**Algoritmi avansati (AA)**

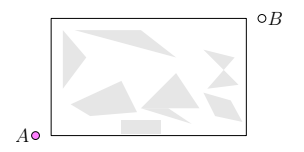
* + 1. **Algoritmi geometrici**

**CURS ~ M.S. Stupariu**

**Curs 1 – 21.02.2023**

Introducere

» Algoritmii geometrici sunt legați de Geometria Computaţională: complexitatea

computationald a problemelor geometrice în contextul analizei algoritmilor [Lee &

Preparata, 1984]

» Problemă (exemplu):

Cum poate fi deplasat discul din A în B fără a atinge obstacolele?

» Complexitatea: memorie, timp, calcule

» Tehnici utilizate: construcții incrementale, divide et impera, plane-sweep,

transformări geometrice, etc.

Criterii numerice. Raport şi test de orientare

Criterii numerice — poziția relativă a unor puncte

» Stabilirea unor relații între puncte / ordonare

» Context 1D

» ordonare (relativ la un sistem de coordonate)

» raport (independent de alegerea unui sistem de coordonate)

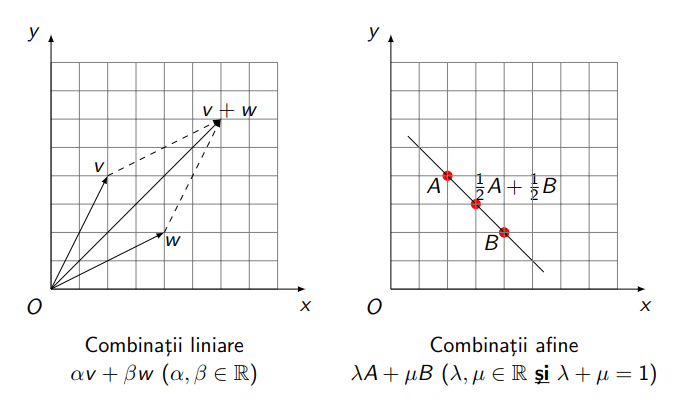
» Context 2D

» ordonare (relativ la un sistem de coordonate — posibile alegeri:

coordonate carteziene, coordonate polare)

» testul de orientare (independent de alegerea unui sistem cartezian

de coordonate)

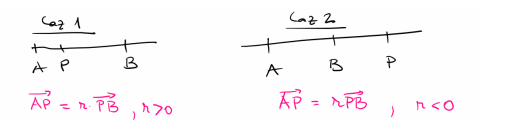


Conceptul de raport

» Lema Fie A si B două puncte distincte în R”. Pentru orice punct

Pc AB, P # B există un unic scalar r E R\ {—1} astfel ca AP=r PB. Reciproc, fiecărui scalar r € R\ {—1}, îi corespunde un unic punct P € AB.

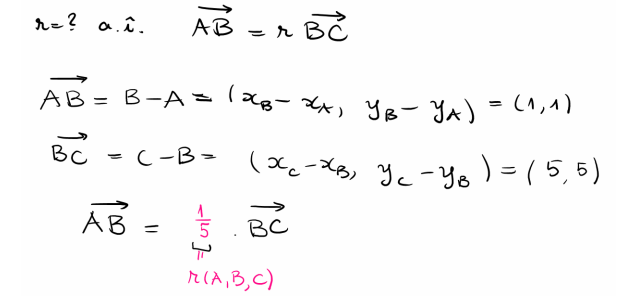
» Definiție Scalarul r definit în lema anterioară se numește raportul punctelor A, B, P (sau raportul în care punctul P împarte segmentul [AB]) si este notat cu r(A, P, B).



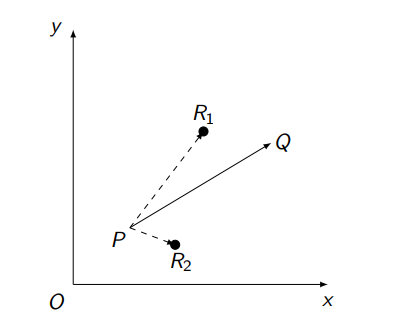
» Observaţie importantă. In calcularea raportului, ordinea punctelor este esenţială. Modul în care este definită această noțiune (mai precis ordinea în care sunt considerate punctele) diferă de la autor la autor.

Raport- exemple

(i) În R? considerăm punctele A = (1,1), B = (2,2), C = (7,7). Determinăm raportul r(A, B, C).



(ii) În R3 considerăm punctele A = (1,2,3), B= (2,1,-1), C = (0,3,7). Atunci punctele A, B, C sunt coliniare și avem r(A, C, B) = -3, r(B,C,A) = 2 r(C,AB)=1, r(C,B, A) = 2.

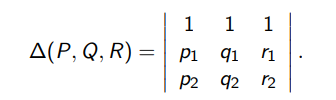
(iii) Fie A, B două puncte din R" și M = LA LB. Atunci r(A,M,B) =1, r(M,A,B) = 1.

Testul de orientare – motivaţie

Poziţia relativă a două puncte faţă

de un vector / o muchie orientată

Enunt principal

» Propozitie. Fie P = (p1,p2), Q = (q1, q2) două puncte distincte din planul R2, fie R = (r1, r2) un punct arbitrar și

Atunci R este situat:

(i) pe dreapta PQ ⬄ det(P,Q, R) = 0 ("ecuaţia dreptei");

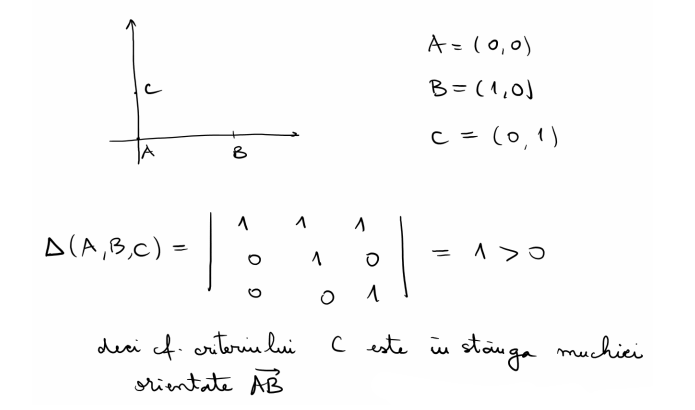
(ii) "în dreapta” segmentului orientat PQ ⬄ det(P,Q,R) < 0;

(iii) "în stânga” segmentului orientat PQ ⬄ det(P,Q,R) > 0.

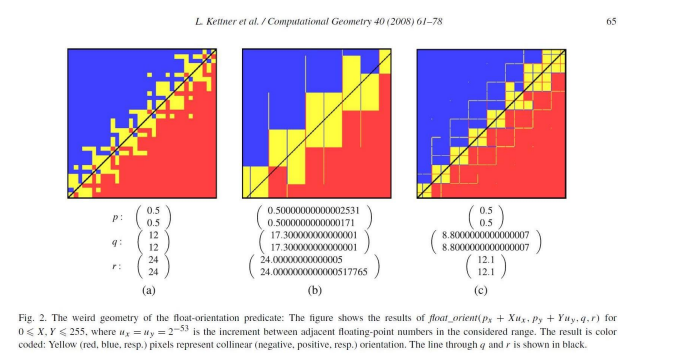
» Obs. Testul de orientare se bazează pe calculul unui polinom de

gradul II (det(P, Q, R)).

Testul de orientare – exemplu



Limitari – robustete si erori de rotunjire



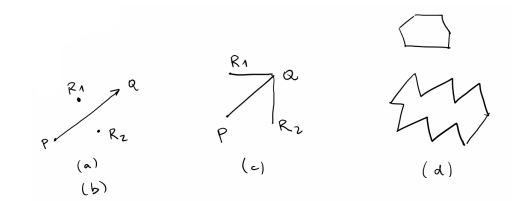
Aplicatii

(a) dacă un punct este în dreapta / stânga unei muchii orientate;

(b) dacă două puncte sunt de o parte si de alta a unui segment / a unei drepte;

(c) natura unui viraj în parcurgerea unei linii poligonale (la dreapta / la stânga);

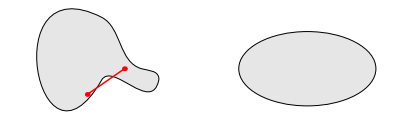
(d) natura unui poligon (convex / concav).



Multimi convexe: generalitati

» Conceptul de mulțime convexă:

O mulțime M C R™ este convexă dacă oricare ar fi p,q € M, segmentul [pq] este inclus în M.

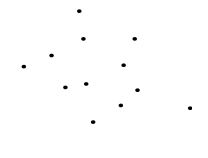
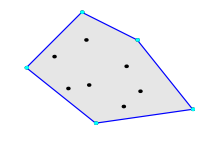


Mulțimea din stânga nu este convexă, întrucât există două puncte, pentru care segmentul determinat nu este inclus în mulțime (punctele cu această proprietate nu sunt unice !).

» Problematizare:

Mulţimile finite cu cel putin două elemente nu sunt convexe —> necesară acoperirea convexă.

Acoperire convexă a unei mulțimi (finite) P: concept





» Caracterizări echivalente:

» Cea mai "mică" (în sensul incluziunii) mulțime convexă care conţine P.

» Intersecţia tuturor mulțimilor convexe care contin P.

» Mulțimea tuturor combinațiilor convexe ale punctelor din P. O combinație convexă a punctelor Py, Ps, ..., P, este un punct P de forma

P=a1P1+...+ anPn, a1,...,an € [0,1], a1 +... + an=1.

» Problematizare: Aceste caracterizări echivalente nu conduc la un algoritm de determinare a acoperirii convexe.

Acoperire convexă a unei mulțimi (finite) P: problematizare

» Dacă P C Rd este finită, acoperirea sa convexă, Conv(P) este un politop convex.

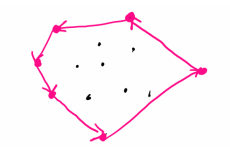
» Cazuri particulare: d = 1 (segment); d = 2 (poligon); d = 3(poliedru).

» Cazul d = 1: acoperirea convexă este un segment; algoritmic: parcurgere a punctelor (complexitate O(n)).

» În continuare: d = 2.

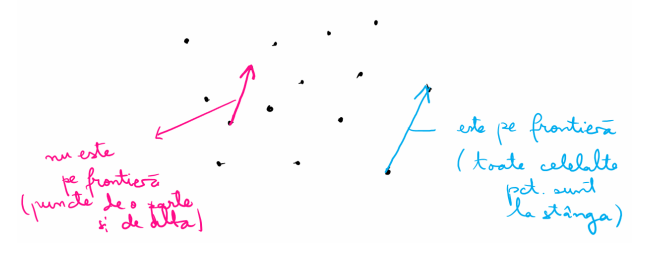
Acoperire convexă a unei mulțimi finite P (practic)

» De fapt, dacă P este finită, acoperirea sa convexă, Conv(P) este un poligon convex.

» Problemă: Cum determinăm, algoritmic, vârfurile acestui poligon?

» Convenţie: Sensul de parcurgere a frontierei este cel trigonometric.

Un algoritm "lent": idee de lucru



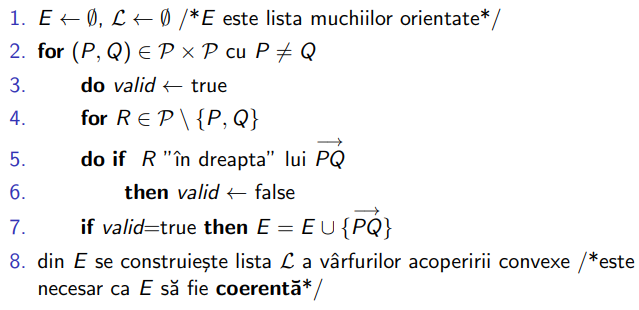
» Sunt considerate muchiile orientate.

» Q: Cum se decide dacă o muchie orientată fixată este pe frontieră?

» A: Toate celelalte puncte sunt "în stanga” ei (v. “testul de orientare” ).

Input: O mulțime de puncte P din R2?.

Output: O listă £ care conţine vârfurile ce determină frontiera acoperirii convexe, parcursă în sensul trigonometric.

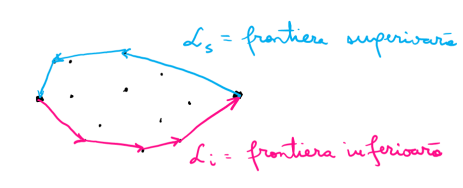


» **Complexitate: O(n3)**

» Complexitate algebrica: polinoame de gradul II

» Tratarea cazurilor degenerate: poate fi adaptat.

» Robustetea: datorită erorilor de rotunjire este posibil ca algoritmul să nu returneze o listă coerentă de muchii.

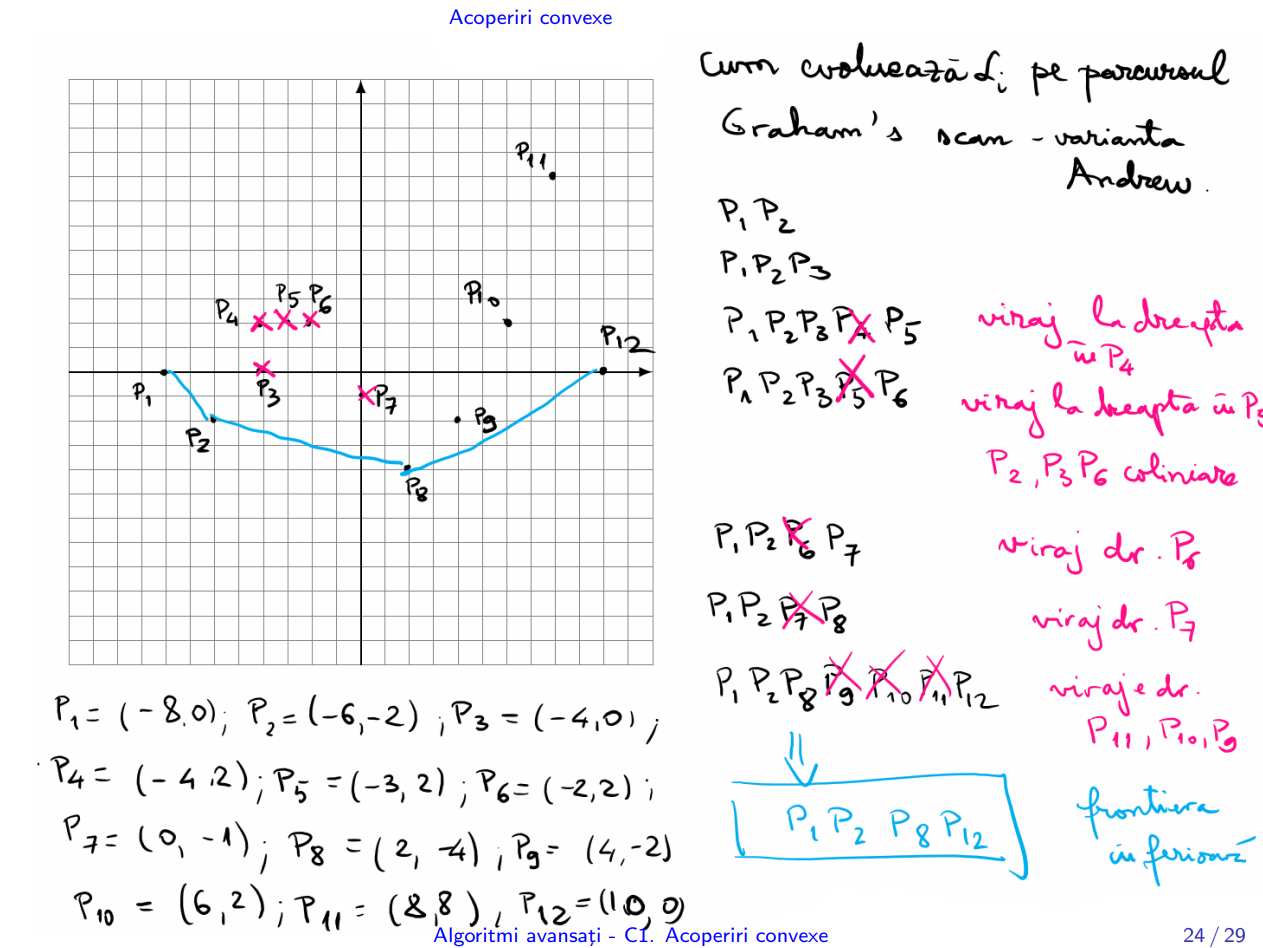
 Graham's scan, varianta Andrew: idee de lucru

» Punctele sunt mai întâi sortate si renumerotate lexicografic.

» Algoritmul este de tip incremental, punctele fiind adăugate unul câte unul, fiind apoi eliminate anumite puncte.

» Q: Cum se decide dacă trei puncte sunt vârfuri consecutive ale acoperirii convexe?

» A: Se efectuează un "viraj la stânga” în punctul din mijloc.

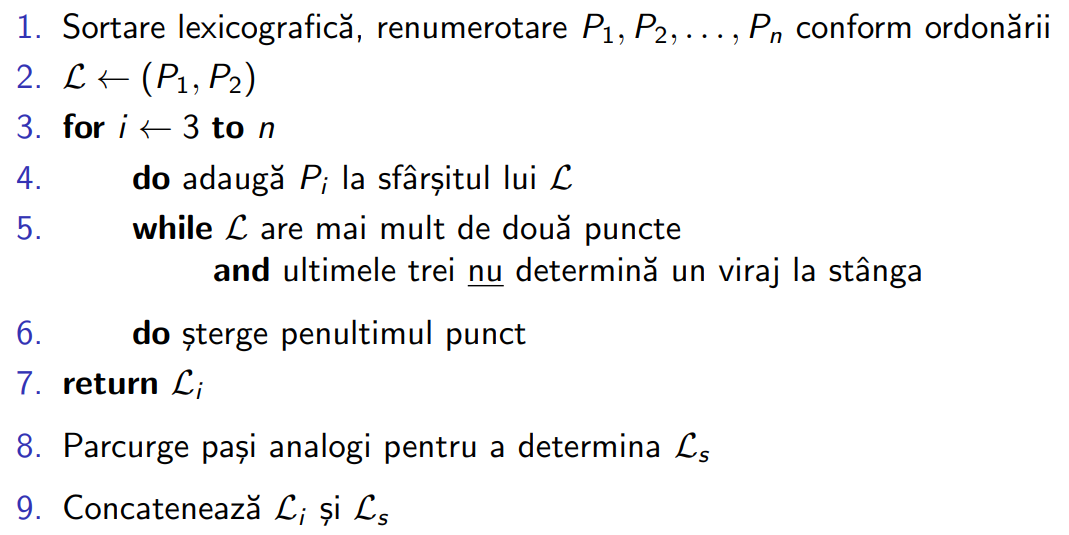




Graham's scan, varianta Andrew (algoritm)

» Input: O mulțime de puncte P din R?.

» Output: O listă L care conține vârfurile ce determină frontiera acoperirii convexe, parcursă în sens trigonometric.



» **Complexitate: O(n log n)**

» Tratarea cazurilor degenerate: corect.

» Robustetea: datorită erorilor de rotunjire este posibil ca algoritmul să returneze o listă eronată (dar coerentă) de muchii.

Jarvis" march / Jarvis" wrap [1973]

» Algoritm de tip incremental. Nu necesită sortare prealabilă.

» Iniţializare: un punct care este sigur un vârf al acoperirii convexe (e.g. punctul cel mai de jos / din stânga / stânga jos).

» Lista se actualizează prin determinarea succesorului: "cel mai la dreapta” punct.

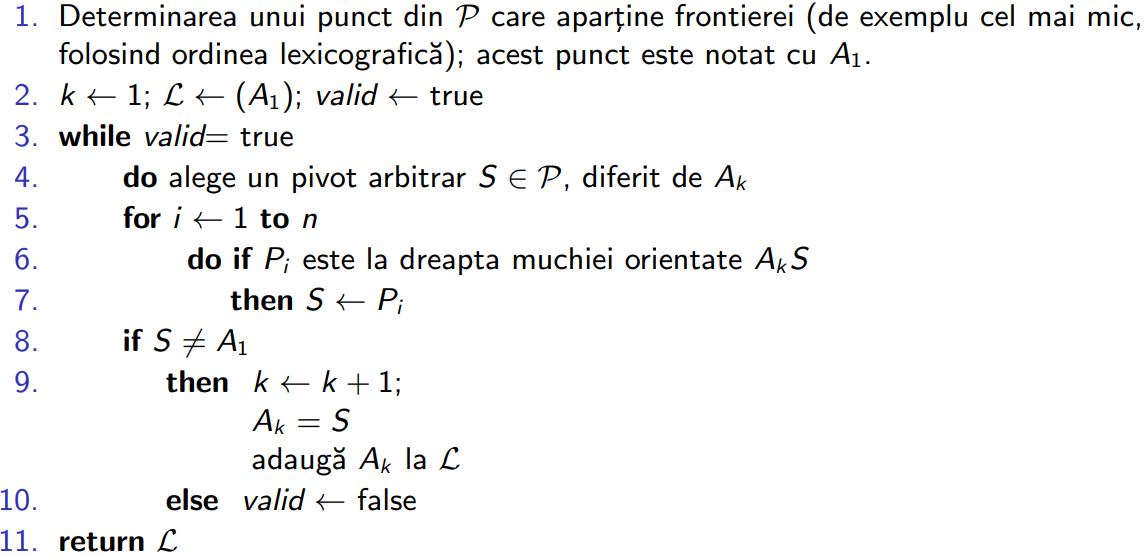
» Implementare: două abordări (i) ordonare; (ii) testul de orientare.

» Complexitate: O(hn), unde h este numărul punctelor de pe frontiera acoperirii convexe.

» Algoritmul lui Chan “combină” ideile celor doi algoritmi, ajungând la complexitatea-timp O(n log h).

» Input: O mulțime de puncte necoliniare P = (Pi, Pz, ..., Pa) din R\* (n > 3).

» Output: O listă £ care conţine vârfurile ce determină frontiera acoperirii convexe, parcursă în sens trigonometric.



Alte directii de lucru

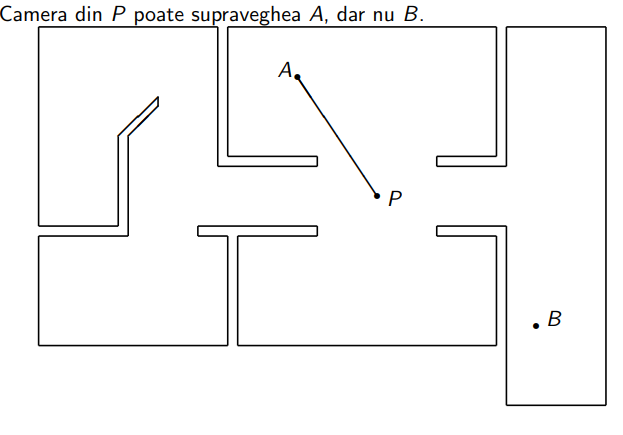
» Aplicaţii: grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.

» Pot fi stabilite legături cu algoritmi studiati în alt context. De exemplu: Traveling Salesman Problem, în context euclidian (costurile sunt date de distanțele dintre puncte). În acest caz, ordinea în care nodurile de pe frontieră apar în traseul optim coincide cu ordinea în care acestea apar în parcurgerea frontierei acoperirii convexe

» Algoritmi pentru spaţii euclidiene de dimensiune m > 3.

» Algoritmi eficienți pentru determinarea acoperirii convexe pentru vârfurile unui poligon arbitrar.

» Algoritmi dinamici (on-line, real-time, convex hull maintenance).

**Curs 2 – 28.02.2023**

Problema galeriei de arta

» O galerie de artă poate fi interpretată (în

contextul acestei probleme) ca un poligon P

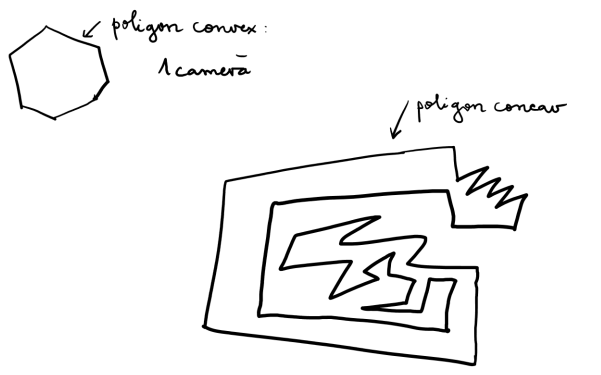
(adică o linie poligonală fără autointersecţii)

având n vârfuri.

» O cameră video (vizibilitate 360°) poate fi

identificată cu un punct din interiorul lui P;

ea poate supraveghea acele puncte cu care poate fi unită printr-un segment inclus în interiorul poligonului.

» Problema galeriei de arta: câte camere video sunt necesare pentru a supraveghea o galerie de artă și unde trebuie amplasate acestea?

Numarul de camere vs. forma poligonului

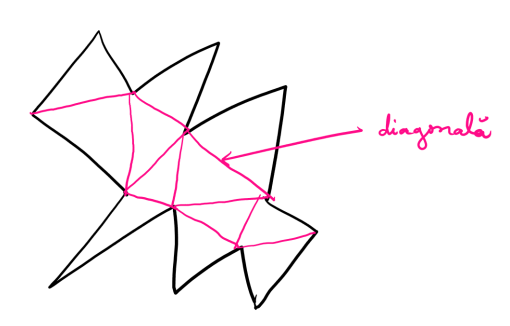
» Se dorește exprimarea numărului de camere necesare pentru

supraveghere în funcție de n (sau controlarea acestuia de către n).

» Pentru a supraveghea un spaţiu având forma unui poligon convex,

este suficientă o singură cameră.

» Numărul de camere depinde și de forma poligonului: cu cât forma

este mai "complexă", cu atât numărul de camere va fi mai mare.

» Principiu: Poligonul considerat: descompus în triunghiuri (triangulare).

Definiție formală

» Fie P un poligon plan.

» (i) O diagonală a lui P este un segment ce unește două vârfuri ale

acestuia si care este situat în interiorul lui P.

» (ii) O triangulare Tp a lui P este o descompunere a lui P în triunghiuri, dată de o mulțime maximală de diagonale ce nu se intersectează.

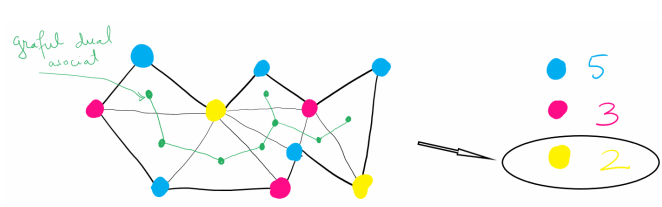
» **Lema**. Orice poligon admite o diagonală.

» **Teoremă**. Orice poligon admite o triangulare. Orice triangulare a unui poligon cu n vârfuri conţine exact n — 2 triunghiuri.

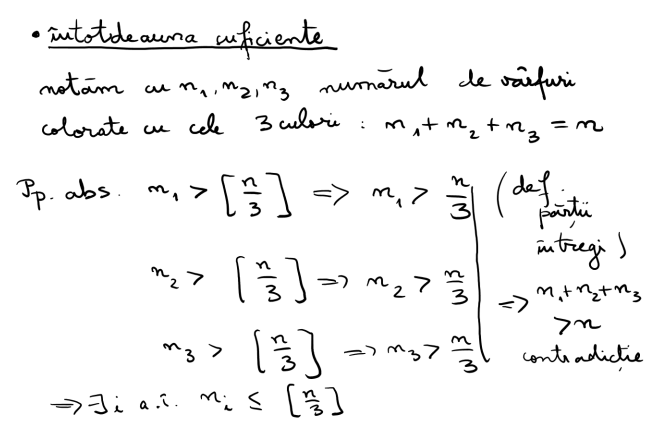
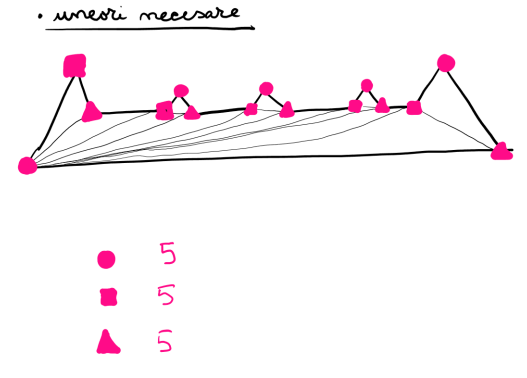
» Amplasarea camerelor se poate face în vârfurile poligonului.

» Dată o pereche (P, Tp) se consideră o 3-colorare a acesteia: fiecărui vârf îi corepunde o culoare dintr-un set de 3 culori și pentru fiecare triunghi, cele 3 vârfuri au culori distincte.

» Observaţie. Dacă P este linie poligonală fără autointersectii o astfel de colorare există, deoarece graful dual asociat perechii (P, Tp) este arbore.



» **Teoremă**. [Chvătal, 1975; Fisk, 1978] Pentru un poligon cu n vârfuri, [n/3] camere sunt uneori necesare și întotdeauna suficiente pentru ca fiecare punct al poligonului să fie vizibil din cel puţin una din camere.



Metode de triangulare: ear cutting / clipping / trimming

» Concepte pentru un poligon P = (Pi, P2,..., Pa):

» **Varf convex/concav( “reflex” )**: se stabilește cu testul de orientare. Un vârf este convex ⬄ are același tip de viraj ca vârful “cel mai din stânga".

» **Varf principal**: Pi este principal dacă [Pi-1,Pi+1] este diagonală (echivalent:nu există un alt vârf în interiorul sau pe laturile triunghiului Pi-1PiPi+1).

» **Ear (vârf / componentă de tip E)**: este un vârf principal convex [Meisters,1975]. Dacă Pi este componentă de tip E, atunci segmentul [Pi-1Pi+1] nu intersectează laturile poligonului si este situat în interiorul acestuia, adică este “diagonală veritabilă”, iar triunghiul Pi-1PiPi+1 poate fi “eliminat”

» **Mouth (vârf / componentă de tip M)**: este un vârf principal concav [Toussaint, 1991].

» **Criterii de clasificare a vârfurilor**: (i) vârf convex/concav; (ii) vârf principal/nu.

» **Teoremă**. (Two Ears Theorem [Meisters, 1975]) Orice poligon cu cel putin 4 vârfuri admite cel putin două componente de tip E care nu se suprapun.

» **Corolar**. Orice poligon admite (cel putin) o diagonală.

» Găsirea unei componente de tip E: complexitate O(n) [ElGindy, Everett, Toussaint, 1993]. Se bazează pe Two Ears Theorem!

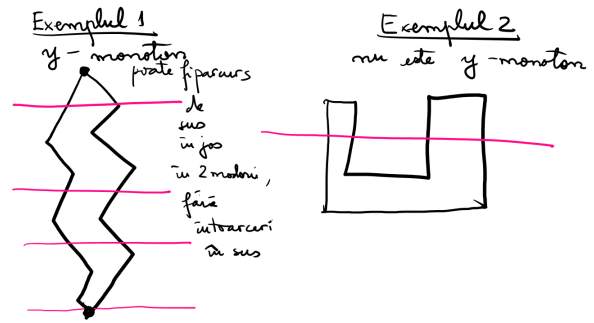
» Algoritmul de triangulare bazat de metoda ear cutting: **complexitate O(n2)**.

» Algoritmi de triangulare eficienți: complexitate O(n) pentru poligoane y-monotone [Garey etal., 1978] (algoritmul este descris pe slide-urile următoare).

» Descompunerea unui poligon oarecare in componente y-monotone poate fi realizată cu un algoritm de complexitate O(n log n) [Lee, Preparata, 1977]. În concluzie, avem următoarea

**Teoremă.** Un poligon poate fi triangulat folosind un algoritm de **complexitate O(n log n)**.

» Concept: poligon y-monoton



Metoda - paradigma dreptei de baleiere (line sweep)

» Se consideră o dreaptă de baleiere orizontală. Algoritmul reține o serie de informații legate de structura geometrică analizată.

» **Statut** al dreptei de baleiere: stivă a vârfurilor deja întâlnite, dar care “mai au nevoie de diagonale” / “mai pot să apară în triunghiuri. (Clarificare. Q: Când este eliminat un vârf? A: Când a fost trasată o diagonală situată “mai jos de acesta”).

» **Evenimente**: modificarea statutului. Sunt vârfurile poligonului, în prealabil ordonate după y; pentru fiecare vârf ştim dacă este pe lanţul din stânga sau pe cel din dreapta.

» **Invariant**: “pâlnie” (funnel) în care (i) vârful de sus este convex; (ii)pe o parte: o muchie; (iii) pe cealaltă parte: muchie / succesiune de vârfuri concave.

**SEMINAR ~ S. Popescu**

**SEMINAR 1 – 20.02.2023**

1. Fie punctele A = (1, 2, 3), B = (4, 5, 6) ∈ R3.

a) Fie C = (a, 7, 8). Aratati ca exista a astfel ca punctele A, B, C sa fie coliniare si pentru a astfel determinat calculati raportul r(A, B, C).

b) Determinati punctul P astfel ca raportul r(A, P, B) = 1.

c) Dati exemplu de punct Q astfel ca r(A, B, Q) < 0 ̧si r(A, Q, B) < 0.

2. Fie punctele P = (1, −1), Q = (3, 3).

a) Calculati valoarea determinantului care apare in testul de orientare pentru

muchia orientata P Q si punctul de testare O = (0, 0).

b) Fie Rα = (α, −α), unde α ∈ R. Determinat ̧i valorile lui α pentru care

punctul Rα este situat ˆın dreapta muchiei orientate P Q.

3. Fie M = {P1, P2, . . . , P9}, unde P1 = (−2, 4), P2 = (−1, 1), P3 = (0, 1),

P4 = (2, 1), P5 = (4, 3), P6 = (5, 5), P7 = (6, 9), P8 = (8, 4), P9 = (10, 6).

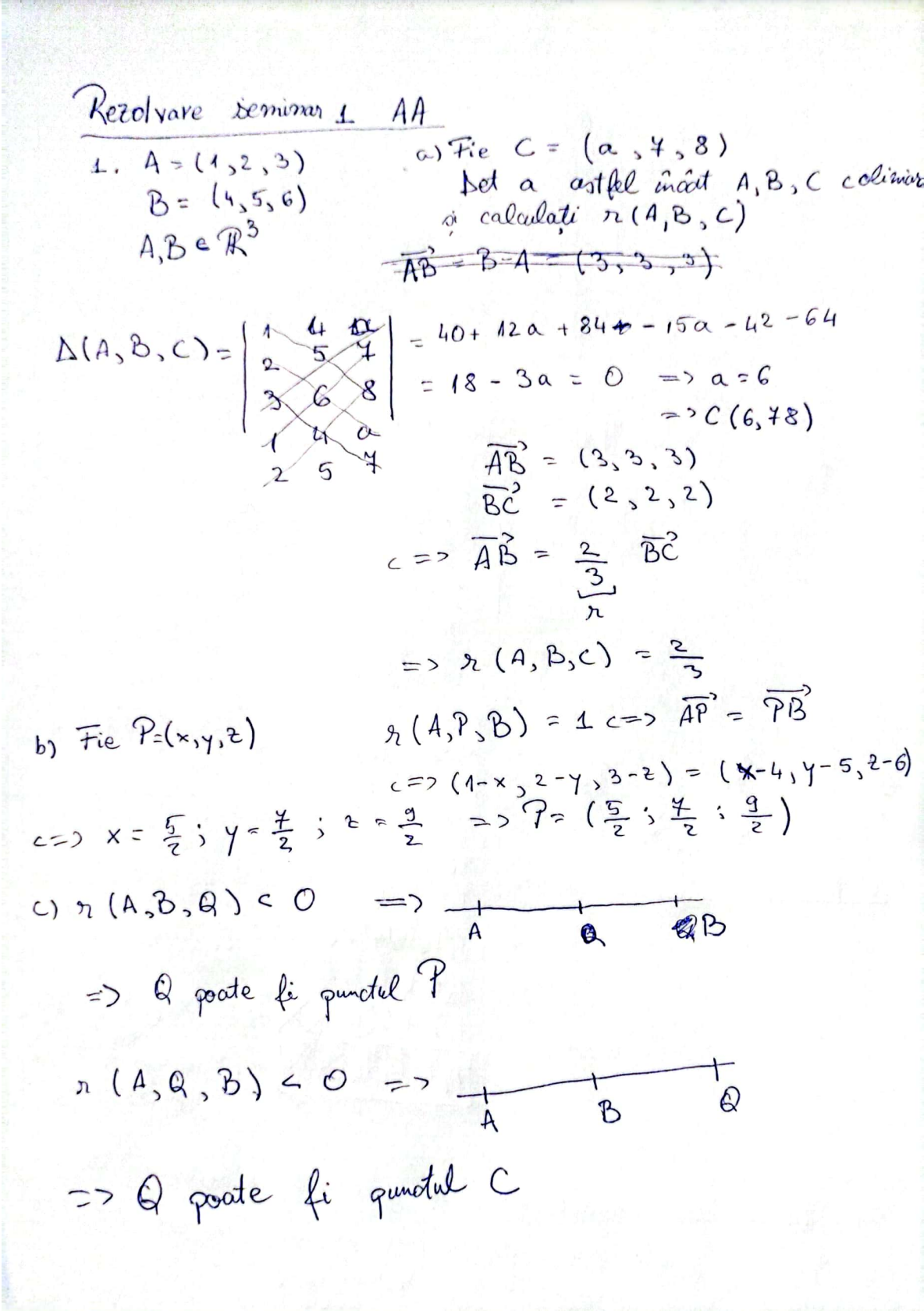
Detaliat ̧i cum evolueaz ̆a lista Li a vˆarfurilor care determin ̆a marginea inferioar ̆a

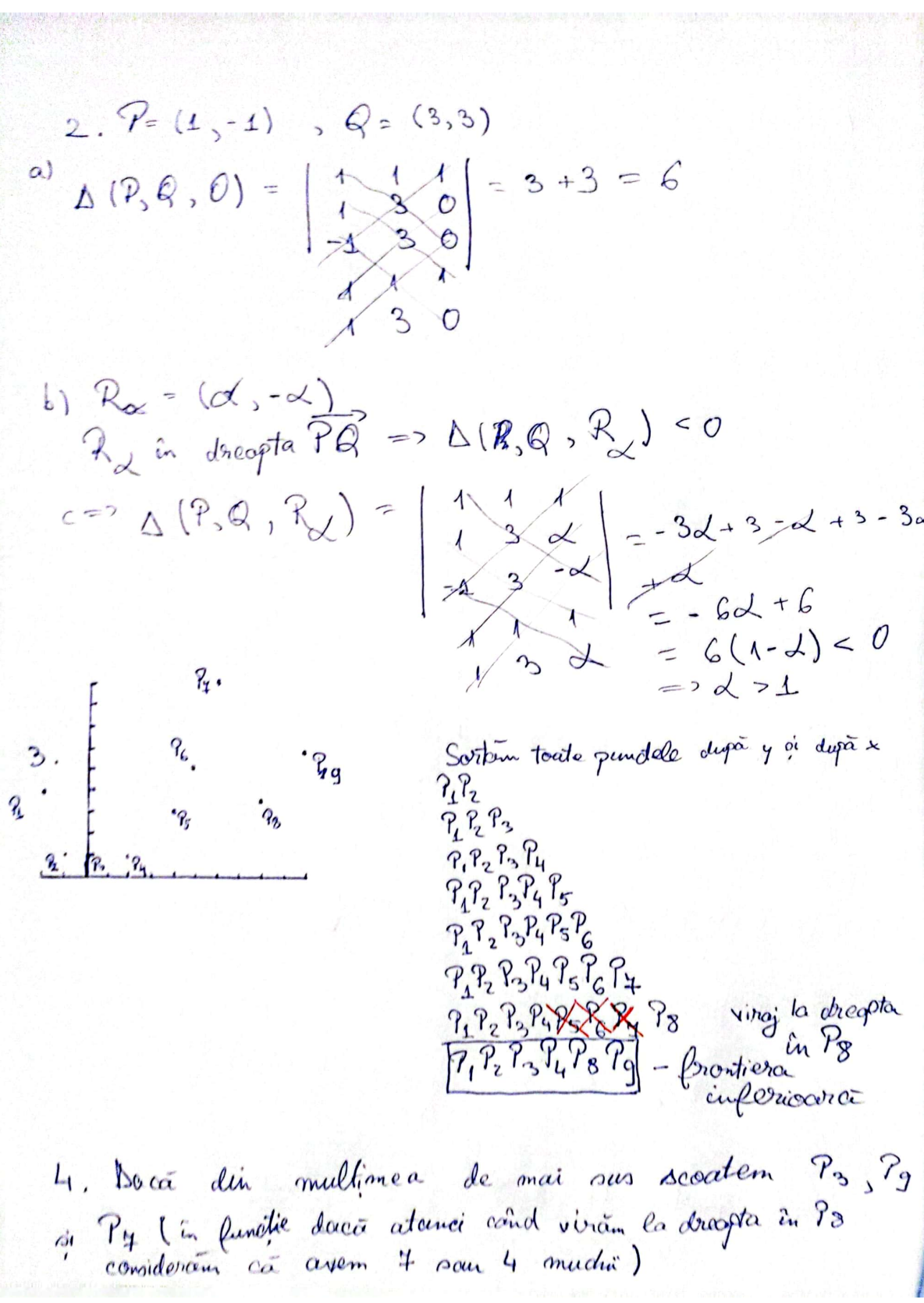
a frontierei acoperirii convexe a lui M, obt ̧inut ̆a pe parcursul Graham’s scan,

varianta Andrew. Justificati!

4. Dati un exemplu de mult ̧ime M din planul R2 pentru care, la final, Li are 4 elemente, dar, pe parcursul algoritmului, numarul maxim de elemente al lui Li este egal cu 6 (Li este lista varfurilor care determina marginea inferioara a frontierei acoperirii convexe a lui M, obtinuta pe parcursul Graham’s scan, varianta Andrew). Justificati!

5. Discutati un algoritm bazat pe paradigma Divide et impera pentru determinarea acoperirii convexe. Analizati complexitatea-timp.

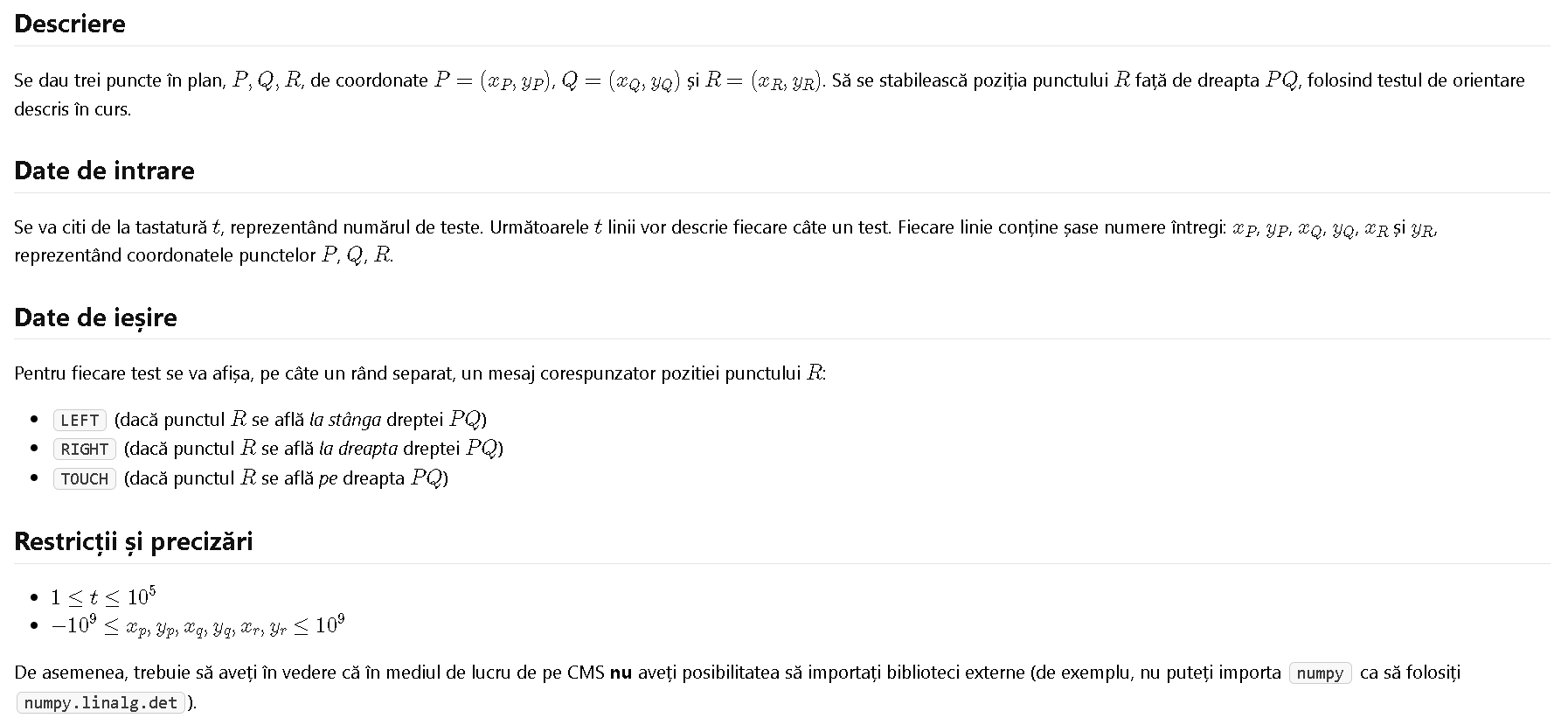




**LABORATOR ~ S. Stancioiu**

**LABORATOR 1 – 22.02.2023**

Exercitiul 1. Testul de orientare 10/10



**class** **Point**(object):

**def** \_\_init\_\_(self, x, y):

self**.**X **=** x

self**.**Y **=** y

**def** **getX**(self):

**return** self**.**X

**def** **getY**(self):

**return** self**.**Y

**def** **makeDeterminant**(A,B,C):

determinant **=** B**.**getX()**\***C**.**getY()**+**A**.**getX()**\***B**.**getY()**+**A**.**getY()**\***C**.**getX()**-**A**.**getY()**\***B**.**getX()**-**B**.**getY()**\***C**.**getX()**-**C**.**getY()**\***A**.**getX()

**if** (determinant **>** 0):

**return** "LEFT"

**elif** (determinant **==** 0):

**return** "TOUCH"

**else**:

**return** "RIGHT"

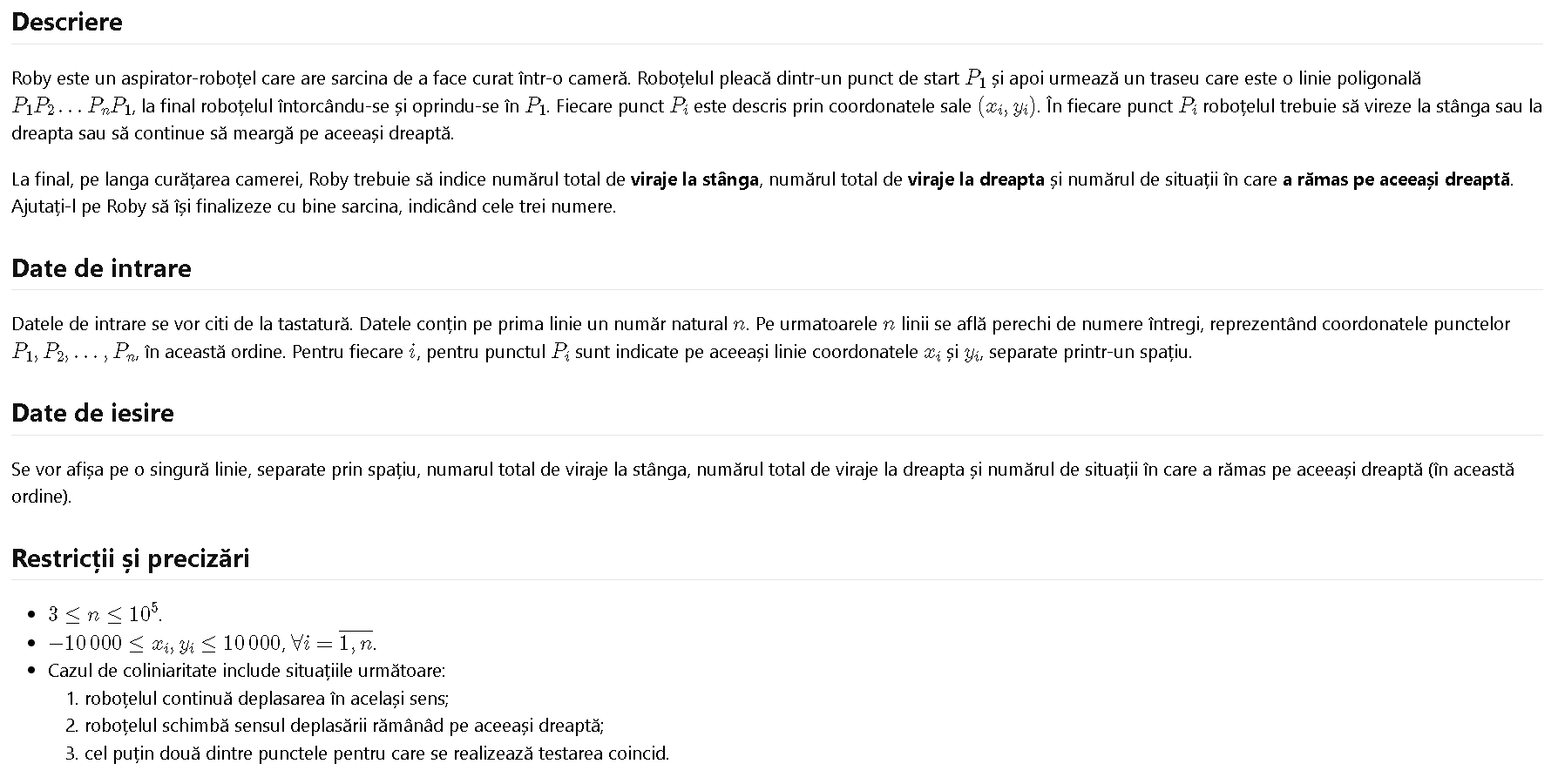
n **=** int(input())

**for** i **in** range(n):

test **=** list(map(int,input()**.**split(" ")))

print(makeDeterminant(Point(test[0],test[1]),Point(test[2],test[3]),Point(test[4],test[5])))

Exercitiul 2. Roby 10/10



**class** **Point**(object):

**def** \_\_init\_\_(self, x, y):

self**.**X **=** x

self**.**Y **=** y

**def** **getX**(self):

**return** self**.**X

**def** **getY**(self):

**return** self**.**Y

**def** **makeDeterminant**(A,B,C):

determinant **=** B**.**getX()**\***C**.**getY()**+**A**.**getX()**\***B**.**getY()**+**A**.**getY()**\***C**.**getX()**-**A**.**getY()**\***B**.**getX()**-**B**.**getY()**\***C**.**getX()**-**C**.**getY()**\***A**.**getX()

**if** (determinant **>** 0):

**return** "LEFT"

**elif** (determinant **==** 0):

**return** "TOUCH"

**else**:

**return** "RIGHT"

n **=** int(input())

left **=** 0

middle **=** 0

right **=** 0

firstPoint **=** list(map(int,input()**.**strip()**.**split(" ")))

A **=** Point( firstPoint[0], firstPoint[1])

primulPunct **=** A

secondPoint **=** list(map(int,input()**.**split(" ")))

B **=** Point (secondPoint[0],secondPoint[1])

**for** i **in** range(n**-**2):

test **=** list(map(int,input()**.**strip()**.**split(" ")))

C **=** Point(test[0],test[1])

testul **=** makeDeterminant(A,B,C)

**if**(testul **==** "LEFT"):

left **+=** 1

**elif**(testul **==** "RIGHT"):

right **+=** 1

**else**:

middle **+=** 1

A **=** B

B **=** C

testul **=** makeDeterminant(A,B,primulPunct)

**if**(testul **==** "LEFT"):

left **+=** 1

**elif**(testul **==** "RIGHT"):

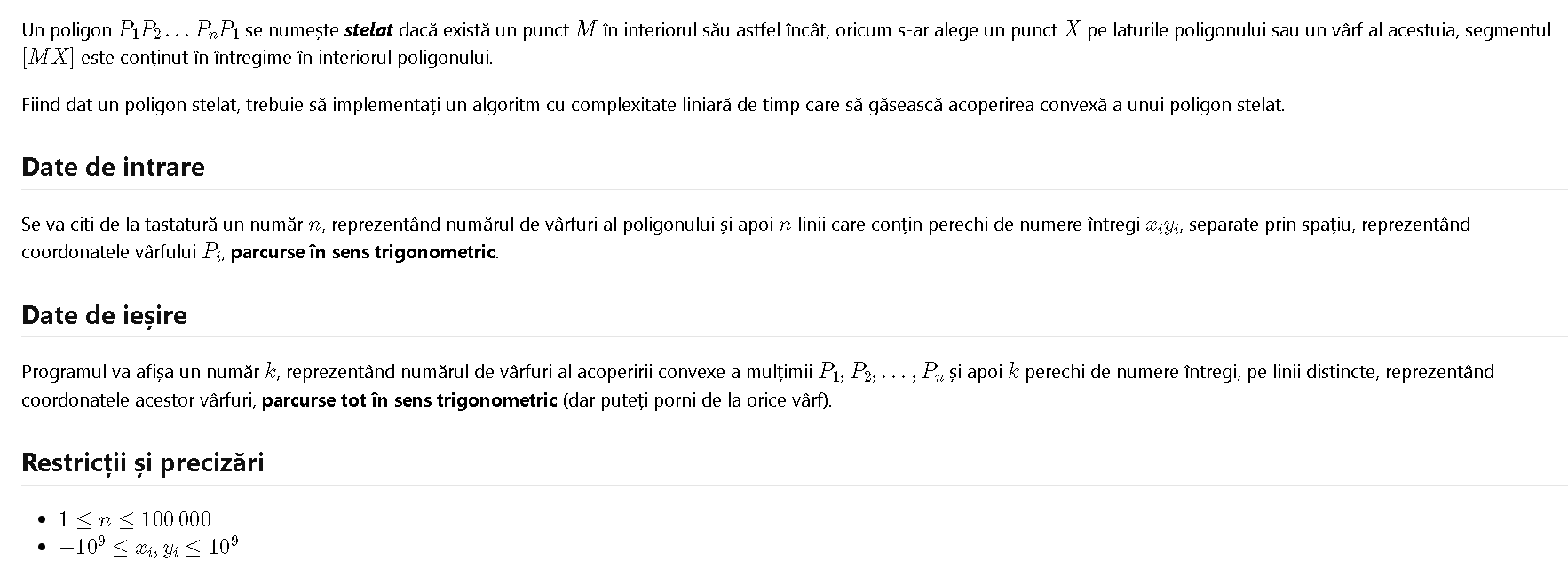
right **+=** 1

**else**:

middle **+=** 1

print(str(left) **+** " " **+** str(right) **+** " " **+** str(middle))

Exercitiu 3. Acoperirea convexa a unui poligon stelat 10/10



#O clasa Point simpla care ma ajuta la vizibilitatea codului zic eu

#intrucat nu am frontiera de frontiera[5][0] si am frontiera[5].getX()

class Point(object):

def \_init\_(self, x, y):

self.X = x

self.Y = y

def getX(self):

return self.X

def getY(self):

return self.Y

#Functia de determinant folosita la celelalte 2 exercitii unde ma interesa sa scot aceste raspunsuri

#de LEFT,TOUCH si RIGHT

def makeDeterminant(A,B,C):

determinant = B.getX()\*C.getY()+A.getX()\*B.getY()+A.getY()\*C.getX()-A.getY()\*B.getX()-B.getY()\*C.getX()-C.getY()\*A.getX()

if (determinant > 0):

return "LEFT"

elif (determinant == 0):

return "TOUCH"

else:

return "RIGHT"

#Functia rightOrNot imi returneaza True daca ultimu punct e pe linie cu penultimul si antepenultimu

#sau se afla la dreapta lui

def rightOrNot(frontiera):

determinant = frontiera[-2].getX() \* frontiera[-1].getY() + frontiera[-3].getX() \* frontiera[-2].getY() + frontiera[-1].getX() \* frontiera[-3].getY() - frontiera[-2].getX() \* frontiera[-3].getY() - frontiera[-1].getX() \* frontiera[-2].getY() - frontiera[-3].getX() \* frontiera[-1].getY()

if determinant < 0 or determinant == 0:

return True

else:

return False

#Iau numarul de puncte

nrPuncte = int(input())

#Initializez frontiera

frontiera = []

#In acest for testez toate punctele (mai putin ultimele 2 la care vom ajunge la al doilea for)

#Aici daca iau frontiera[-3], frontiera[-2] si frontiera[-1] si rezultatul testului de orientare

#este right (in cazul de fata asteptam un raspuns True de la functia rightOrNot) atunci scot frontiere[-2]

#si continui pana cand fie rezultatul este False(frontiera[-1] se afla in stanga) sau ajung sa am doar 2 punct

for i in range(nrPuncte):

point = list(map(int,input().strip().split(" ")))

frontiera.append(Point(point[0], point[1]))

valid = True

while valid and len(frontiera) >= 3:

valid = rightOrNot(frontiera)

if valid:

frontiera.pop(-2)

# #Aici fac aceeasi verificare de mai sus , dar si pe primele 2 puncte.

# #Astfel, daca cumva ultimul punct face viraj la dreapta in primu scoata ultimul punct si daca

# #cumva al doilea punct se afla in dreapta primului punct, atunci o sa scoatem si primul punct si terminam frontiera

# for i in range(2):

punct = frontiera[0]

frontiera.append(punct)

valid = True

while valid and len(frontiera) >= 0:

valid = rightOrNot(frontiera)

if valid:

frontiera.pop(-2)

frontiera.pop(0)

print(len(frontiera))

for point in frontiera:

print(point.getX(),point.getY(), sep=" ")