Outline

Cuprins

1 Introducere

1.1 Motivaţie

Ce este geometria computaţională

Obiectele geometrice - punctele, liniile, poligoanele, etc. - constituie baza multor aplicaţii şi conduc la multe probleme şi algoritmi interesanţi.

Disciplina a fost numită în jurul anului 1975, când teza de doctorat a lui Shamos a atras atenţia multor cercetători.

În centrul acestei discipline stă o serie de tehnici de proiectare şi de analiză a algoritmilor. Aceşti algoritmi operează cu sau sunt ghidaţi de o serie de structuri de date caracteristice geometriei computaţionale. Acestea includ aranjamente de obiecte geometrice, localizări, înfăşuratoarea convexă, diagrame Voronoi, triangularizări.

Scopul următoarelor trei lecţii este de a face o scurtă introducere în algoritmica din acest domeniu.

Putem vedea aceste lecţii ca un studiu de caz de algoritmică specifică unui domeniu (Domain Specific Algorithms).

Vom considera numai obiecte din geometria plană.

Aplicaţii

- grafică ("computer vision", reconstruirea de imagini)
- robotică (mişcare în plan, vizibilitate)
- proiectare asistată de calculator (CAD)
- siteme informatice geografice (GIS)
- statistică

Exemplu: înfăşuratoarea convexă

Dată o mulţime finită S de puncte în plan, să se determine cea mai mică mulţime convexă care include S.

Exemplu: intersecţia de poligoane

Date două suprafeţe poligonale, să se calculeze intersecţia lor.

Exemplu: problema galeriei de arta

Podeaua unei galerii de arta este sub forma unui poligon. Se pune problema determinării numărului de paznici care să supravegheze complet întreaga galerie. Unghiul de vedere al unui paznic este de 360 de grade dar el nu poate vedea prin ziduri.

Formulare echivalentă: câte becuri sunt necesare pentru luminarea galeriei (se presupune că tavanul are aceeaşi formă ca podeaua şi pereţii nu reflectă lumina).

Variante:
– număr minim de paznici,
– numărul de paznici necesari pentru orice polgon cu n vârfuri, pentru un n dat

Vizual - simplu, Algoritmic - mai complicat

Instance: Se consideră un triunghi T şi un punct P.
Question: Este P interior lui T?
Aceeaşi problemă dar cu un poligon convex PLG în loc de triunghiul T.

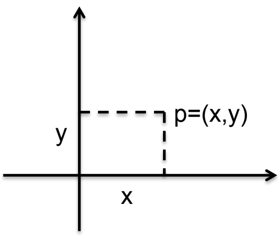
- dacă desenăm triunghiul (poligonul) şi punctul, răspunsul este identificat vizual imediat;

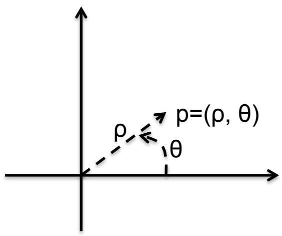
- algoritmic nu e aşa de simplu:
  – structuri de date pentru T (PLG) şi P
  – formularea matematică a răspunsului
  – translatarea formulării matematice în limbaj algoritmic, utilizând termenii structurilor de date

1.2 Cunoaşterea domeniului problemei

Puncte

Un punct P poate fi reprezentat prin coordonate carteziene, $P = (x, y)$, sau coordonate polare, $P = (\rho, \theta)$.





Tipuri de date pentru puncte

- structuri

Coordonate carteziene:

```
P |-> {x |-> 4, y |-> 3}
```

$\mathtt{P} \mapsto v_P$, $v_P \in \mathit{CPoint}$

$$\mathit{CPoint} = \mathit{Str}\langle \mathtt{x} : \mathit{Float}, \mathtt{y} : \mathit{Float} \rangle = \{\{\mathtt{x} \to v_x \ \mathtt{y} \to v_y\} \mid v_x, v_y \in \mathit{Float}\}$$

`CPoint` este descrierea abstractă a tiupului C++:
`struct CPoint { float x, y ;}`

Coordonate polare:

`P |-> {rho |-> 5, theta |-> 0.643501109}` $\mathtt{P} \mapsto v_P, v_P \in \mathit{PPoint}$ $\mathit{PPoint} = \mathit{Str}\langle \mathtt{rho} : \mathit{Float}, \mathtt{theta} : \mathit{Float}\rangle = \{\{\mathtt{rho} \to \rho\ \mathtt{theta} \to \theta\} \mid \rho, \theta \in \mathit{Float}\}\}$

Conversie polare $\mapsto$ carteziene

$(\rho, \theta) \mapsto (\rho \cdot \cos(\theta), \rho \cdot \sin(\theta))$ (timp uniform: $O(1)$)

```
cart(PP) {
  CP.x = PP.rho * cos(PP.theta);
  CP.y = PP.rho * sin(PP.theta);
  return CP;
}

pi = 3.14159265359;

PP1.rho = sqrt(2);
PP1.theta = pi / 4;
CP1 = cart(PP1);
krun ../tests/comp-geom1/polar2cart.alk -cINIT=".Map"
<k>
   .K
</k>
<state>
   CP1 |-> { (x -> 9.9999999999994837e-01) (y -> 1.0000000000000517e+00) }
   PP1 |-> { (rho -> 1.4142135623730951e+00)
             (theta -> 7.8539816339750002e-01) }
   pi |-> 3.1415926535900001e+00
</state>
```

Conversie carteziene $\mapsto$ polare 1/6

$\rho = \sqrt{x^2 + y^2}$

- $x = 0$

  $$\theta' = \begin{cases} \dfrac{\pi}{2} & , y > 0 \\ -\dfrac{\pi}{2} & , y < 0 \\ \text{nedefinit} & , y = 0 \end{cases}$$

- $x \neq 0$

  $\theta' = \mathrm{atan}(\dfrac{y}{x})$

$$\theta = \begin{cases} \theta' & , \theta' \geq 0 \\ \theta' + 2\pi & , \theta' < 0 \end{cases}$$

Convenim $\theta' = 0$ dacă $x = y = 0$.

Conversie carteziene $\mapsto$ polare 2/6

```
polar(CP) {
  PP.rho = sqrt(CP.x * CP.x + CP.y * CP.y);
  if (CP.x == 0.0) {
```

```
    if (CP.y < 0.0) theta1 =  0.0 - pi/2.0;
    else theta1 =  pi/2.0;
  }
  else {
    theta1 = atan(CP.y / CP.x);
  }
  if (theta1 >= 0) PP.theta = theta1;
  else PP.theta = theta1 + 2 * pi;
  return PP;
}
```

Conversie carteziene ↦ polare 3/6

```
CP1  = { x -> 1.0 y -> 1.0};
PP1 = polar(CP1);
CP1 = cart(PP1);

krun ../tests/comp-geom1/cart2polar.alk -cINIT=".Map"

<k>
    .K
</k>
<state>
    CP1 |-> { (x -> 1.0000000000000002e+00) (y -> 1e+00) }
    PP1 |-> { (rho -> 1.4142135623730951e+00)
    (theta -> 7.8539816339744828e-01) }
</state>
```

Intermezzo: atenţie la erorile de calcul!!

```
CP1  = { x -> 1.0 y -> 1.0};
PP1 = polar(CP1);
CP11 = cart(PP1);

if (CP1.x == CP11.x) b = true;
else b = false;

krun ../tests/comp-geom1/cart2polar.alk -cINIT=".Map"
<k>
    .K
</k>
<state>
    ...
    CP1  |-> { (x -> 1e+00) (y -> 1e+00) }
    CP11 |-> { (x -> 1.0000000000000002e+00) (y -> 1e+00) }
    b    |-> false
</state>
```

Conversie carteziene ↦ polare 4/6

```
CP2  = { x -> -1.0 y -> 1.0};
PP2 = polar(CP2);
CP2 = cart(PP2);
```

```
krun ../tests/comp-geom1/cart2polar.alk -cINIT=".Map"

<k>
    .K
</k>
<state>
    CP2 |-> { (x -> 1.0000000000004137e+00)
    (y -> -9.9999999999958644e-01) }
    PP2 |-> { (rho -> 1.4142135623730951e+00)
    (theta -> 5.4977871437825518e+00) }
    pi |-> 3.1415926535900001e+00
</state>
```

Conversie carteziene $\mapsto$ polare 5/6

$\text{atan} : \mathbb{R} \to (-\frac{\pi}{2}, \frac{\pi}{2})$

Trebuie determinat cadranul la care aparţine punctul:

$$\theta' = \begin{cases} \text{atan}(\frac{y}{x}) & , x > 0 \\ \text{atan}(\frac{y}{x}) + \pi & , x < 0 \wedge y \geq 0 \\ \text{atan}(\frac{y}{x}) - \pi & , x > 0 \wedge y < 0 \end{cases}$$

$$\theta = \begin{cases} \theta' & , \theta' \geq 0 \\ \theta' + 2\pi & , \theta' < 0 \end{cases}$$

În multe limbaje de programare θ' este notată cu `atan2()`.

Conversie carteziene $\mapsto$ polare 6/6

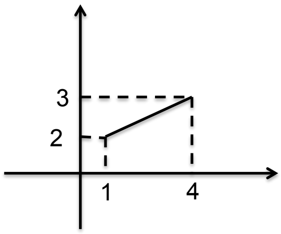
```
polar(CP) {
  PP.rho = sqrt(CP.x * CP.x + CP.y * CP.y);
  if (CP.x == 0.0) {
    if (CP.y < 0.0) theta1 =  0.0 - pi/2.0;
    else theta1 =  pi/2.0;
  } else {
    arctg = atan(CP.y / CP.x);
    if (CP.x >= 0.0) theta1 = arctg;
    else {
      if (CP.y < 0.0) theta1 =  arctg - pi;
      else theta1 = arctg + pi; }
  }
  if (theta1 >= 0) PP.theta = theta1;
  else PP.theta = theta1 + 2 * pi;
  return PP;
}
```

(timp uniform: $O(1)$)

Un segment este reprezentat de o pereche de puncte:

`{A -> {x -> 1, y |-> 2} B -> {x -> 4, y -> 3}}`

Accesarea coordonatelor: `A.x` `A.y` `B.x` `B.y` ...



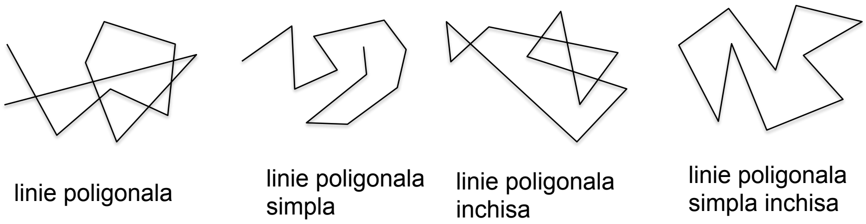
$Segm = Str\langle \mathtt{A} : Point, \mathtt{B} : Point\rangle = \{\{\mathtt{rho} \to \rho\ \mathtt{theta} \to \theta\} \mid \rho, \theta \in Float\}$

$Point \in \{CPoint, PPoint\}$

Linii poligonale

Structura de date: lista liniare de puncte
Pot fi:

- simple
- închise/deschise



Structuri de date pentru linii poligonale

- liste liniare

$PolygLine = List\langle Point\rangle$

Crearea unei liste cu punctele:

```
A = { x -> 3 y -> 5 };
B = { x -> 2 y -> 1 };
C = { x -> 0 y -> 0 };
```

se poate face prin

```
L = emptyList;
L.pushBack(A);
L.pushBack(B);
L.pushBack(C);
```

sau

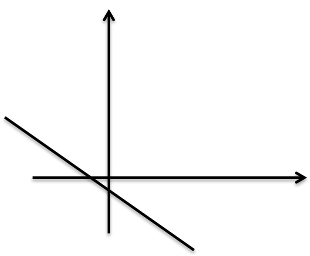
```
L = emptyList;
L[0] = A;
L[1] = B;
L[2] = C;
```

sau

```
L = emptyList;
L.insert(0, A);
L.insert(1, B);
L.insert(2, C);
```

[1ex]

O dreaptă este reprezentată printr-o ecuatie liniară: $a \cdot x + b \cdot y + c = 0$



Structura de date: structura
$Line = Str\langle \texttt{a} : Float, \texttt{b} : Float, \texttt{c} : Float \rangle$

Exemplu: dreapta $3x + 4y + 2 = 0$ este reprezentată de structura
`d |-> {a -> 3 b -> 4  c -> 2}`

Dreapta care trece prin două puncte distincte `P` şi `Q` 1/3

- se consideră sistemul:
$$\begin{cases} \texttt{d.a} * \texttt{P.x} + \texttt{d.b} * \texttt{P.y} + \texttt{d.c} = 0 \\ \texttt{d.a} * \texttt{Q.x} + \texttt{d.b} * \texttt{Q.y} + \texttt{d.c} = 0 \end{cases}$$
- îhm, două ecuaţii şi trei necunoscute?

  Distingem cazurile:
- dreapta este paralelă cu Oy: rezultă $P.x = Q.x$ iar ecuaţia este $x = P.x$ (sau $x = Q.x$), i.e., $d.a = 1, d.b = 0, d.c = -P.x$

Dreapta care trece prin două puncte distincte `P` şi `Q` 2/3

- dreapta NU este paralelă cu Oy: rezultă $d.b \neq 0$ şi sistemul devine
$$\begin{cases} \texttt{P.y} = \texttt{m} * \texttt{P.x} + \texttt{n} \\ \texttt{Q.y} = \texttt{m} * \texttt{Q.x} + \texttt{n} \end{cases}$$
  unde $m = -\dfrac{a}{b} = \dfrac{P.y - Q.y}{P.x - Q.x}$, $n = -\dfrac{c}{a} = P.y - \dfrac{P.y - Q.y}{P.x - Q.x} * P.x$
  Luăm $d.a = -m, d.b = 1, d.c = -n$ şi obţinem forma mult mai cunoscută $y = mx + n$ pentru ecuaţie, unde m este panta dreptei.
- $P = Q$: dreapta este nedefinită; luăm $d.a = d.b = d.c = 0$

Dreapta care trece prin două puncte distincte `P` şi `Q` 1/3

```
line(P,Q) {
  if (P.x == Q.x && P.y == Q.y)
    return {a -> 0.0 b -> 0.0 c -> 0.0};
  if (P.x == Q.x) {
    l.a = 1.0;
    l.b = 0.0;
    lc = P.x;
    return l;
  }
  l.a = 0.0 - (P.y - Q.y) / (P.x - Q.x);
  l.b = 1.0;
  l.c = 0.0 - P.y - l.a * P.x;
  return l;
}
```

1.3 Operaţii primitive

Fie `P` un punct şi `d` o dreaptă. Relativ la d, P se poate afla:

- *într-un semiplan*: `d.a * P.x + d.b * P.y + c > 0`
- *pe dreaptă*: `d.a * P.x + d.b * P.y + c = 0`
- *pe celălalt semiplan*: `d.a * P.x + d.b * P.y + c < 0`

Notăm cu (d, P) semiplanul determinat de dreapta d şi punctul P.

Poziţionarea faţă de un segment `AB`:

- se determină dreapta suport
- dacă se află pe dreaptă, se verifică dacă este între `A` şi `B`

Se poate testa şi în ordine inversă.

Timp uniform: $O(1)$

Intersecţia a două drepte 1/2

$$\begin{cases} a_1 \cdot x + b_1 \cdot y + c_1 = 0 \\ a_2 \cdot x + b_2 \cdot y + c_2 = 0 \end{cases}$$

$$det = \begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} \quad det_x = \begin{vmatrix} -c_1 & b_1 \\ -c_2 & b_2 \end{vmatrix} \quad det_y = \begin{vmatrix} a_1 & -c_1 \\ a_2 & -c_2 \end{vmatrix} [1ex]$$

$det = 0$, drepte paralele (excludem cazul când coincid)

$det \neq 0$, sistemul are solutie unică: $x = \dfrac{det_x}{det}$, $y = \dfrac{det_y}{det}$

În notaţia structurilor de date pentru drepte:

$$det = \begin{vmatrix} \texttt{d1.a} & \texttt{d1.b} \\ \texttt{d2.a} & \texttt{d2.b} \end{vmatrix} = \texttt{d1.a} * \texttt{d2.b} - \texttt{d1.b} * \texttt{d2.a}$$

Intersecţia a două drepte 2/2

```
lineIntersection(l1, l2) {
  det = l1.a *l2.b - l1.b*l2.a;
  detx = l1.b*l2.c - l1.c *l2.b;
  dety = l1.c*l2.a - l1.a *l2.c;
  if (det == 0.0) return emptySet;
  P.x = detx / det;
  P.y = dety / det;
  return singletonSet(P);
}
```

Timp uniform: $O(1)$

Intersecţia a două segmente (când există)

Soluţia 1:

- se determină punctul P de intersecţie a dreptelor suport;
- se verifică dacă P aparţine segmentelor;

Soluţia 2:

- se verifică dacă pentru fiecare segment capetele sale sunt deoparte şi de alta a dreptei suport determinate de celălalt segment (*detalii pe tablă*);

Soluţia 3 (sweep line):

- se baleiază (mătură) planul cu o dreaptă orizontală (sau verticală) şi se analizează poziţiile punctelor eveniment (oarecum asemănător ca la 2);

Timp uniform: $O(1)$

Orientarea a trei puncte (primitiva `ccw`) 1/3

CCW = "Counter-Clock-Wise" (sensul invers arcelor de ceasornic) [1ex] Fie A, B, C trei puncte.

$$det(A, B, C) = \begin{vmatrix} \mathtt{A.x} & \mathtt{A.y} & 1 \\ \mathtt{B.x} & \mathtt{B.y} & 1 \\ \mathtt{C.x} & \mathtt{C.y} & 1 \end{vmatrix}$$

- $det(A, B, C) > 0$: A, B, C formează un ciclu în sens invers arcelor de ceasornic (întoarcere stânga)
- $det(A, B, C) < 0$: A, B, C formează un ciclu în sensul arcelor de ceasornic (întoarcere dreapta)
- $det(A, B, C) = 0$: A, B, C sunt coliniare

Timp uniform: $O(1)$

Convenţie: $det(A, B, C) \rightarrow$ `sign2xTriArea(A,B,C)` (vom vedea mai târziu de ce)

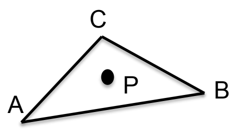
Orientarea a trei puncte (primitiva `ccw`) 2/3

```
sign2xTriArea(A, B, C) {
  d1 = B.y * A.x + C.y * B.x + A.y * C.x;
  d2 = C.x * B.y + B.x * A.y + A.x * C.y;
  return d1 - d2;
}

ccw(A, B, C)
/*
  turn left = +1;
  turn right = -1;
  colinear = 0;
*/
{
  ax2 = sign2xTriArea(A, B, C);
  if (ax2 > 0.0) return 1;
  if (ax2 < 0.0) return -1;
  return 0;
}
```

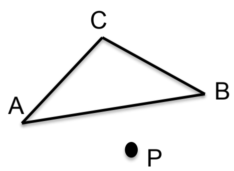
Orientarea a trei puncte (primitiva `ccw`) 3/3

```
A = {x -> 1.0 y -> 1.0};
B = {x -> 3.0 y -> 1.0};
C = {x -> 2.0 y -> 2.0};

t1 = ccw(A, B, C);
t2 = ccw(A, C, B);
t3 = ccw(A, B, B);

krun ../tests/comp-geom1/ccw.alk -cINIT=".Map"
<k>
    .K
</k>
<state>
    A |-> { (x -> 1e+00) (y -> 1e+00) }
    B |-> { (x -> 3e+00) (y -> 1e+00) }
    C |-> { (x -> 2e+00) (y -> 2e+00) }
    t1 |-> 1
    t2 |-> -1
    t3 |-> 0
</state>
```

Localizarea unui punct faţă de un triunghi 1/7



`ccw(P, A, B)`, `ccw(P, B, C)` şi `ccw(P, C, A)` *au toate acelaşi semn.*



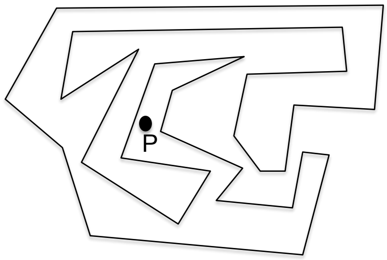
`ccw(P, A, B)`, `ccw(P, B, C)` şi `ccw(P, C, A)` *NU au toate acelaşi semn.*

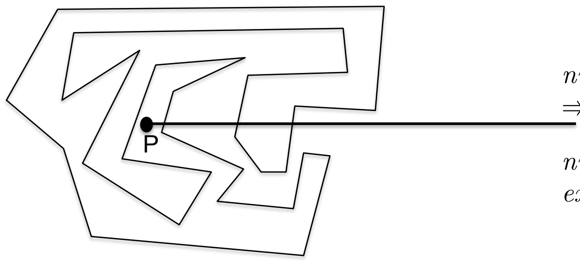
Timp uniform: $O(1)$

Localizarea unui punct faţă de o linie poligonală simplă închisă 2/7

Theorem (Jordan)
Orice curbă simplă închisă împarte planul în două regini distincte: interiorul liniei (mărginită) şi exteriorul (nemărginită).

Instance: O linie poligonală simplă închisă L şi un punct P. *Question*: Aparţine P interiorului lui L?[1.5ex]

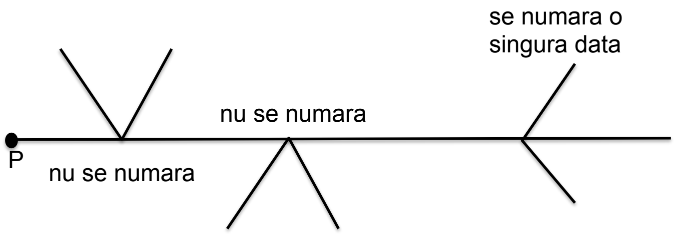




număr de intesecţii impar $\Rightarrow$ *interior,*

număr de intesecţii par$\Rightarrow$ *exterior*

Localizarea unui punct faţă de o linie poligonală simplă închisă 3/7
Număratul intersecţiilor trebuie făcut cu atenţie deoarece sunt cazuri de excepţie:



+ cazurile când o latură este inclusă în semidreaptă.

Calculul unei intersecţii: $O(1)$ Determinarea numărului de intersecţii: $O(n)$, n numărul de segmente ale liniei L

Presupunem că $L[0]$ nu se află pe semidreaptă şi că P nu se află pe linia poligonală L.

```
isInteriorOf(L, P) {
  L.pushBack(L[0]);
  counter = 0;
  for (i = 0; i < L.size()-1; i = j)
    counter = counter + count(i, j);
  return counter % 2 == 1;
}
```

Localizarea unui punct faţă de o linie poligonală simplă închisă 5/7
`count(i, out j)`:

- dacă $L[i]$ şi $L[i+1]$ sunt de aceeaşi parte a semidreptei, atunci $\texttt{j} = \texttt{i} + 1$ şi întoarce zero;

- altfel determină $\{Q\} = line(L[i], L[i+1]) \cap$ dreapta suport a semidreptei (Q există deoarece cazul când cele două drepte sunt paralele este exclus de itemul precedent);
- dacă $Q.x < P.x$, atunci `j` $=$ `i` $+$ `1` şi întoarce zero;
- dacă $Q.x \geq P.x$ şi $Q \neq L[i+1]$, atunci `j` $=$ `i` $+$ `1` şi întoarce unu;
- dacă $Q.x \geq P.x$ şi $Q = L[i+1]$:
  - determină primul $j > i+1$ care nu aparţine semidreptei (rezultă că nu aparţine nici dreptei suport);
  - dacă $L[i]$ şi $L[j]$ sunt de aceeaşi parte a semidreptei, atunci întoarce zero;
  - altfel întoarce unu;

Localizarea unui punct faţă de o linie poligonală simplă închisă 6/7

- dreapta suport a semidreptei:

```
dP. a = 0; dP.b = 1; dP.c =  0 - P.y;
```

- $L[i]$ şi $L[i+1]$ ($L[j]$) sunt de aceeaşi parte a semidreptei:

```
onTheSameSide(P, Q, l) {
  return (l.a * P.x + l.b * P.y + l.c) *
         (l.a * Q.x + l.b * Q.y + l.c) > 0;
}
```

- $\{Q\} = line(L[i], L[i+1]) \cap$ dreapta suport a semidreptei:

```
Q = lineIntersection(line(L[i], L[i+1]), dP);
```

- determină primul $j > i+1$ care nu aparţine semidreptei:

```
j = i+1;
while(L[j].y == P.y && L[j].x >= P.x) ++j;
```

Localizarea unui punct faţă de o linie poligonală simplă închisă 7/7

```
count(i, out j) {
  dP. a = 0; dP.b = 1; dP.c =  0 - P.y;  //dreapta suport a semidreptei
  if(onTheSameSide(L[i], L[i+1], d)) {
    j = i+1;
    return 0;
  }
  Q = lineIntersection(line(L[i], L[i+1]), dP)[0];
  if (Q.x < P.x) {
    j = i+1;
    return 0;
  }
  if (Q.x != L[i+1].x || Q.y != L[i+1].y) {
    j = i+1;
    return 1;
  }
  j = i+1;
  while(L[j].y == P.y && L[j].x >= P.x) ++j;
  if(onTheSameSide(L[i], L[j], d))
    return 0;
  return 1;
}
```

Aria unui triunghi 1/2

Theorem
Dacă A, B, C formează o întoarcere stânga atunci `sign2xTriArea`(A, B, C) este dublul ariei triunghiului ABC (de unde si numele).

```
a1 = sign2xTriArea(A, B, C);
a2 = sign2xTriArea(A, C, B);
```

```
krun ../tests/comp-geom1/ccw.alk -cINIT=".Map"
<k>
    .K
</k>
<state>
    A |-> { (x -> 1e+00) (y -> 1e+00) }
    B |-> { (x -> 3e+00) (y -> 1e+00) }
    C |-> { (x -> 2e+00) (y -> 2e+00) }
    a1 |-> 2e+00
    a2 |-> -2e+00
</state>
```

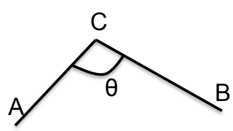
Aria unui triunghi 2/2
Theorem
Dacă A, B, C formează o întoarcere stânga şi P un punct oarecare din plan, atunci `sign2xTriArea`$(P, A, B)+$ `sign2xTriArea`$(P, B, C)+$ `sign2xTriArea`(P, C, A) este dublul ariei triunghiului ABC.

```
P =  {x -> -7.0 y -> -9.0};
```

```
a3 = sign2xTriArea(P, A, B) +
     sign2xTriArea(P, B, C) +
     sign2xTriArea(P, C, A);
```

```
krun ../tests/comp-geom1/ccw.alk -cINIT=".Map"
<k>
    .K
</k>
<state>
    A |-> { (x -> 1e+00) (y -> 1e+00) }
    B |-> { (x -> 3e+00) (y -> 1e+00) }
    C |-> { (x -> 2e+00) (y -> 2e+00) }
    P |-> { (x -> -7e+00) (y -> -9e+00) }
    a3 |-> 2e+00
</state>
```

Unghiul convex format de trei puncte 1/3



- distanţa dintre două puncte:

$$\mathtt{dist(P,Q)} = \sqrt{(\mathtt{Q.x} - \mathtt{P.x}) * (\mathtt{Q.x} - \mathtt{P.x}) + (\mathtt{Q.y} - \mathtt{P.y}) * (\mathtt{Q.y} - \mathtt{P.y})}$$

- se aplică teorema cosinusului

```
a = dist(C, B);
b = dist(C, A);
c = dist(A,B);
theta = acos((a*a + b*b - c*c)/ 2*a*b);
```

Timp uniform: $O(1)$

Unghiul convex format de trei puncte 2/3

```
dist(P, Q) {
  d1 = (Q.x-P.x)*(Q.x-P.x);
  d2 = (Q.y-P.y)*(Q.y-P.y);
  return sqrt(d1 + d2);
}

angle(A, C, B) {
  a = dist(C, B);
  b = dist(C, A);
  c = dist(A, B);
  return acos((a*a + b*b - c*c)/ (2*a*b));
}
```

Unghiul convex format de trei puncte 3/3

```
A = {x -> 1.0 y -> 1.0};
B = {x -> 3.0 y -> 1.0};
C = {x -> 2.0 y -> 2.0};
D = {x -> 5.0 y -> 1.0};
theta1 = angle(A, B, C);
theta2 = angle(B, C, A);
theta3 = angle(B, A, C);
theta4 = angle(A, B, D);

krun ../tests/comp-geom1/angle.alk -cINIT=".Map"
<k>
    .K
</k>
<state>    ...
    theta1 |-> 7.853981633974485e-01
    theta2 |-> 1.5707963267948963e+00
    theta3 |-> 7.853981633974485e-01
    theta4 |-> 3.1415926535897931e+00
</state>
```

Triangularizare (definiţia)

O diagonală a unui poligon L este un segment AB determinat de două vârfuri A şi B cu proprietatea că orice punct Q din AB, $Q \notin \{A, B\}$, se află în interiorul lui L.

Proposition

Orice poligon cu $n \geq 4$ vârfuri are cel puţin o diagonală.

Theorem (Triangularizare)

Orice poligon cu $n \geq 3$ poate fi împărţit în triunghiuri ducând diagonale.

Demonstractia prin inducţie după n.

Graful dual al unei triangularizări

Graful dual unei triangularizări:

- vârfuri: triunghiuri ale traingularizării
- există muchie între două vârfuri dacă acestea partajează o diagonală.

Proposition
Graful dual al unei triangularizări a unui poligon este un arbore.

"Ureche" a unui poligon

O ureche a unui poligon L este o secvenţă de trei vârfuri consecutive A, B, C a.î. AC este diagonală. B se numeşte vârful urechii.

Dacă A, B, C este ureche atunci $\widehat{ABC}$ este convex ($< \pi$).

Theorem
Orice poligon cu $n \geq 3$ vârfuri are cel puţin două urechi care nu se suprapun.

Urechile sunt frunze în arborele dual.

Aria unui poligon 1/2

Theorem
Dublul ariei unui poligon $P_0P_1 \dots P_{n-1}$, $n \geq 3$, cu vârfurile parcurse in sens invers arcelor de ceasornic, este

$$\begin{aligned}&\texttt{sign2xTriArea}(Q, P_0, P_1) + \texttt{sign2xTriArea}(Q, P_1, P_2) + \cdots + \\ &\texttt{sign2xTriArea}(Q, P_{n-2}, P_{n-1}) + \texttt{sign2xTriArea}(Q, P_{n-1}, P_0) \qquad (1)\end{aligned}$$

unde Q este un punct oarecare din plan.

Demonstraţia prin inducţie după n şi se utilizează că poligonul are o ureche.

Aria unui poligon 2/2

Suma (1) este egală cu

$$\sum_{i=0}^{n-1} (P_i.xP_{i+1}.y - P_{i+1}.xP_i.y)$$

care este egală cu

$$\sum_{i=0}^{n-1} (P_i.x + P_{i+1}.x)(P_{i+1}.y - P_i.y)$$

considerând Q originea şi făcând calcule.

Proposition
Aria unui poligon cu $n \geq 3$ vârfuri poate fi calculat cu n înmuţiri şi $2n$ adunări/scăderi.