + C = 0$.

Prin transformări elementare din (2.1) se obține:

$$x(y_2 - y_1) + y(x_1 - x_2) + (x_2y_1 - x_1y_2) = 0. \quad (2.2)$$

Din (2.2) rezultă formulele de calcul pentru coeficienții ecuației generale:

$$\begin{aligned} A &= y_2 - y_1, \\ B &= x_1 - x_2, \\ C &= x_2y_1 - x_1y_2. \end{aligned} \quad (2.3)$$

Determinarea punctului de intersecție a două drepte

Fie dreptele p și l determinate de perechile de puncte $p_1(x_1, y_1)$, $p_2(x_2, y_2)$ și respectiv $p_3(x_3, y_3)$, $p_4(x_4, y_4)$.

Determinarea punctului q de intersecție a acestor drepte se reduce la rezolvarea sistemului de ecuații:

$$\begin{cases} A_1x + B_1y + C_1 = 0 \\ A_2x + B_2y + C_2 = 0, \end{cases}$$

unde $A_1x + B_1y + C_1 = 0$ este ecuația generală a dreptei p și $A_2x + B_2y + C_2 = 0$ este ecuația generală a dreptei l . Coeficienții acestor ecuații pot fi calculați conform formulelor (2.3).

Până la rezolvarea nemijlocită a sistemului urmează să se verifice dacă dreptele p și l sunt paralele sau coincid.

Pentru aceasta se verifică condițiile:

a) $A_1B_2 = A_2B_1$ & $A_1C_2 \neq A_2C_1$ – dreptele p și l sunt paralele;

b) $A_1B_2 = A_2B_1$ & $A_1C_2 = A_2C_1$ – dreptele p și l coincid.

Dacă niciuna din condițiile precedente nu se satisface, atunci există un punct unic de intersecție a dreptelor p și l , ale cărui coordonate pot fi calculate după formulele:

a) dacă $A_1 \neq 0$,

$$\begin{aligned} y_{sol} &= \frac{C_1A_2 - A_1C_2}{B_2A_1 - B_1A_2}, \\ x_{sol} &= -\frac{B_1y_{sol} + C_1}{A_1}; \end{aligned} \tag{2.4 a}$$

b) dacă $A_1 = 0$,

$$\begin{aligned} y_{sol} &= -\frac{C_1}{B_1}, \\ x_{sol} &= -\frac{B_2y_{sol} + C_2}{A_2}. \end{aligned} \tag{2.4 b}$$

Cu ajutorul setului de formule (2.4) pot fi calculate coordonatele punctului de intersecție q a dreptelor p și l .

2.2. Intersecția dreptei cu un segment

Formulele deduse în paragraful precedent permit să se calculeze coordonatele punctului de intersecție a două drepte. Cazul intersecției a două segmente sau a unei drepte și a unui segment pot fi reduse la cel al intersecției dreptelor, dar cu verificarea unor condiții suplimentare.

Fie dreapta l care trece prin punctele p_1 , p_2 și segmentul s cu punctele extreme s_1 , s_2 .

Se observă că în cazul intersecției dreptei l și a segmentului s , extremitățile segmentului sunt poziționate de părți diferite față de vectorul $\overrightarrow{p_2p_1}$. Dacă obiectele cercetate nu se intersectează, atunci ambele extremități ale segmentului se află de aceeași parte a vectorului $\overrightarrow{p_2p_1}$ (fig. 2.2).

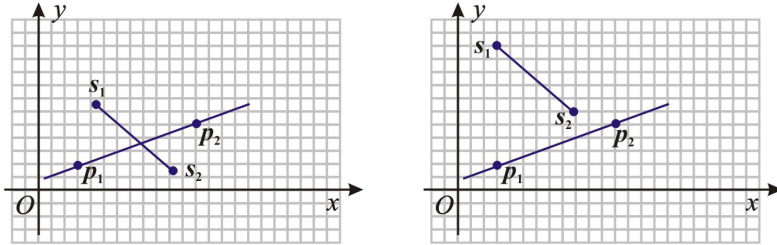


Fig. 2.2. Poziția segmentului față de dreaptă

Poziția punctului față de un vector

Fie vectorul $\overrightarrow{p_2p_1}$ de coordonate (x_2, y_2) , (x_1, y_1) și punctul s de coordonate (x_3, y_3) .

Pentru a poziționa punctul s față de vectorul $\overrightarrow{p_2p_1}$, poate fi folosit următorul determinant [12, p. 60]:

$$\Delta = \begin{vmatrix} x_2 & y_2 & 1 \\ x_1 & y_1 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}.$$

Determinantul este pozitiv, dacă punctul s este situat în semiplanul drept față de vectorul $\overrightarrow{p_2p_1}$; este nul, dacă punctul s aparține dreptei determinate de acest vector; este negativ, dacă s este situat în semiplanul stâng.

O realizare posibilă a funcției de calcul al determinantului este următoarea:

```
function sarrus(p1,p2,p3:point ): real;
begin
    sarrus:= p1.x*p2.y+p2.x*p3.y+p1.y*p3.x
            -p3.x*p2.y-p3.y*p1.x-p1.y*p2.x;
end;
```

Copie autorizata pentru .campion

Expresia $\text{Sarrus}(p_2, p_1, s_1) * \text{Sarrus}(p_2, p_1, s_2)$ va avea valoare: *pozitivă*, dacă dreapta l și segmentul s nu se intersectează (ambele extremități ale segmentului sunt situate de aceeași parte a dreptei, valorile determinantului sunt de același semn); *nulă*, dacă cel puțin una dintre extremități aparține dreptei (pentru extremitatea segmentului care aparține dreptei, determinantul este egal cu 0); *negativă*, dacă dreapta l și segmentul s se intersectează (extremitățile segmentului sunt situate de părți diferite ale vectorului, valorile determinantului au semne diferite).

Prin urmare, dacă pentru dreapta l determinată de punctele p_1 și p_2 și segmentul s cu extremitățile s_1 și s_2 expresia $\text{Sarrus}(p_2, p_1, s_1) * \text{Sarrus}(p_2, p_1, s_2)$ are valoare negativă, atunci dreapta l și segmentul s se intersectează, iar coordonatele punctului de intersecție pot fi calculate conform formulelor (2.4).

Dacă valoarea expresiei este nulă, se verifică nemijlocit care dintre extremitățile segmentului aparține dreptei. În cazul în care ambele extremități ale segmentului aparțin dreptei, întreg segmentul este conținut de aceasta.

2.3. Intersecția segmentelor

Fie segmente s și p cu extremitățile s_1, s_2 , respectiv p_1, p_2 . Pentru simplitate se va considera că segmentele nu aparțin aceleiași drepte și nu sunt paralele (cazurile date pot fi verificate cu ajutorul condițiilor pentru formulele (2.4)).

Segmentele se vor intersecta numai dacă

$$\text{Sarrus}(p_2, p_1, s_1) * \text{Sarrus}(p_2, p_1, s_2) \leq 0 \text{ și}$$

$$\text{Sarrus}(s_2, s_1, p_1) * \text{Sarrus}(s_2, s_1, p_2) \leq 0.$$

3. Înfășurătoarea convexă

Problema determinării înfășurătoarei convexe este una dintre problemele centrale ale geometriei computaționale. Ea apare atât în cadrul unor aplicații economice, financiare, ecologice, arhitecturale, cât și în probleme geometrice analitice.

Noțiunea de înfășurătoare convexă în plan este intuitiv simplă:

pentru o mulțime S de puncte ale planului, înfășurătoarea convexă $Q(S)$ este mulțimea de vârfuri ale poligonului¹ convex cu cea mai mică arie, care conține toate punctele mulțimii S . Înfășurătoarea convexă poate fi modelată cu ajutorul unei benzi elastice, întinse în jurul mulțimii S . La eliberare, banda elastică va repeta conturul înfășurătoarei convexe (fig. 3.1).

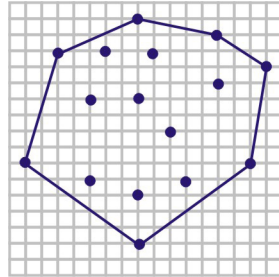


Fig. 3.1. Înfășurătoarea convexă a unei mulțimi de puncte

3.1. Algoritmul elementar

Fie mulțimea de puncte din plan $S = \{p_1, \dots, p_N\}$. Fiecare element p_i al mulțimii este descris prin coordonatele sale carteziene (x_i, y_i) . O metodă intuitivă de determinare a înfășurătoarei convexe presupune eliminarea din mulțimea inițială a tuturor punctelor ei interioare.

¹ Poligon – figură geometrică plană, închisă, formată dintr-un număr finit de segmente, numite laturi. Aici și în continuare prin poligon se va înțelege frontiera acestuia în reuniune cu interiorul său. Poligonul P este *convex*, dacă $\forall x_1, x_2 \in P, [x_1, x_2] \subseteq P$. Poligonul P este *simplu*, dacă laturile lui se intersectează doar în extremitățile lor.

Algoritmul elementar se bazează pe două afirmații simple:

- a) înfășurătoarea convexă a unei mulțimi de puncte S este formată din punctele extreme ale mulțimii S ;
- b) un punct $p \in S$ nu este un punct extrem dacă și numai dacă există cel puțin un triunghi, determinat de punctele $p_i, p_j, p_k \in S$, astfel încât p să-i aparțină (fig. 3.2).

În baza acestor afirmații, punctele interioare ale mulțimii se exclud prin cercetarea apartenenței fiecărui punct la fiecare triunghi generat de punctele mulțimii. Atât pseudocodul, cât și implementarea algoritmului sunt extrem de simple:

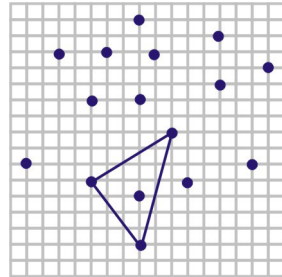


Fig. 3.2. Determinarea unui punct interior

Pseudocod

Pas 1 $C \leftarrow S$

Pas 2 Pentru toate punctele $p_i \in S$

Pentru toate punctele $p_j \in S, p_j \neq p_i$

Pentru toate punctele $p_k \in S, p_k \neq p_i, p_k \neq p_j$

Pentru toate punctele

$p \in S, p \neq p_i, p \neq p_j, p \neq p_k$

if $p \in \Delta p_i p_j p_k$ **then** $C \leftarrow C / \{p\}$

Apartenența punctului la un triunghi

Condiția $p \in \Delta p_i p_j p_k$ poate fi verificată prin determinarea poziției punctului p față de fiecare din vectorii $p_i p_j$, $p_j p_k$, $p_k p_i$. Dacă semnul valorilor returnate la apelurile funcției $\text{sarrus}(p_i, p_j, p)$, $\text{sarrus}(p_j, p_k, p)$, $\text{sarrus}(p_k, p_i, p)$ este același, atunci $p \in \Delta p_i p_j p_k$. Prin urmare, verificarea apartenenței punctului la un triunghi poate fi realizată într-un număr de operații mărginit de o constantă.

Instrucțiunile ciclice din pasul 2 al algoritmului generează un număr de operații proporțional cu N^4 , ceea ce determină și complexitatea finală a algoritmului – $O(N^4)$.

Implementare

Structura de date utilizată pentru determinarea înfășurătoarei convexe este un tablou de elemente, fiecare dintre ele fiind un articol cu trei componente: coordonatele punctului și marcajul (0 – punct extrem, 1 – punct interior), care specifică apartenența la înfășurătoarea convexă:

```
type point=record
  x,y,int: integer;
end;
```

Verificarea dacă punctul aparține la un triunghi este realizată de funcția `apart`:

```
function apart(l,i,j,k:integer) : boolean;
  var k1,k2,k3: real;
  begin
    apart:=true;
    k1:=sarrus(a[i],a[j],a[l]);
    k2:=sarrus(a[j],a[k],a[l]);
    k3:=sarrus(a[k],a[i],a[l]);
    if (k1*k2 < 0 ) or (k1*k3<0) or (k2*k3<0)
    then apart:=false;
  end;
```

Copie autorizata pentru .campion

Aplicarea marcajelor este realizată în fragmentul:

```
for i:=1 to n-2 do
  for j:=i+1 to n-1 do
    for k:=j+1 to n do
      for l:=1 to n do
        if (l<>i) and (l<>j) and (l<>k) then
          if apart(l,i,j,k) then a[l].int:=1;
```

Afișarea coordonatelor punctelor care formează înfășurătoarea se realizează prin verificarea marcajelor:

```
for i:=1 to n do if a[i].int=0
  then writeln(a[i].x, ' ', a[i].y);
```

Algoritmul descris, deși este unul polinomial, nu este cel mai eficient pentru determinarea înfășurătoarei convexe a unei mulțimi de puncte.

3.2. Algoritmul Graham

Un algoritm eficient pentru determinarea înfășurătoarei convexe a fost propus de R. L. Graham în 1972. Algoritmul se bazează pe determinarea unui punct interior al mulțimii S , deplasarea în el a originii sistemului de coordonate și sortarea celorlalte puncte p_i ale mulțimii după măsura unghiului format de vectorul $\overline{Op_i}$ cu axa Ox . După sortare, punctele din S sunt plasate într-o listă bidirecțională, circulară. La următoarea etapă se parcurge lista formată, pornind de la punctul de abscisă minimă (fig. 3.3). Acesta este un punct

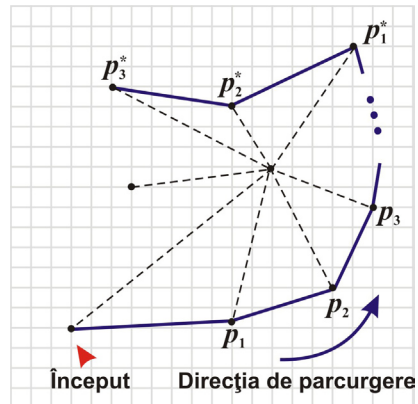


Fig. 3.3. Parcurgerea listei de puncte conform algoritmului Graham (punctele p_1 , p_2 , p_3 marchează triplețul curent)

care, în mod cert, aparține înfășurătoarei convexe. La fiecare „moment” al parcurgerii se cercetează un triplet de elemente² consecutive ale listei p_1, p_2, p_3 . Cercetarea este realizată prin verificarea poziției punctului p_2 față de vectorul $p_1 p_3$.

Poziționarea în semiplanul stâng stabilește p_2 ca fiind un punct interior al mulțimii. În acest caz, p_2 este exclus din listă, iar tripletul care urmează să fie cercetat devine p_0, p_1, p_3 (p_0 – elementul precedent pentru p_1).

Poziționarea în semiplanul drept stabilește p_2 ca fiind un punct posibil al înfășurătoarei convexe. În acest caz, p_2 este păstrat în listă, iar tripletul care urmează să fie cercetat devine p_2, p_3, p_4 (p_4 – elementul următor pentru p_3). Parcurgerea ia sfârșit când se revine în punctul de unde a început.

Pseudocod

- Pas 1** Se determină un punct interior z , $z \in S$.
- Pas 2** Se transferă originea sistemului de coordonate în punctul z .
- Pas 3** Se determină coordonatele polare (r, φ) ale fiecărui punct $p \in S$, $p \neq z$, apoi se sortează după creșterea φ (pentru punctele cu unghiuri congruente sortarea se efectuează după creșterea r).
- Pas 4** Se formează o listă bidirecțională, circulară, ale cărei elemente sunt punctele sortate (Q).
- Pas 5** Se stabilește p_0 – punctul de abscisă minimă (în sistemul cartezian de coordonate). $p \leftarrow p_0$.
- Pas 6** Cât timp la p_0 nu se ajunge prin mișcări „înainte”, se repetă:

² Fiecare element al listei descrie un punct al mulțimii S .

- a) Se consideră tripletul $p_1 \leftarrow p$, $p_2 \leftarrow p[urm]$,
 $p_3 \leftarrow p_2[urm]$.
- b) Dacă p_2 e poziționat în semiplanul drept față de vectorul $p_1 p_2$, atunci se efectuează mișcarea „înainte”: $p \leftarrow p[urm]$, altfel p_2 se exclude din lista Q și se efectuează mișcarea „înapoi”: $p \leftarrow p[prec]$.

Pas 7 Q – înfășurătoarea convexă.

Complexitatea algoritmului este $O(N \log N)$ și e determinată de complexitatea pasului 3 – sortarea punctelor după unghiul φ . Pașii 1, 2, 4, 5 au o complexitate liniară. Aceeași complexitate o are și pasul 6 – la fiecare „moment” fie este eliminat un punct, fie se realizează un pas înainte. Numărul de operații pentru verificarea poziției punctului p_2 este mărginit de o constantă.

Modificarea Andrew

Modificarea Andrew a algoritmului Graham are drept scop omiterea determinării punctului interior z , a deplasării originii sistemului de coordonate și a calculului coordonatelor polare. La baza variantei Andrew stă următorul principiu: partea superioară (după y) a înfășurătoarei convexe este bombată (în sus) pentru oricare 3 puncte consecutive ale sale, cea inferioară – bombată în jos (fig. 3.4).

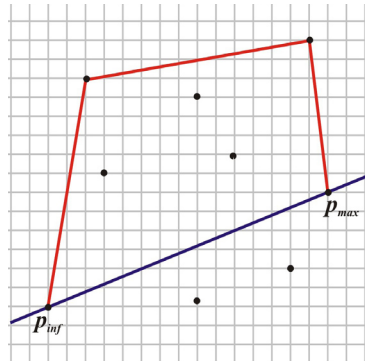


Fig. 3.4. Construirea înfășurătoarei convexe. Modificarea Andrew pentru algoritmul Graham

Pseudocodul algoritmului are următoarea formă:

- Pas 1** Se determină două puncte extreme $p_{\min}, p_{\max} \in S$ de abscisă minimă (respectiv maximă).
- Pas 2** Se separă S în S_{\sup} și S_{\inf} după poziția punctelor din mulțimea inițială față de vectorul $\overline{p_{\min} p_{\max}}$. S_{\sup} va fi formată din punctele extreme și cele din stânga vectorului, S_{\inf} – din punctele extreme și cele din dreapta vectorului.
- Pas 3** Se sortează S_{\sup}, S_{\inf} după creșterea abscisei.
- Pas 4**
- a) Se verifică toate tripletele de puncte consecutive $p_i, p_{i+1}, p_{i+2} \in S_{\sup}$, pornind de la p_{\min} .
Dacă p_{i+1} este poziționat în stânga vectorului $\overline{p_i p_{i+2}}$, **atunci** se execută mișcarea „înainte”, **altfel** – mișcarea „înapoi”. La atingerea p_{\max} , punctele rămase în S_{\sup} formează partea superioară a înfășurătoarei convexe.
- b) Se verifică toate tripletele de puncte consecutive $p_i, p_{i+1}, p_{i+2} \in S_{\inf}$, pornind de la p_{\min} .
Dacă p_{i+1} este poziționat în dreapta vectorului $\overline{p_i p_{i+2}}$, **atunci** se execută mișcarea „înainte”, **altfel** – mișcarea „înapoi”. La atingerea p_{\max} , punctele rămase în S_{\inf} formează partea inferioară a înfășurătoarei convexe.
- Pas 5** $Q \leftarrow S_{\inf} \cup S_{\sup}$.

În cele ce urmează este propusă o realizare a acestui algoritm. Se presupune că punctele sunt date prin coordonatele lor – numere întregi. Sortarea este realizată de procedura **qsort** (descrierea și implementarea poate fi găsită,

Copie autorizata pentru .campion
 de exemplu, în [7, 303]). Poziția reciprocă a punctelor este
 determinată de funcția **sarrus**, descrisă anterior.

Structurile de date:

drag – mulțimea S
sus, jos – mulțimile $S_{\text{sup}}, S_{\text{inf}}$
n – $|S|$

```

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
            if sarrus(drag[imin],drag[imax],drag[i])<0
            then
                begin inc(njos); jos[njos]:=drag[i]; end
            else
                begin inc(nsus); sus[nsus]:=drag[i]; 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 infășurătoarei convexe}
    {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;
{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;
{5. asamblarea}
for i:= nsus-1 downto 2 do
  begin
    inc(njos);
    jos[njos]:=sus[i];
  end;
drag:=jos; n:=njos;
end; {conv}

```

La finalul execuției procedurii punctele care formează înfășurătoarea convexă vor fi stocate în structura de date `jos`.

4. Triangularizări

Din punct de vedere geometric, triangularizarea $T(S)$ a mulțimii de puncte S este o divizare a înfășurătoarei convexe $Q(S)$ în triunghiuri. Suplimentar, se vor respecta condițiile:

- a) vârfuri ale triunghiurilor pot fi numai puncte din S ;
- b) toate punctele mulțimii S vor fi utilizate în calitate de vârfuri (fig. 4.1).

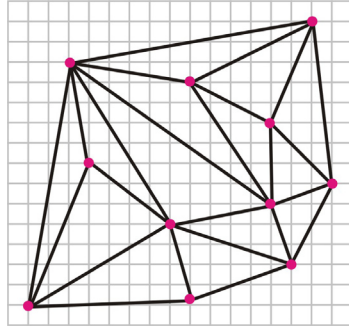


Fig. 4.1. Triangularizarea unei mulțimi de puncte

4.1. Un algoritm greedy pentru mulțimi de puncte

Fie mulțimea de puncte S și $N=|S|$. Fiecare punct $p \in S$ este descris prin coordonatele sale carteziene (x, y) . Triangularizarea $T(S)$ poate fi cercetată ca un graf planar cu mulțimea de noduri S . Prin urmare, numărul de laturi M în S este proporțional cu N (conform formulei Euler, $M \leq 3N - 6$).

O soluție simplă în plan este generarea tuturor segmentelor posibile cu extremități în S , sortarea lor după lungime, apoi adăugarea consecutivă în triangularizare. În proces se verifică dacă latura curentă intersectează laturile adăugate anterior. Dacă intersecțiile lipsesc, latura este adăugată, în caz contrar, se trece la cercetarea laturii următoare.

Numărul total de laturi posibile pe punctele din S este $N^2/2$. Fiecare latură poate fi cercetată ca un segment descris prin extremitățile sale:

```
type segment=record
    p1,p2:nod; l:real;
end;
```

Prin urmare, verificarea intersecției laturilor poate fi realizată folosind o procedură identică cu procedura de verificare a intersecției segmentelor (§ 2.3).

```
Function intersect (A,B: segment): boolean;
Begin
    if sarrus(A.p2,A.p1,B.s1)*sarrus(A.p2,A.p1,B.s2) ≤ 0 and
        sarrus(B.s2,B.s1,A.p1)*Sarrus(B.s2,B.s1,A.p2) ≤ 0
    then
        intersect:= true else intersect:= false
    End;
```

Calculul distanței³ dintre două puncte $p_i, p_j \in S$ (lungimea laturii) poate fi realizat printr-o funcție elementară:

```
Function distant (p[i],p[j]: nod): real;
Begin
    distant:=
        sqrt(sqr(p[i].x-p[j].x)+ sqr(p[i].y-p[j].y));
End;
```

Pseudocod

Pas 1 $m \leftarrow 0$

```
Pas 2 for i ← 1 to N do
    for j ← 1+i to N do
        Begin
            m ↑
            Latura[m].l ← distant(p[i],p[j])
            Latura[m].st ← p[i]
            Latura[m].fin ← p[j]
        End;
```

Pas 3 `qsort(Latura,m);`

³ Pentru două puncte a, b date prin coordonatele $(x_a, y_a), (x_b, y_b)$,
 $d(a,b) = \sqrt{|x_a - x_b|^2 + |y_a - y_b|^2}$.

Copie autorizata pentru .campion

Pas 4 $k \leftarrow 0$

```
Pas 5 for i ← 1 to M do
  begin
    z ← false
    for j ← 1 to k do
      if intersect(Latura[i], Triang[j])
        then z ← true
    if NOT z then
      begin
        k ↑ ; Triang[k] ← Latura[i]
      end;
  end;
```

Numărul total de laturi generate este proporțional cu N^2 . Sortarea lor va necesita un număr de operații proporțional cu $N^2 \log(N)$. Complexitatea pasului 5 este determinată de numărul de verificări ale intersecțiilor, care nu depășește N^3 . Prin urmare, complexitatea algoritmului greedy este $O(N^3)$, ceea ce lasă de dorit pentru valori mari ale lui N .

4.2. Triangularizarea poligoanelor convexe

Este o problemă elementară, care poate fi rezolvată în timp liniar.

Fie P un poligon convex, dat prin lista vârfurilor p_1, \dots, p_N în ordinea parcurgerii lor. Pentru a construi o triangularizare în P , este suficient să fie construite segmentele diagonale $p_1p_3, \dots, p_1p_{N-1}$ (fig. 4.2). Numărul diagonalelor este proporțional cu N (numărul de vârfuri ale poligonului). Construirea unei diagonale necesită un număr

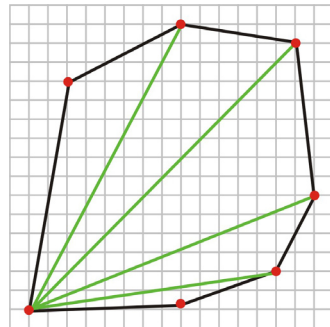


Fig. 4.2. Triangularizarea unui poligon convex

constant de operații. Triangularizarea poligonului convex poate fi utilă pentru calculul ariei lui. Aria poligonului este egală cu suma ariilor triunghiurilor din triangularizare.

4.3. Triangularizarea poligoanelor simple

Metoda greedy

În cazul poligoanelor simple nu este posibil de a realiza direct metoda din compartimentul precedent, deoarece apar două condiții suplimentare care trebuie verificate:

- 1) aparține oare latura curentă poligonului sau exteriorului poligonului triangularizat;
- 2) laturile care formează frontiera poligonului urmează să fie incluse în triangularizare indiferent de locul ocupat în lista distanțelor.

Verificarea primei condiții este echivalentă cu verificarea apartenenței mijlocului laturii la poligon. Algoritmul care rezolvă această problemă este prezentat în § 5.1.

Problema a doua se rezolvă elementar: prin includerea laturilor ce formează frontiera P în începutul listei care descrie triangularizarea.

Fie P un poligon simplu, dat prin lista de vârfuri p_1, \dots, p_N , în ordinea parcurgerii lor. Algoritmul greedy va fi descris de următorul pseudocod:

Pas 1 $m \leftarrow 0$

Pas 2 **for** $i \leftarrow 1$ **to** N **do**
 for $j \leftarrow 2+i$ **to** N **do**

Begin

$m \uparrow$

Latura[m].1 \leftarrow distant($p[i], p[j]$)

Copie autorizata pentru .campion

```
Latura[m].st ← p[i]  
Latura[m].fin ← p[j]
```

End;

Pas 3 `qsort(Latura,m);`

Pas 4 `for i ← 1 to N-1 do`

`Begin`

```
    Triang[i].st ← p[i]
```

```
    Triang[i].fin ← p[i+1]
```

`End;`

```
Triang[N].st ← p[N]
```

```
Triang[N].fin ← p[1]
```

```
k ← N
```

Pas 5 `for i ← 1 to M do`

`Begin`

```
    z ← false
```

```
    for j ← 1 to k do
```

```
      if intersect(Latura[i],Triang[j])
```

```
        then z ← true
```

```
    if NOT z then
```

```
      Begin
```

```
        x,y ← middle(Latura[i])
```

```
        if apart(x,y,P) then
```

```
          Begin
```

```
            k↑; Triang[k] ← Latura[i]
```

```
          End;
```

```
      End;
```

```
    End;
```

Calculul coordonatelor mijlocului segmentului necesită un număr constant de operații:

```
Procedure middle(a: segment; var x,y:real);
```

```
  Begin
```

```
    x:=(a.st.x+ a.fin.x)/2; y:=(a.st.y+ a.fin.y)/2;
```

```
  End;
```

Verificarea apartenenței unui punct la un poligon are o complexitate liniară față de numărul N de laturi ale acestuia. Prin urmare, complexitatea pasului 5 și complexitatea totală a algoritmului rămâne aceeași ca și pentru algoritmul descris în § 4.1.

5. Apropiere și apartenență

Capitolul este consacrat analizei unor probleme geometrice în plan: determinarea celei mai apropiate perechi de puncte, apartenența punctului la un poligon, construcția poligoanelor cu proprietăți prestabilite. Soluțiile directe ale acestor probleme sunt relativ simple, dar nu și optime.

5.1. Cea mai apropiată pereche de puncte

Fie în plan o mulțime de puncte $S = \{s_1, \dots, s_N\}$. Se cere să se determine o pereche de puncte (fig. 5.1) $s_i^*, s_j^*, i \neq j$:

$$d(s_i^*, s_j^*) = \min_{\substack{i=1, \dots, N \\ j=1, \dots, N \\ i \neq j}} d(s_i, s_j).$$

Calculul direct al tuturor distanțelor dintre N puncte necesită un număr de operații proporțional cu N^2 :

```

index1 ← 1; index2 ← 2;
min ← distance( $s_1, s_2$ );
For i ← 1 to N do
  For j ← 1+i to N do
    If distance( $s_i, s_j$ ) < min then
      Begin
        min ← distance( $s_i, s_j$ )
        index1 ← i; index2 ← j;
      End;

```

Același rezultat poate fi obținut într-un timp mai restrâns, folosind algoritmul optim cu o complexitate de $O(N \log N)$.

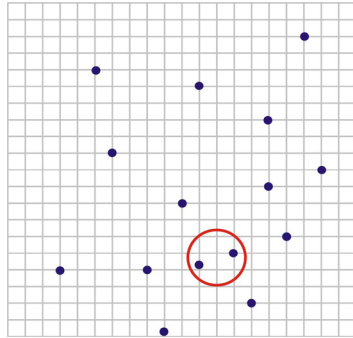


Fig. 5.1. Cea mai apropiată pereche de puncte

Algoritmul optim

Pentru a determina în timp optim cea mai apropiată pereche de puncte, poate fi folosită tehnologia recursivă *desparte și stăpânește*. Ideea majoră este divizarea, la fiecare pas recursiv, a mulțimii inițiale în două submulțimi liniar separabile și rezolvarea problemei pe fiecare submulțime în parte (fig. 5.2).

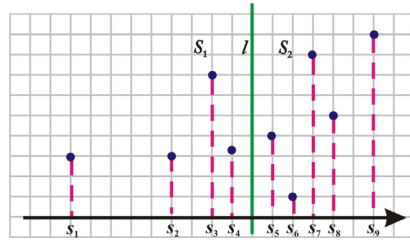


Fig. 5.2. Divizarea mulțimii în submulțimi separabile față de dreapta l pentru rezolvarea recursivă a problemei „cea mai apropiată pereche de puncte”

Cazul elementar, care permite calculul direct al soluției, este mulțimea formată din două sau trei puncte. Numărul de operații la acest pas este constant. Specificul problemei constă în determinarea soluției optime la etapa de asamblare: având două submulțimi S_1 și S_2 , cea mai apropiată pereche de puncte în $S_1 \cup S_2$ poate să fie determinată de o pereche $s', s'' : s' \in S_1, s'' \in S_2$. Se poate demonstra că, la etapa asamblării soluției, numărul necesar de verificări la fiecare nivel este liniar față de numărul de puncte din mulțimile asamblate. Numărul de divizări consecutive ale mulțimii în submulțimi „balansate”⁴ nu va depăși $\log(N)$. Dacă numărul de operații necesare pentru determinarea soluției la un nivel de asamblare este proporțional cu N , complexitatea finală a algoritmului va fi $O(N \log N)$. Pentru soluționarea optimă a problemei este necesară o preprocesare a mulțimii: sortarea punctelor după abscisă (se obține șirul sortat X) și sortarea punctelor după ordonata lor (se obține șirul sortat Y). Având complexitatea $O(N \log N)$, sortarea nu modifică complexitatea finală a algoritmului.

⁴ Numărul de elemente din fiecare submulțime diferă cu cel mult 1.

Fie la un nivel de divizare mulțimea S , șirul X al elementelor din S sortate după abscisa x , șirul Y cu elementele S sortate după ordonata y .

Aparent, procesul de asamblare a soluției la un nivel dat are o complexitate $O(N^2)$: maxim N puncte în S_1 și maxim N puncte în S_2 , dintre care urmează să fie calculate distanțele. În realitate, numărul de operații este mult mai mic.

Fie S a fost divizat în submulțimile S_1 și S_2 , pe care au fost determinate distanțele minime δ_1 și δ_2 , precum și perechile respective de puncte. Se consideră $\delta = \min(\delta_1, \delta_2)$ (fig. 5.3).

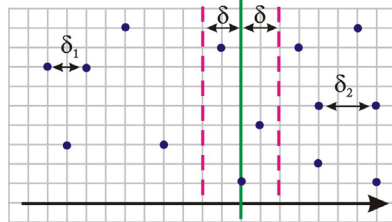


Fig. 5.3. Determinarea punctelor potențiale pentru soluția pe $S_1 \cup S_2$

Pentru determinarea soluției optime pe $S_1 \cup S_2$ este necesar de a cerceta doar punctele situate în fâșia de lățime δ de la dreapta l care separă submulțimile. Celelalte puncte din submulțimi se află, evident, la o distanță mai mare decât δ unul de altul și nu pot îmbunătăți soluția. Această restrângere nu garantează efectuarea unui număr liniar de operații, deoarece pentru un punct cercetat $s \in S_1$ în fâșia δ poate fi un număr de puncte proporțional cu $|S_2|$. O analiză mai detaliată permite excluderea din cercetare a tuturor punctelor care se află în exteriorul dreptunghiului de dimensiunile $\delta \times 2\delta$, asociat punctului s (fig. 5.4). Numărul de puncte ale mulțimii S_2 care se pot

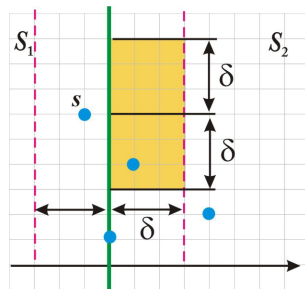


Fig. 5.4. Zona de cercetare în S_2 pentru un punct dat $s \in S_1$

Copie autorizata pentru .campion
 afla în această zonă nu poate fi mai mare decât 6. Prin urmare, numărul de verificări la etapa de asamblare nu va depăși $6N$.

Folosind șirurile X și Y , se formează șirul Y' , care conține punctele din S situate în fâșia $[l-\delta, l+\delta]$, sortate după y – mulțimea punctelor care pot îmbunătăți soluția. Pentru fiecare element din Y' se calculează distanțele doar până la următoarele 8 elemente: ele reprezintă (în cel mai rău caz) simetric 6 puncte din S_2 plus 6 puncte din S_1 minus 3 puncte care coincid, minus punctul cercetat (fig. 5.5).

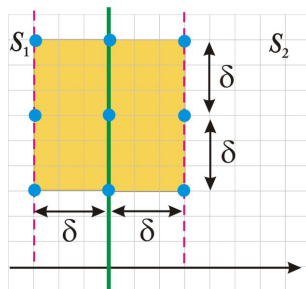


Fig. 5.5. Numărul elementelor consecutive Y' care pot modifica distanța minimă nu depășește 9

Pseudocod

Preprocesare

$X \leftarrow S, \text{sort}(X)$ {sortare după x }

$Y \leftarrow S, \text{sort}(Y)$ {sortare după y }

Procedură `apr2p(S, X, Y)`

if $|S| \geq 4$ **then**

begin formează $S_1, S_2, X_1, X_2, Y_1, Y_2$

$\delta \leftarrow \min(\text{apr2p}(S_1, X_1, Y_1), \text{apr2p}(S_2, X_2, Y_2))$

 formează Y'

for $i \leftarrow 1$ **to** Y' **do**

for $j \leftarrow 1$ **to** 8 **do**

if $\text{distance}(Y'[i], Y'[i+j]) < \delta$

then $\delta \leftarrow \text{distance}(Y'[i], Y'[i+j])$

return δ

end else return distanța minimă în S {calculată direct}

5.2. Apartenența punctului la un domeniu

Apartenența punctului la un poligon convex

Problema determinării apartenenței unui punct la un poligon convex este una simplă și poate fi rezolvată cu ajutorul algoritmilor cercetați anterior. Se observă ușor (fig. 5.6) că poziția unui punct interior față de fiecare din vectorii determinați de vârfurile consecutive ale poligonului este una și aceeași – la dreapta, dacă vârfurile sunt parcurse în direcția mișcării acelor de ceasornic, și la stânga, în cazul parcurgerii vârfurilor în direcție opusă.

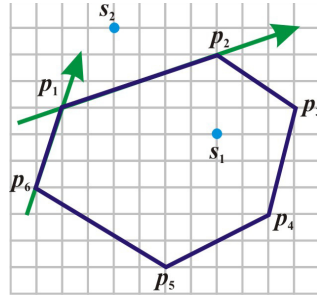


Fig. 5.6. Punctele interioare sunt plasate de aceeași parte a vectorilor determinați de vârfurile consecutive ale poligonului, poziția celor exterioare poate varia.

Poziția punctului s față de un vector $\overrightarrow{p_i p_j}$ poate fi determinată în timp constant (§ 3.1). Prin urmare, complexitatea algoritmului este proporțională doar cu numărul de laturi ale poligonului. Fie punctul s de coordonate (x_s, y_s) și poligonul convex $P = (p_1, p_2, \dots, p_N)$ cu N laturi, descris prin lista vârfurilor parcurse consecutiv (coordonatele vârfurilor sunt stocate în tabloul liniar de articole **P** cu câmpurile **x** și **y**). Pentru a simplifica implementarea, în lista de vârfuri ale poligonului este inclus un vârf virtual $p_{N+1} \leftarrow p_1$.

```
function apart : boolean;
var i : integer;
function verific_punct(x1,y1,x2,y2,x3,y3:real):boolean;
begin
  if x1*y2+x2*y3+y1*x3-x3*y2-x2*y1-y3*x1 > 0 then
    verific_punct:=true else verific_punct:=false;
end;
```

Copie autorizata pentru .campion

```
begin {apart}
  apart:=true;
  for i:=1 to N do
    if not
      verific_punct(p[i].x,p[i].y,p[i+1].x,p[i+1].y,s.x,s.y)
    then apart:=false;
end; {apart}
```

Apartenența punctului la un poligon stelat

Fie un poligon arbitrar $P = (p_1, p_2, \dots, p_N)$ cu N laturi. Dacă în P există un punct interior c , astfel încât toate intervalele $[c, p_1], [c, p_2], \dots, [c, p_N]$ aparțin integral P , poligonul se numește *stelat* (fig. 5.7).

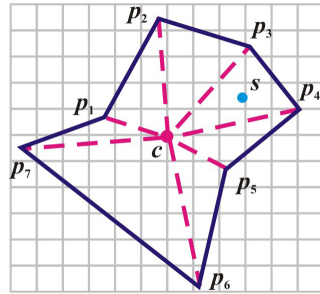


Fig. 5.7. P - poligon stelat

În cazul în care punctul interior c este cunoscut apriori, determinarea apartenenței unui punct arbitrar s la poligonul P se reduce la verificarea existenței unui triunghi⁵ $\triangle cp_i p_{i+1}$, $i = \overline{1, N}$, care să conțină punctul s în calitate de punct interior.

Numărul triunghiurilor este N , numărul de operații necesare pentru verificarea apartenenței punctului la interiorul unui triunghi este mărginit de o constantă. Prin urmare, complexitatea determinării apartenenței punctului la un poligon stelat va fi $O(N)$ în condiția că punctul c este cunoscut.

Apartenența punctului la un poligon simplu

Problema apartenenței unui punct s la un poligon simplu P poate fi rezolvată prin triangularizarea P și prin

⁵ Vârful p_{N+1} este unul virtual și coincide cu p_1 .

verificarea ulterioară a apartenenței s la unul din triunghiurile formate în procesul triangularizării.

Un algoritm mai eficient este bazat pe numărarea intersecțiilor ale dreptei orizontale l care trece prin punctul dat s cu laturile poligonului P (fig. 5.8). Se va cerceta partea dreptei spre stânga de s . Pentru latura curentă se determină:

- poziția punctului față de aceasta;
- intersecția laturii cu semidreapta l .

Dacă numărul intersecțiilor spre stânga de s este impar, punctul se află în interiorul poligonului; pentru un număr par de intersecții, punctul se află în

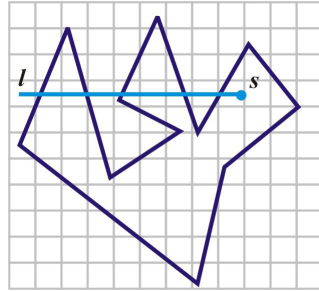


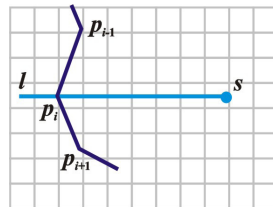
Fig. 5.8. Numărul de intersecții ale semidreptei l cu laturile poligonului determină apartenența punctului s la poligonul P

exterior. Demonstrația este elementară: la parcurgerea spre stânga, prima intersecție marchează intrarea semidreptei în interiorul poligonului, următoarea – ieșirea. Fiecare pereche următoare de intersecții are aceeași semnificație.

Pentru stabilirea intersecției semidreptei cu latura curentă pot fi utilizate metodele descrise în § 2.2.

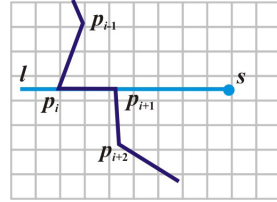
Cazuri speciale

A. Dreapta l trece printr-un vârf p_i al poligonului P la stânga de s . Vârfulurile p_{i-1} și p_{i+1} se află de părți diferite ale dreptei l . Numărul de intersecții se incrementează cu 1.

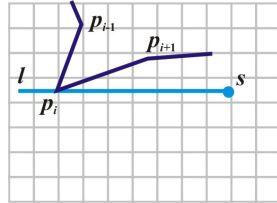


Copie autorizată pentru .campion

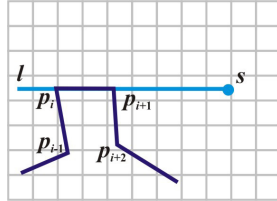
B. Dreapta l conține latura $p_{i-1}p_{i+1}$ a poligonului P la stânga de s . Vârfulurile p_{i-1} și p_{i+2} se află de părți diferite ale dreptei l . Numărul de intersecții se incrementează cu 1.



C. Dreapta l trece printr-un vârf p_i al poligonului P la stânga de s . Vârfulurile p_{i-1} și p_{i+1} se află de aceeași parte a dreptei l . Numărul de intersecții nu se incrementează.



D. Dreapta l conține latura $p_{i-1}p_{i+1}$ a poligonului P la stânga de s . Vârfulurile p_{i-1} și p_{i+2} se află de aceeași parte a dreptei l . Numărul de intersecții nu se incrementează.



Numărul de laturi procesate este N . Determinarea intersecției laturii cu dreapta l este realizată într-un număr constant de operații. Procesarea cazurilor speciale este realizată de asemenea într-un număr de operații mărginit de o constantă. Prin urmare, complexitatea algoritmului este $O(N)$.

5.3. Poligonul și diagrama Voronoi

O altă problemă de apropiere în plan este problema determinării poligonului Voronoi pentru o mulțime de puncte.

Fie în plan mulțimea de puncte $S = \{p_1, \dots, p_N\}$. Fiecare punct p_i , $i = 1, N$, este descris de coordonatele carteziene (x_i, y_i) . Pentru un punct dat $p_i \in S$ se cere să se determine poligonul $V(i)$ care va conține toate punctele planului, mai apropiate de p_i decât de oricare alt punct $p_j \in S$, $p_i \neq p_j$ (fig. 5.9).

Problema generalizată pentru toate punctele mulțimii S este cunoscută sub numele *diagrama Voronoi* (fig. 5.10).

De menționat că pentru unele puncte ale mulțimii S , domeniul determinat de poligonul Voronoi poate fi unul infinit. Se poate demonstra că această proprietate o posedă punctele care formează înfășurătoarea convexă a mulțimii S .

Algoritmul direct pentru determinarea poligonului Voronoi $V(i)$ se bazează pe următoarea observație:

Dreapta l , perpendiculară pe segmentul $p_i p_j$ și care trece prin mijlocul acestuia, conține punctele egal depărtate

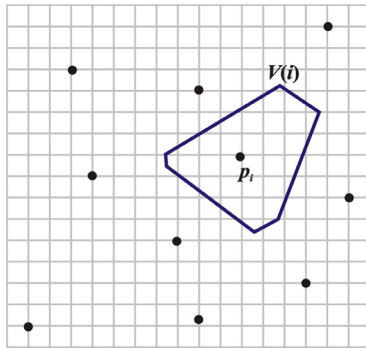


Fig. 5.9. Poligonul Voronoi $V(i)$ pentru punctul p_i .

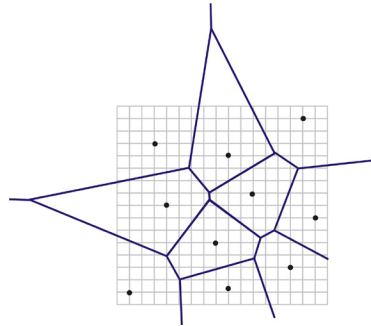


Fig. 5.10 Diagrama Voronoi pentru mulțimea S

Copie autorizata pentru .campion

atât de p_i , cât și de p_j . Punctele mai apropiate de p_i decât de p_j vor forma semiplanul determinat de dreapta l și care conține punctul p_i .

Fiecare punct $p \in S$ este descris prin coordonatele sale carteziene (x, y) . Prin urmare, pentru perechea de puncte p_i, p_j ,

- mijlocul m al segmentului $p_i p_j$ va fi determinat de punctul de coordonate:

$$x_m = \frac{x_i + x_j}{2},$$

$$y_m = \frac{y_i + y_j}{2};$$

- dreapta d care conține segmentul $p_i p_j$ va fi descrisă de ecuația $Ax + By + C = 0$, unde

$$A = y_j - y_i,$$

$$B = x_i - x_j,$$

$$C = x_j y_i - x_i y_j;$$

- orice dreaptă l' perpendiculară pe segmentul $p_i p_j$ va avea ecuația generală de forma $Ax + By + C' = 0$;
- pentru dreapta l , perpendiculară pe segmentul $p_i p_j$ și care trece prin punctul de mijloc al acestuia, valoarea coeficientului C este: $Ay_m - Bx_m$.

Odată ce este cunoscută ecuația dreptei l , perpendiculară pe segmentul $p_i p_j$ în mijlocul acestuia, poate fi determinat semiplanul $R(j)$: $\forall p \in R(j), d(p, p_i) < d(p, p_j)$.

$$\text{Atunci } V(i) = \bigcap_{\substack{j=1, \dots, N \\ j \neq i}} R(j).$$

Pentru realizarea algoritmului poate fi construit un poligon R de „restrângere” a planului, astfel încât $S \in R$. Cel mai simplu poligon R este un dreptunghi având laturile paralele cu axele de coordonate, determinat de punctele

diagonale $(x_{\min}, y_{\min}), (x_{\max}, y_{\max})$:

$$x_{\min} = \min_{i=1, \dots, N} (x_i) - 1, \quad x_{\max} = \max_{i=1, \dots, N} (x_i) + 1,$$

$$y_{\min} = \min_{i=1, \dots, N} (y_i) - 1, \quad y_{\max} = \max_{i=1, \dots, N} (y_i) + 1.$$

Complexitatea algoritmului direct pentru construirea poligonului Voronoi $V(i)$ este $O(N^2)$:

Etapa 1: Determinarea poligonului de restrângere necesită un număr de operații proporțional cu N .

Etapa 2: Determinarea ecuației dreptei l , perpendiculară pe segmentul $p_i p_j$, necesită un număr de operații mărginit de o constantă.

Etapa 3: Determinarea intersecției dreptei l cu poligonul R și restrângerea ulterioară a acestuia necesită un număr de operații proporțional cu numărul de laturi ale poligonului R (nu mai mare decât N).

Etapele 2–3 se repetă pentru toate punctele $p \in S$, $p \neq p_i$. Prin urmare, numărul de operații necesare pentru determinarea poligonului Voronoi va fi proporțional cu N^2 .

Nu este eficient de a aplica direct algoritmul descris anterior pentru determinarea diagramei Voronoi: complexitatea lui crește până la $O(N^3)$.

Pentru realizarea eficientă a algoritmului de determinare a diagramei Voronoi se aplică tehnica divizării consecutive a problemei în mulțimi liniar separabile, până la atingerea cazurilor elementare, și asamblarea ulterioară a soluției [12, pag. 258 - 269].

Fie $S = S_1 \cup S_2$ și au fost construite diagramele Voronoi $DV(S_1), DV(S_2)$ (fig. 5.11). Pentru „asamblarea” soluției vor fi realizate următoarele etape:

Etapa 1: Determinarea vectorilor de intrare (ieșire) ai lanțului de concatenare: se construiesc laturile lipsă (două

Copie autorizata pentru .campion

la număr) pentru înfășurătoarea convexă comună a mulțimii $S_1 \cup S_2$, se determină mijloacele acestor laturi m_1, m_2 . Prin punctele m_1, m_2 se trasează vectorii perpendiculari pe laturile adăugate la înfășurătoarea convexă. Pentru latura superioară vectorul este orientat spre latură, pentru cea inferioară – de la ea (fig. 5.12). Din înfășurătoarea convexă pentru $S_1 \cup S_2$ sunt eliminate lanțurile interioare de laturi ale înfășurătoarelor separate pentru S_1 și S_2 .

Etapa 2: Construirea lanțului de concatenare. Construcția lanțului începe pe vectorul superior, dintr-un punct care precede toate intersecțiile vectorului cu razele diagramelor construite anterior.

Lanțul se trasează de-a lungul vectorului, până la intersecția cu

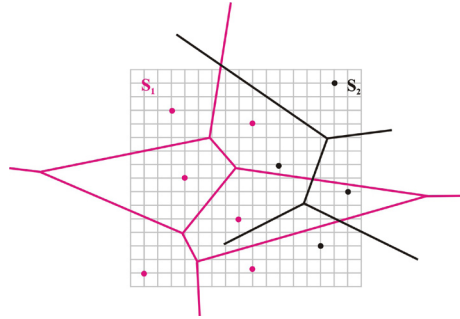


Fig. 5.11. Diagramele Voronoi $DV(S_1)$ și $DV(S_2)$ până la concatenare.

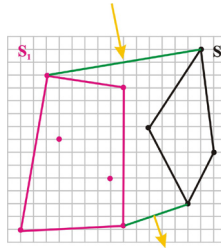


Fig. 5.12 Construirea înfășurătoarei convexe pentru S prin adăugarea laturilor. Trasarea vectorilor de intrare (ieșire) ai lanțului de concatenare

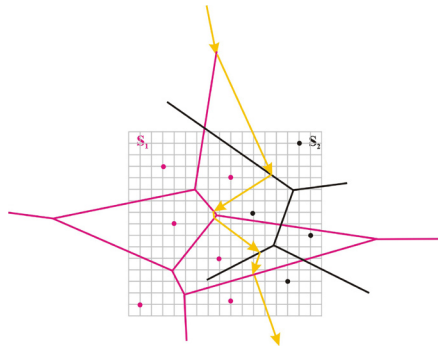


Fig. 5.13. Construirea lanțului de concatenare

una din laturile diagramelor $DV(S_1)$ sau $DV(S_2)$. În punctul de intersecție direcția vectorului se modifică (fig. 5.13).

Modificarea direcției lanțului este determinată de modificarea perechii de puncte între care se trasează acesta. Fie, inițial, lanțul marchează „frontiera” dintre punctele p_i și p_j . Atingerea unei laturi a diagramelor construite anterior semnifică fie înlocuirea p_i prin p_k (dacă latura atinsă separă punctul p_i de p_k), fie înlocuirea p_j prin p_k (dacă latura atinsă separă punctul p_j de p_k). Direcția nouă a lanțului de concatenare este determinată de perpendiculara pe segmentul ce unește perechea nou creată și care trece prin punctul de mijloc al acestuia. Procesul ia sfârșit în momentul atingerii vectorului inferior al lanțului de concatenare.

Etapa 3: Modificarea laturilor diagramei (fig. 5.14). Laturile infinite (semidrepte) intersectate de lanțul de concatenare se transformă în segmente delimitate de originea semidreptei și punctul de intersecție a acesteia cu lanțul de concatenare, construit la etapa 2.

Laturile finite (segmentele) își modifică unul din punctele extreme, acesta fiind înlocuit prin punctul de intersecție cu lanțul de concatenare.

Laturile din $DV(S_1)$, situate integral în dreapta de lanțul de concatenare, și laturile din $DV(S_2)$, situate integral în stânga acestuia, se exclud din diagrama finală pentru $S_1 \cup S_2$.

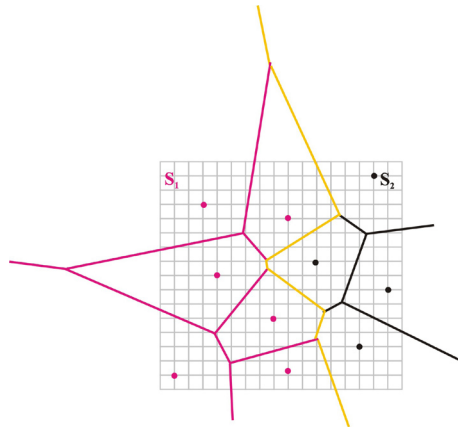


Fig. 5.14. Concatenarea diagramelor

Cazul elementar se obține pentru $2 \leq |S| \leq 3$. Procesarea acestuia este realizată printr-un număr de operații mărginit de o constantă.

Divizarea consecutivă a mulțimii pentru obținerea cazurilor elementare necesită o sortare prealabilă a punctelor din S după abscisa lor. Complexitatea preprocesării este $O(N \log N)$. Numărul de divizări consecutive ale mulțimii curente în două submulțimi „aproape egale”⁶ este proporțional cu $\log(N)$. Pentru concatenarea soluțiilor parțiale, în cazul alegerii structurii de date adecvate⁷, este necesar un număr de operații proporțional cu numărul total de laturi din soluțiile parțiale. Deoarece diagrama Voronoi este o divizare planară a unei mulțimi din N puncte, numărul de laturi va fi proporțional cu N . Prin urmare, complexitatea totală a algoritmului optim este $O(N \log N)$.

5.4. Nuclee

Printre problemele geometrice de calcul se numără și problema determinării *nucleului* poligonului simplu. Nucleul unui poligon simplu P este, în general, format din mulțimea de puncte $Q \subseteq P$, astfel încât:

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

Cu alte cuvinte, nucleul poligonului este mulțimea de puncte ale poligonului, astfel încât oricare dintre

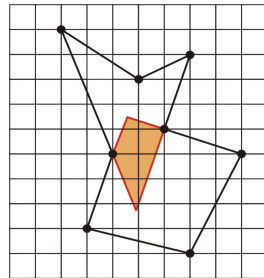


Fig. 5.15. Poligon simplu cu nucleu (partea colorată a poligonului)

⁶ S_1 “este aproape egală” cu S_2 , dacă $\left| |S_1| - |S_2| \right| \leq 1$.

⁷ O structură optimă este lista bidirecțională a laturilor.

ele, fiind unit cu orice alt punct al poligonului, formează un segment ce aparține în întregime poligonului (fig. 5.15).

Nu întotdeauna un poligon simplu are nucleu nevid (fig. 5.16).

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

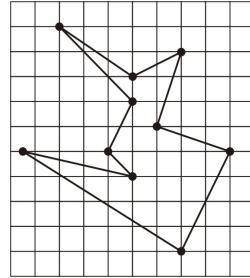


Fig. 5.16. Poligon simplu fără nucleu

Algoritmul pentru determinarea nucleului unui poligon simplu

Date inițiale. Fie un poligon simplu P cu N vârfuri. Se consideră că poligonul este descris prin lista vârfurilor sale $(p_1, p_2, \dots, p_{N-1}, p_N)$, parcurse în direcția mișcării acelor de ceasornic. Fiecare vârf p_i este descris de coordonatele sale (x_i, y_i) .

Algoritmul de determinare a nucleului necesită o construcție secundară: un poligon convex Q care, inițial, conține poligonul P . Pentru determinarea poligonului Q pot fi abordate două metode:

- construirea înfășurătoarei convexe pentru P ;
- construirea unui dreptunghi care conține P .

Metoda a doua este mult mai simplă. Ea se reduce la determinarea, pentru coordonatele vârfurilor P , a valorilor minime și maxime ale abscisei și ale ordonatei. În calitate de Q poate fi considerat 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 condiție obligatorie.

După construirea poligonului Q poate fi determinat nemijlocit nucleul. Metoda (sau cel puțin descrierea ei în limbajul uman) este foarte simplă:

Se analizează consecutiv laturile poligonului P , fiind parcurse în direcția mișcării acelor de ceasornic. Latura curentă $p_i p_{i+1}$ se cercetează ca o dreaptă l . Se determină partea poligonului Q care se află în stânga vectorului $p_i p_{i+1}$ și se exclude din Q . Dacă poligonul Q este situat integral în stânga vectorului $p_i p_{i+1}$, cercetarea ia sfârșit – poligonul P nu are nucleu.

Procedeeul se repetă până nu sunt analizate toate laturile poligonului P .

În final, Q conține nucleul lui P (fig. 5.17).

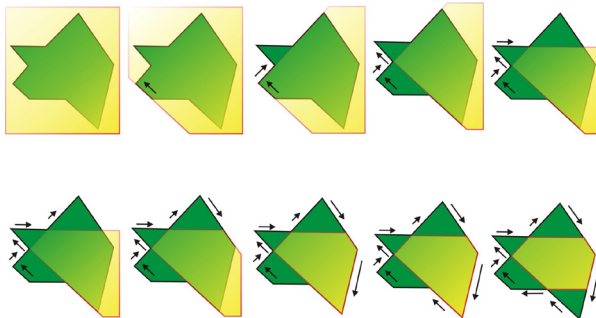


Fig. 5.17. Construirea consecutivă a nucleului poligonului simplu

Pseudocod

Pas 1 Inițializare

La sfârșitul listei de vârfuri (după p_N) se adaugă vârful virtual p_{N+1} . Se construiește poligonul Q (nucleul inițial), conform uneia din metodele descrise anterior. $i \leftarrow 1$.

Pas 2 Procesarea laturii i

Fie $Q = (q_1, q_2, \dots, q_k)$. Pentru dreapta l ce conține latura i a poligonului P definită de vârfurile p_i, p_{i+1} se determină punctele de intersecție cu poligonul Q , precum și laturile intersectate. Fie d_1, d_2 punctele de intersecție a dreptei l cu Q , iar laturile intersectate – $q_j q_{j+1}$ și $q_k q_{k+1}$.⁸

- a) Se formează poligoanele $Q_1 = (d_1, q_{j+1}, \dots, q_k, d_2)$ și $Q_2 = (d_2, q_{k+1}, \dots, q_N, q_1, \dots, q_j, d_1)$.
- b) Se determină $\underline{Q^*} \in \{Q_1, Q_2\}$ care se află în dreapta vectorului $\overrightarrow{p_i p_{i+1}}$.
- c) $Q \leftarrow Q^*$

Pas 3 Repetare

```
if  $i < N$  then begin  $i \uparrow$  goto pas 2 end else
STOP –  $Q$  este nucleul.
```

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

⁸ Dacă dreapta nu intersectează poligonul Q , se verifică doar poziția Q față de $\overrightarrow{p_i p_{i+1}}$. Dacă poligonul e poziționat în stânga, nucleul final este vid, altfel Q nu se modifică la acest pas.

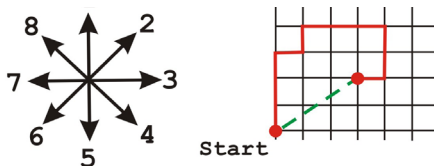
6. Probleme geometrice de concurs

6.1. Robot

Un robot punctiform poate executa instrucțiuni de deplasare în 8 direcții (1 – nord, 2 – nord-est, 3 – est, 4 – sud-est, 5 – sud, 6 – sud-vest, 7 – vest, 8 – nord-vest). Lungimea pasului robotului este 1 pentru direcțiile 1, 3, 5, 7 și $\sqrt{2}$ pentru direcțiile 2, 4, 6, 8. Numărul de pași efectuați în direcția aleasă este un parametru al instrucțiunii de deplasare.

Fiind dat un set de instrucțiuni, să se determine coordonatele punctului final în care se va deplasa robotul.

Astfel, pentru setul
 (1, 3) (3, 1) (1, 1) (3, 3)
 (5, 2) (7, 1) coordonatele
 finale vor fi (3, 2).



Să se scrie un program care, după un set de instrucțiuni, să determine coordonatele punctului final în care se va deplasa robotul. Se consideră că axa Ox e orientată spre est, iar Oy – spre nord. Inițial robotul se află în originea sistemului de coordonate $(0, 0)$.

Date de intrare

Prima linie a fișierului de intrare conține numărul N – numărul de instrucțiuni ($1 \leq N \leq 40$). Următoarele N linii conțin instrucțiunile propriu-zise – numărul direcției (un număr întreg de la 1 la 8) și numărul de pași (un număr întreg de la 1 la 1000), separate prin spațiu.

Date de ieșire

Unica linie a fișierului de ieșire va conține două numere întregi x și y , separate prin spațiu, – coordonatele punctului final în care se deplasează robotul.

Exemple

date.in	date.out
6	3 2
1 3	
3 1	
1 1	
3 3	
5 2	
7 1	
1	
8 10	-10 10

Rezolvare

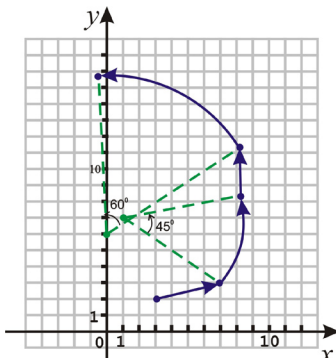
```

program p01;
var   I,N:integer;
        D,L,X,Y:Longint;
begin
  assign(Input,'date.in'); reset(Input);
  read(N);
  X:=0; Y:=0;           {fixarea pozitiei initiale}
  for I:=1 to N do
    begin               {modelarea deplasarii}
      read(D,L);
      case D of
        1: Y:=Y+L;           { deplasare nord cu L}
        2: begin X:=X+L; Y:=Y+L; end; {nord-est cu L}
        3: X:=X+L;           {est cu L}
        4: begin X:=X+L; Y:=Y-L; end; {sud-est cu L}
        5: Y:=Y-L;           {sud cu L}
        6: begin X:=X-L; Y:=Y-L; end; {sud-vest cu L}
        7: X:=X-L;           {vest cu L}
        8: begin X:=X-L; Y:=Y+L; end; {nord-vest cu L}
      end;
    end;
  assign(Output,'date.out'); rewrite(Output);
  write(X,' ',Y);
  close(Input); close(Output);
end.

```

6.2. Robot II

Sistemul de comandă al unui robot care poate executa instrucțiuni de deplasare s-a deteriorat. Robotul continuă să primească instrucțiuni, dar le interpretează nu întotdeauna corect. Fiecare instrucțiune este un triplet de numere (u, v_x, v_y) întregi pozitive. Dacă $u > 90$, atunci robotul se deplasează cu v_x după axa Ox și cu v_y după axa Oy . Dacă $u \leq 90$, robotul se deplasează pe un arc de măsura u al cercului de centru (v_x, v_y) împotriva mișcării acelor de ceasornic. Raza cercului este segmentul care unește robotul cu centrul cercului.



Să se scrie un program care, după coordonatele inițiale ale robotului și un set dat de instrucțiuni, determină punctul final în care va fi poziționat acesta.

Date de intrare

Prima linie a fișierului de intrare conține două numere întregi – coordonatele inițiale ale robotului. Următoarele linii conțin câte o instrucțiune, formată din trei numere întregi, separate prin spațiu.

Date de ieșire

Unica linie a fișierului de ieșire va conține două numere x și y , separate prin spațiu, – coordonatele finale ale robotului, indicate cu cel puțin 3 cifre după virgulă.

Exemplu

date.in	date.out
3 2	-0.653 15.697
130 4 1	
45 1 7	
91 0 3	
60 0 6	

Rezolvare

```

program p02;
const pi=3.141592;
type point=record
    x,y : real;
end;
var P: point;
    alfa,xn,yn:real;

procedure move(var P:point; vx,vy:real);
begin
    P.x:=P.x+vx;
    P.y:=P.y+vy;
end;

procedure rotate (var P:point; u,vx,vy:real);
var old:point;
begin
    old:=P;
    P.x:=vx+(old.x-vx)*cos(u*pi/180)-
        (old.y-vy)*sin(u*pi/180);
    P.y:=vy+(old.x-vx)*sin(u*pi/180)+
        (old.y-vy)*cos(u*pi/180);
end;

begin
    assign(input,'data.in'); reset(input);
    assign(output,'data.out'); rewrite(output);
    readln(P.x,P.y);
    while not eof do
        begin
            readln(alfa,xn,yn);
            if alfa>90 then move(P,xn,yn)
                else rotate(P,alfa,xn,yn);
        end;
    writeln(P.x:10:3,' ',P.y:10:3);
    close(input); close(output);
end.

```

6.3. Piatra

... și unde nu pornește stâncă la vale, săltând tot mai sus de un stat de om; și trece prin gardul și prin tinda Irinucăi, pe la capre, și se duce drept în Bistrița, de clocotea apa!

Ion Creangă. Amintiri din copilărie

Piatra pornită de Ion Creangă se rostogolea în linie dreaptă, dărâmând totul în calea sa. Casa Irinucăi are mulți pereți, unii nimerind în calea pietrei. Fiind date coordonatele a două puncte prin care trece piatra și extremitățile segmentelor care descriu baza pereților casei, să se determine câți pereți vor fi dărâmați de piatra în cădere. Peretele atins într-un punct extrem rămâne întreg.

Să se scrie un program care determină numărul de pereți dărâmați de piatră.

Date de intrare

Fișierul de intrare **date.in** conține N ($N < 1000$) linii a câte patru numere întregi. Prima linie conține descrierea a două puncte prin care trece piatra, în forma x_1, y_1, x_2, y_2 ($|x_i, y_i| < 500$). Următoarele $N-1$ linii conțin descrierea coordonatelor extremităților bazelor pereților, în aceeași consecutivitate, cu aceleași restricții de valori.

Date de ieșire

Unica linie a fișierul de ieșire **date.out** va conține un număr întreg – numărul de pereți dărâmați de piatră.

Exemplu

date.in	date.out
2 1 9 14	4
2 4 1 8	
1 12 3 9	
4 14 6 11	
7 14 9 10	
9 14 7 16	
6 10 8 8	
3 6 6 6	
4 2 4 5	
11 10 12 14	
8 6 10 8	
12 8 14 4	

Rezolvare

```

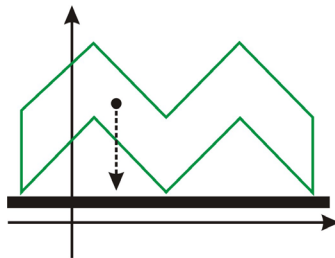
program p03;
type point=record      x,y: real;      end;
      segment=record    e1,e2: point; end;
var g: array[1..1000] of segment;
      l: segment;
      n,i,k: integer;
function sarrus(p1,p2,p3:point): real;
begin      sarrus:=p1.x*p2.y+p2.x*p3.y+p1.y*p3.x
            -p3.x*p2.y-p3.y*p1.x-p1.y*p2.x;
end;

begin
assign(input, 'date.in'); reset(input); n:=0;
readln(l.e1.x,l.e1.y,l.e2.x,l.e2.y);
while not eof do
  begin
    inc(n); readln( g[n].e1.x,
                    g[n].e1.y,g[n].e2.x,g[n].e2.y);
  end;
  k:=0;
  for i:=1 to n do
    if
      sarrus(l.e1,l.e2,g[i].e1)*sarrus(l.e1,l.e2,g[i].e2) < 0
    then inc(k);
    assign(output, 'date.out'); rewrite(output);
    writeln(k); close(input); close(output);
  end.

```

6.4. Carcasa

Fie o carcasa având forma unui poligon simplu. Pentru a fi poziționată vertical pe o suprafață plană, carcasa trebuie să aibă centrul de masă situat strict între 2 puncte de contact cu suprafața. Centrul de masă este întotdeauna un punct interior al carcasei și nu coincide cu nici un vârf al ei.



Să se scrie un program care va determina numărul pozițiilor în care poate fi stabilizată vertical carcasa.

Date de intrare

Prima linie a fișierului de intrare **date.in** conține trei numere întregi, separate prin spațiu: N – numărul de vârfuri ale carcasei – și x_c, y_c – coordonatele centrului de masă.

Urmează N linii ce conțin câte două numere întregi x_i, y_i ($-1000 \leq x_i, y_i \leq 1000$), separate prin spațiu, – coordonatele vârfurilor poligonului în ordinea parcurgerii lor.

Date de ieșire

Fișierul de ieșire **date.out** va conține un singur număr întreg – numărul de poziții în care poate fi stabilizată vertical carcasa.

Restricții

$$3 \leq N \leq 1000$$

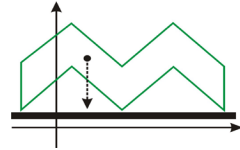
$$-1000 \leq x_c, y_c \leq 1000$$

Exemplu

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

Rezolvare

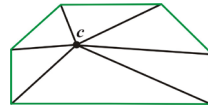
Deoarece carcasa se sprijină pe careva puncte extreme ale poligonului, problema poate fi divizată în două subprobleme relativ simple:



- 1) determinarea înfășurătoarei convexe a vârfurilor poligonului;
- 2) verificarea dacă un triunghi are un unghi obtuz.

Prima subproblemă este rezolvată cu ajutorul algoritmului descris anterior (§ 3.2).

După determinarea înfășurătoarei convexe, problema poate fi reformulată în felul următor:



Fie un poligon convex P și un punct interior c , unit prin linii drepte cu vârfurile poligonului. În câte dintre triunghiurile formate înălțimea construită din punctul c se proiectează într-un punct interior al laturii poligonului care aparține triunghiului?

Evident, înălțimea construită din c se proiectează pe latură dacă niciunul dintre unghiurile alăturate laturii poligonului nu este obtuz. Verificarea unghiurilor se realizează elementar cu ajutorul teoremei cosinusurilor. Complexitatea etapei este liniară după numărul de vârfuri.

6.5. Turnuri

Bitlanda este o țară care se extinde permanent, liniar. Frontiera țării reprezintă o linie frântă închisă, fără autointersecții. 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 Bitlandei, Bytezar, a decis sa trimită la vatră garnizoanele turnurilor care nu se mai află la hotarele țării.

Să se scrie un program care va determina câte turnuri vor fi lipsite de garnizoane.

Date de intrare

Prima linie a fișierului de intrare conține un număr întreg: N ($1 \leq N \leq 10000$) – numărul de turnuri de veghe în Bitlanda.

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

Date de ieșire

Fișierul de ieșire va conține un număr întreg – numărul de turnuri care pot fi lipsite de garnizoane.

Exemplu

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

6.6. Atac

Agencia Spațială a planetei Bitterra (ASB) a recepționat un roi meteoritic, care se apropie de planetă. Fiecare stat de pe Bitterra a fost anunțat despre pericol și a primit lista punctelor de cădere a meteoriților, calculate de ASB. Serviciile pentru situații excepționale ale fiecărui stat urmează să determine câți meteoriți vor cădea pe teritoriul țării. Frontiera oricărui stat de pe Bitterra este un poligon convex; meteoriții sunt punctiformi. Punctele de pe frontieră nu se consideră aparținând statului. Frontiera poate conține mai multe vârfuri consecutive coliniare.

Să se scrie un program care va determina numărul de meteoriți ce cad pe teritoriul unui stat dat.

Date de intrare

Prima linie a fișierului de intrare **atak.in** conține numărul n – numărul de vârfuri ale poligonului care descrie frontiera unui stat. Următoarele n linii conțin descrierea consecutivă a vârfurilor frontierei: fiecare linie va conține câte o pereche de numere separate prin spațiu – coordonatele unui vârf. Urmează o linie care conține un număr întreg m – numărul de meteoriți, apoi m linii, care conțin câte două numere, separate prin spațiu, – coordonatele punctelor de cădere a meteoriților.

Date de ieșire

Fișierul de ieșire **atak.out** va conține un singur număr întreg – cel al meteoriților care cad pe teritoriul statului dat.

Restricții

1. Coordonatele vârfurilor poligonului și ale punctelor de cădere a meteoriților sunt numere întregi din intervalul $[-1000000, 1000000]$.
2. $3 \leq n \leq 20000$.
3. $2 \leq m \leq 20000$.

Exemplu

atac.in	atac.out	Explicație
4	2	
2 4		
8 4		
6 8		
4 6		
4		
3 5		
4 7		
5 5		
6 7		

Rezolvare

Formularea abstractă a problemei este următoarea:

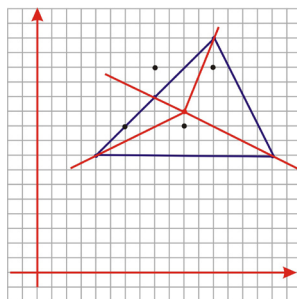
Fie în plan un poligon convex P cu n vârfuri. În același plan este dată o mulțime M din m puncte. Se cere să se determine câte puncte din M aparțin interiorului P .

Rezolvarea se divide în câteva etape.

1. Se determină un punct interior al poligonului de coordonate (x_{cm}, y_{cm}) (de exemplu, centrul de masă).

Fiind date coordonatele (x_i, y_i) , $i = \overline{1, n}$, ale vârfurilor poligonului P , coordonatele centrului de masă pot fi calculate după formula:

$$x_{cm} = \frac{\sum_{i=1}^n x_i}{n}, \quad y_{cm} = \frac{\sum_{i=1}^n y_i}{n}.$$



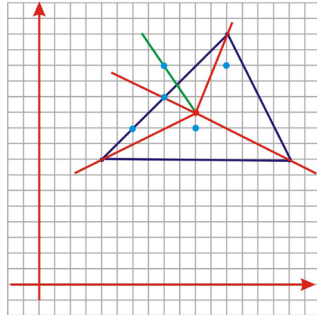
2. Fiind dat un punct interior (x_{cm}, y_{cm}) al poligonului, se calculează unghiurile pe care le formează cu axa Ox semidreptele duse din acesta prin vârfurile (c_x, c_y) .

```

function angle(cx,cy:real): real;
begin
  sinus:=(cy -ycm) / sqrt (sqr(cy-ycm)+sqr(cx-xcm));
  cosinus:= (cx -xcm) / sqrt (sqr(cy-ycm)+sqr(cx-xcm));
  if cosinus=0 then
    if sinus>0 then angle:= pi/2
      else if sinus<0 then angle:=3*pi/2
    else begin
      if (sinus>0) and (cosinus>0)
        then angle:=arctan(sinus/cosinus);
      if (sinus>0) and (cosinus<0)
        then angle:=pi-arctan(abs(sinus/cosinus));
      if (sinus<=0) and (cosinus<0)
        then angle:=pi+arctan(abs(sinus/cosinus));
      if (sinus<=0) and (cosinus>0)
        then angle:=2*pi-arctan(abs(sinus/cosinus));
      end;
    end;
end;

```

3. Pentru un punct dat al mulțimii M se calculează unghiul format de semidreapta determinată de acesta și centrul de masă al poligonului cu axa Ox . Se folosește aceeași funcție de la etapa 2.



```

alfa:=angle(b[i].x,b[i].y)

```

4. Se determină latura poligonului intersectată de semidreapta determinată la pasul 3. Pentru optimizarea acestui pas se folosește căutarea binară.

```

k:=1; {determinarea vârfului cu unghi maxim}
while (a[k].u<=a[k+1].u) and (k<n) do k:=k+1;

{divizarea binara}
if (alfa < a[k+1].u) or (alfa>a[k].u)

```

```

Copie autorizata pentru .campion
  then begin st:=k; dr:=k+1; end
  else begin
    if (alfa < a[1].u)
      then begin st:=k+1; dr:=n+1; end
      else begin st:=1; dr:=k; end;
    repeat
      mj:=(st + dr) div 2;
      if a[mj].u >alfa then dr:=mj else st:=mj
    until dr=st+1;
  end;

```

5. Se determină poziția centrului de masă și a punctului curent al mulțimii M față de latura determinată. Dacă sunt de aceeași parte, punctul curent este în interior. Pentru determinarea poziției se

folosește semnul determinantului
$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix},$$

unde (x_1, y_1) , (x_2, y_2) sunt coordonatele vârfurilor care determină latura, iar (x_3, y_3) – coordonatele punctului cercetat (respectiv coordonatele centrului de masă).

```

q:=semn(a[st].x,a[st].y,a[dr].x,a[dr].y,b[i].x,b[i].y);
p:=semn(a[st].x,a[st].y,a[dr].x,a[dr].y,xcm,ycm);
if p*q>0 then inc(s);

```

6.7. Evadare

Un grup de pinguini a decis să evadeze din grădina zoologică. În acest scop ei au săpat un tunel liniar. Din nefericire, zidul grădinii zoologice formează un poligon simplu cu N ($3 \leq N \leq 10000$) laturi, astfel încât, ieșind din tunel, pinguinii nu pot afirma cu certitudine dacă se află în interiorul grădinii zoologice sau în afara ei.

Să se scrie un program care, după coordonatele punctelor extreme ale tunelului și coordonatele vârfurilor poligonului ce stabilesc perimetrul grădinii zoologice, va determina dacă evadarea s-a soldat cu succes sau nu.

Date de intrare

Prima linie a fișierului de intrare **evadare.in** conține patru numere întregi, separate prin spațiu, – coordonatele punctelor extreme ale tunelului. Următoarele linii conțin descrierea consecutivă a vârfurilor zidului: fiecare linie va conține câte o pereche de numere, separate prin spațiu, – coordonatele unui vârf.

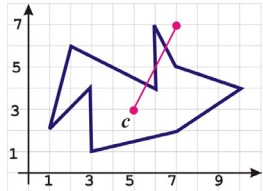
Date de ieșire

Fișierul de ieșire **evadare.out** va conține un singur cuvânt: DA – în cazul în care punctul final al tunelului este în afara grădinii zoologice; NU – în caz contrar.

Restricții

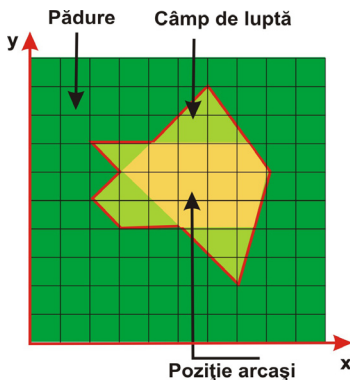
Coordonatele vârfurilor poligonului și ale punctelor extreme ale tunelului sunt numere întregi din intervalul $[-1000, 1000]$.

Exemplu

evadare.in	evadare.out	explicație
5 3 7 7 1 2 2 6 6 4 6 7 7 5 10 4 7 2 3 1	DA	

6.8. Arcași

Secretul victoriilor faimosului comandant de oști Mega-Flop este strategia lui de alegere a poziției arcașilor pe câmpul de luptă. Câmpul are forma unui poligon simplu și e înconjurat de păduri. Mega-Flop plasează arcașii doar pe poziții din care este **văzut** tot câmpul de luptă. Se consideră că arcașii 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 arcaș are nevoie de o unitate de suprafață. Astfel, numărul maxim de arcași care pot fi plasați pe poziții este determinat de aria poligonului din care este văzută toată câmpia.



Să se scrie un program care determină numărul maxim de arcași care pot fi plasați pe poziții de tragere în câmpul de luptă.

Date de intrare

Fișierul 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 mișcării 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 ieșire

Fișierul de ieșire **arcas.out** va conține un singur număr întreg – numărul maxim de arcași care pot fi plasați pe poziții.

Restricții

$$3 \leq N \leq 1000$$

$$0 < x_i, y_i \leq 10000$$

Exemplu

arcas.in	arcas.out
9	11
2 5	
3 6	
2 7	
4 7	
6 9	
8 6	
7 2	
5 4	
3 4	

Rezolvare

```
program p68;
type
  lat=record x,y:real; end;
  pol=array[0..1001] of lat;
```

Copie autorizata pentru .campion

```
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;

    {... si initializare poligon}
    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;
end;

function intersect
    (al,bl,cl,ad,bd,cd:real;i,j:integer): boolean;

{determină intersecția dreptei și laturii
 + coordonate punctului de intersecție}
begin

{1. dreapta și latura sunt paralele}
    if Ad*B1=Bd*A1 then begin intersect:=false; exit; end;

{2. dreapta intersectează 2 laturi adiacente în punct
extrem}
    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;
```



```

{3. Dreapta și latura nu sunt paralele}
if (Ad*B1<>Bd*Al) then
  if Al<>0 then begin
    yint:=(Ad*C1-Cd*Al)/(Bd*Al-Ad*B1);
    xint:=(-B1*yint-C1)/Al;
  end
  else
    begin yint:=-C1/B1;
      xint:=(-Bd*yint-Cd)/Ad;
    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);

{ Proceșează latura j a poligonului P}
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; B1:=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

```

```

Copie autorizata pentru .campion
      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
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):real;

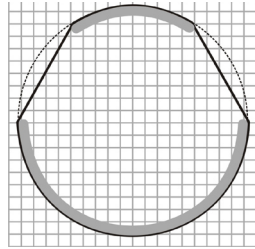
{aria poligonului simplu}
var s:real;
    i:integer;
begin
  s:=0;
  for i:=1 to n do
    s:=s+(a[i+1].x-a[i].x)*(a[i+1].y+a[i].y)/2;
  area:=s;
end;

{programul principal}

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(input);
  close(output);
end.
```

6.9. Cetate

Arheologii din Bitlanda în timpul săpăturilor au descoperit niște pietre așezate oarecum straniu. Ei au ajuns la concluzia ca acestea formează fragmente ale unui zid circular, care înconjură o cetate veche.



Pentru a proteja de turiști și de curioși fragmentele descoperite, arheologii au decis să le înconjoare cu un gard din plasă metalică. Deoarece este destul de complicat și incomod de a înconjura fiecare fragment, s-a luat decizia de a împrejmui toate fragmentele împreună.

Să se scrie un program care să determine lungimea minimă a plasei necesare pentru a îngrădi fragmentele zidului.

Date de intrare

Prima linie a fișierului de intrare conține două numere: n ($1 \leq n \leq 180$) – numărul fragmentelor de zid; r ($1 \leq r \leq 100$) – raza zidului. Urmează n perechi de numere întregi, care descriu fragmentele de zid: a_i, b_i – măsura unghiurilor (în grade) care descriu începutul și sfârșitul fragmentului. Unghiurile se măsoară începând cu direcția nord de la centrul cetății, în sens opus mișcării acelor de ceasornic ($0 \leq a_i; b_i < 360, a_i \neq b_i$). Fiecare fragment este descris la fel împotriva direcției de mișcare a acelor de ceasornic. Fragmentele de zid nu au puncte comune.

Date de ieșire

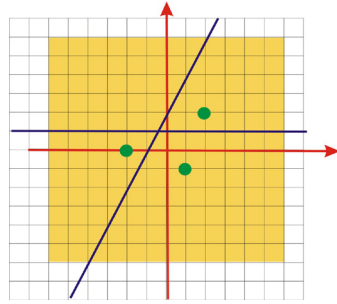
Fișierul de ieșire va conține lungimea minimă a plasei necesare, cu trei cifre după virgulă.

Exemplu

cetate.in	cetate.out
2 10	61.888
330 30	
90 270	

6.10. Druizi

Ruinele din Stonehenge se află într-un loc special, unde se intersectează M linii energetice ale Pământului. Liniile energetice împart câmpia Stonehenge în sectoare. În fiecare an, în cea mai scurtă zi a anului, N druzi se adună în câmpie pentru un ritual, ce asigură o roadă bogată



pentru anul următor. Pentru succesul ritualului este necesar ca în fiecare sector al câmpiei, delimitat de liniile energetice, să se afle nu mai mult decât un druid. Câmpia are forma unui pătrat cu centrul în originea sistemului de coordonate și care are lungimea laturii $2L$. Liniile energetice sunt linii drepte. Druzii sunt punctiformi și nu se pot afla pe liniile energetice.

Să se scrie un program care determină dacă există sectoare ale câmpiei în care se află 2 sau mai mulți druzi.

Date de intrare

Fișierul de intrare conține câteva (cel mult 5) seturi de date, separate prin câte o linie ce conține semnul #.

Prima linie a fiecărui set de date conține numerele N , M și L ($1 \leq N \leq 1000$, $0 \leq M \leq 1000$, $1 \leq L \leq 2000$).

Urmează N linii ce conțin câte două numere întregi x_i , y_i – coordonatele druzilor. Toți druzii se află în interiorul câmpiei, oricare doi druzi nu coincid.

Următoarele M linii ale setului de date conțin câte un triplet de numere întregi a_i , b_i , c_i . Numerele corespund coeficienților ecuației $A_i x + B_i y + C_i = 0$, care descrie linia energetică i . Niciun druid nu se află pe o careva linie. Oricare două linii nu coincid. Valorile a_i , b_i , c_i nu depășesc 10000 după modul.

Copie autorizata pentru .campion

Date de ieşire

Fişierul de ieşire va conţine pentru fiecare set de date cuvântul YES, dacă există cel puţin un sector în care se află doi sau mai mulţi druizi, NO, în caz contrar. Fiecare răspuns se va scrie într-o linie aparte, în ordinea apariţiei seturilor de date în fişierul de intrare.

Exemplu

druizi.in	druizi.out
3 2 3	YES
2 2	NO
1 -1	
-2 0	
1 1 -1	
0 1 -1	
#	
1 0 100	
0 0	

Rezolvare

Pentru implementare vor fi definite tipurile şi structurile:

```
type pt=record x,y,a:real; T:char;
end;
dr=array[1..3003] of pt;
var
d:dr;
n,m,i:integer;
int,xint,r,a,b,c,ae,be,ce,de,
l1a,l2a,l1b,l2b,l1c,l2c: real;
```

Pentru citirea şi preprocesarea datelor va fi utilizată procedura **readdata**:

```
procedure readdata;
begin
readln(n,m,r);
for i:=1 to n do
begin readln(d[i].x,d[i].y); d[i].t:='L'; end;
```

```

for i:=1 to m do
  begin
    readln(a,b,c);
    if i=1 then begin l1a:=a; l1b:=b; l1c:=c; end;
    if i=2 then begin l2a:=a; l2b:=b; l2c:=c; end;
    ae:=b*b+a*a;
    be:=2*b*c;
    ce:=c*c-a*a*2*r*r;
    de:=be*be-4*ae*ce;
    if a<>0 then begin
      inc(n);
      d[n].y:=(-be-sqrt(de))/2/ae;
      d[n].x:=(-b*d[n].y-c)/a;
      d[n].t:='D';
      inc(n);
      d[n].y:=(-be+sqrt(de))/2/ae;
      d[n].x:=(-b*d[n].y-c)/a;
      d[n].t:='D';
    end
    else begin
      inc(n);
      d[n].y:=-c/b;
      d[n].x:=sqrt(2*r*r-c*c/b/b);
      d[n].t:='D';
      inc(n);
      d[n].y:=d[n-1].y;
      d[n].x:=-d[n-1].x;
      d[n].t:='D';
    end;
  end;
end; {readdata}

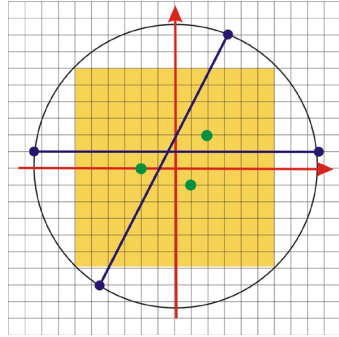
```

În rezolvare vor fi parcurse următoarele etape:

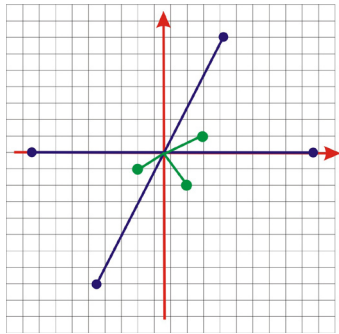
1. Toate liniile se intersectează într-un punct, care trebuie determinat. Dacă numărul de linii este 0 sau 1, sunt câteva cazuri elementare, care se cercetează aparte. Dacă numărul de linii este mai mare sau egal cu doi, atunci se folosesc oricare două linii pentru a determina coordonatele punctului de intersecție.

Copie autorizata pentru .campion

2. Druizii se află în interiorul pătratului cu latura $2L$ și centul în originea sistemului de coordonate. Ei se află și în interiorul cercului de rază $\sqrt{2}L$ și centrul în originea sistemului de coordonate. Pentru fiecare din liniile energetice se determină punctele ei de intersecție cu cercul de rază $\sqrt{2}L$ și centrul în originea sistemului de coordonate. Aceste puncte se marchează aparținând dreptelor (D) de separare a sectoarelor.



3. Originea sistemului de coordonate se deplasează în punctul de intersecție a dreptelor (liniilor energetice). Apoi se trece la coordonatele polare pentru punctele ce reprezintă druizii și intersecțiile liniilor energetice cu cercul (realizare – procedura `movecenter`).



```
procedura movecenter;  
begin
```

```
{se determină punctul de intersecție al dreptelor}  
if l1a<>0 then  
    begin yint:=(l1c*l2a-l1a*l2c)/(l2b*l1a-l2a*l1b);  
          xint:=(-l1b*yint-l1c)/l1a;  
    end  
else begin yint:=-l1c/l1b;  
          xint:=(-l2b*yint-l2c)/l2a;  
    end;
```



```

for i:=1 to n do
  begin {se transferă originea..}
    d[i].x:=d[i].x - xint; d[i].y:=d[i].y - yint;

{ ... si se determină coordonatele polare - unghiul!}

    if d[i].x<>0 then
      begin
        d[i].a:=180/pi*arctan(abs(d[i].y/d[i].x));
        if (d[i].x<0) and (d[i].y>0) then d[i].a:=180-d[i].a;
        if (d[i].x<0) and (d[i].y<=0) then d[i].a:=d[i].a+180;
        if (d[i].x>0) and (d[i].y<0) then d[i].a:=360-d[i].a;
        end
      else begin if d[i].y> 0 then d[i].a:=90;
                 if d[i].y<0 then d[i].a:=270;
        end;
    end
end; {movecenter}

```

4. Se sortează sistemul de puncte p_i după creșterea măsurii unghiului format de fiecare vector $\overline{Op_i}$ cu axa Ox .
5. Dacă după sortare există cel puțin două puncte vecine, reprezentând druzi, rezultă existența unui sector al câmpiei cu aceeași proprietate.

```

procedure solve;
  var vec: boolean;
begin
  {cazurile elementare...}

  if ((m=0) and (n=1)) or ((m=1) and (n=3))
    then writeln('NO');
  if ((m=0) and (n>1)) or ((m=1) and (n>4))
    then writeln('YES');
  if (m=1) and (n=4) then
    begin

```

Copie autorizata pentru .campion

```
{se determină pozitia druizilor fata de dreapta}

if sarrus(d[3],d[4],d[1])*sarrus(d[3],d[4],d[2]) < 0
    then writeln('NO') else writeln('YES');
end;
if m>=2 then
    begin
    {... si cazul general}
    movecenter;
    qsort(d,1,n);
    vec:=FALSE;
    for i:=1 to n-1 do
        if (d[i].t='L') and (d[i+1].t='L')
            then vec:=TRUE;
        if vec then writeln('YES') else writeln('NO');
    end;
end; {solve}
```

Notatii

$\overrightarrow{p_1 p_2}$	– segment orientat (vector) cu originea în punctul p_1
$S = \{p_1, \dots, p_N\}$	– mulțimea S , cu elementele p_1, \dots, p_N
$ S $	– cardinalul mulțimii S
$i \uparrow$	– incrementarea valorii i cu 1
$k \leftarrow m$	– variabilei k i se atribuie valoarea variabilei m
$[a, b]$	– interval cu extremitățile a, b
$\triangle abc$	– triunghi cu vârfurile a, b, c
$DV(S)$	– diagrama Voronoi pentru mulțimea S
$T(S)$	– triangularizarea mulțimii S
$Q(S)$	– înfășurătoarea convexă a mulțimii S
$A \Rightarrow B$	– din A rezultă B
$A \Leftrightarrow B$	– A este echivalent cu B
$x \in X, (x \notin X)$	– x aparține (nu aparține) mulțimii X
$\{x \in X : Q\}$	– submulțimea elementelor x din X , care satisfac condiția Q
$A \subseteq B$	– A se conține în B (A este submulțime a mulțimii B)
<code>qsort(A, n)</code>	– apel al procedurii pentru sortarea structurii de date A din n elemente

Bibliografie

1. Ammeraal, Leen. *Programming Principles in Computer Graphics*. John Wiley and Sons, 1986.
2. Ammeraal, Leen. *Computer Graphics for the IBM PC*. John Wiley and Sons, 1986.
3. Cerchez, Emanuela. *Programarea în limbajul C/C++ pentru liceu. Vol III*. Iași: Polirom, 2007.
4. Cormen, Thomas. *Introducere în algoritmi*. Cluj: Agora.
5. Foley, James; Andries, Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley Publishing, 1984.
6. Giumale, Cristian. *Introducere în analiza algoritmilor*. Iași: Polirom, 2004.
7. Sedgewick, Th. *Algorithms in C*. Addison Wesley, 2001.
8. Williamson, Gill. *Combinatorics for Computer Science*. New York: Dover publications, 2002.
9. Ласло, М. *Вычислительная геометрия и компьютерная графика на C++*. Москва: Бином, 1997.
10. Нагао, М. *Структуры и базы данных*. Москва: Мир, 1986.
11. Новиков, Ф. А. *Дискретная математика для программистов*. Санкт Петербург: Питер, 2001.
12. Пападимитриу, Х. *Комбинаторная оптимизация. Алгоритмы и сложность*. Москва: Мир, 1985.
13. Препарата, Ф.; Шеймос, М. *Вычислительная геометрия. Введение*. Москва: Мир, 1989.