Algoritmi Avansati

Algoritmi aproximativi

Definiție **1**

- Un algoritm ALG pentru o problema de **minimzare** se numește ρ-aproximativ, pentru o valoare $\rho > 1$, dacă $ALG(I) \leq \rho \cdot OPT(I)\ pt\ \forall I$ $-$ intrare

- Un algoritm ALG pentru o problema de **maximizare** se numește ρ-aproximativ, pentru o valoare $\rho < 1$, dacă $ALG \geq \rho \cdot OPT(I)\ pt\ \forall I$ $-$ intrare

***Definiție* 2**

Fie *ALG* un algoritm ρ-aproximativ pentru o problema de minimizare. Spunem că factorul de aproximare este ”tight bounded” atunci când avem $\rho = supremum_I \frac{ALG(I)}{OPT(I)}$

Ca să arătăm că un algoritm este ρ-aproximativ ”tight bounded” , trebuie deci să justificăm următoarele 2 lucruri:

1. Trebuie să arătmăm că este ρ-aproximativ, adică $ALG(I) \leq \rho x OPT(I)$ pentru orice intrare *I*
2. Pentru orice ρ'<ρ există un *I* pentru care $ALG(I) > \rho' x OPT(I)$. Adesea totuși ne este mai la îndemână să arătăm ca există un *I* pentru care $ALG(I) = \rho x OPT(I)$

Load Balancing Problem

Input:

- ***m*** **calculatoare identice;** ***n*** **activitați ce trebuiesc procesate. Fiecare activitate** ***j*** **având nevoie de** t_j **unități de timp pentru execuție.**
- **Odată inițiată, fiecare dintre activități trebuie derulată în mod continuu pe același calculator**
- **Un calculator poate executa cel mult o activitate în același timp.**

Scop:

Să asignăm fiecare activitate unui calculator astfel încât să minimizăm timpul până când toate activitățile sunt terminate.

Notații:

- $J(i)$ -submulțimea tuturor activităților (job-urilor) care au fost programate să se desfășoare pe mașina i.
- L_i va reprezenta "load-ul" (timpul de lucru) al mașinii i.
- $L_i = \Sigma_{j \in J(i)} t_j$

  Scop: O asignare a activităților astfel încât L_k este minimizat, unde $k = max_i(L_i)$, adică mașina cu cel mai mare load.

Pseudocodul:

$Load - Balance\left(m, t_1, t_2, \ldots, t_n \right)$

$\quad for\ i = 1\ to\ m :$

$\qquad L_i = 0;\ J(i) = \varnothing$ # $initializare$: $Fiecare\ Load\ este\ 0\ iar\ multimea\ joburilor\ este\ nula\ pt\ fiecare\ masina$

$\quad for\ j = 1\ to\ n:$

$\qquad i = arg\left(min\{L_k | k \in \{1, \ldots, m\}\}\right)$ # $i\ -\ masina\ cu\ incarcatura\ cea\ mai\ mica\ in\ acest\ moment$

$\qquad J(i) = J(i) \cup \{j\}$

$\qquad L_i += t_j$

Care este Factorul de Aproximare?

Lema 1.

$$OPT \geq max\left\{ \frac{1}{m} \sum_{1 \leq j \leq n} t_j,\ max\left\{ t_j | 1 \leq j \leq n \right\} \right\}$$

Lema 2.
Algoritmul descris anterior este un algoritm 2-Aproximativ.
Altfel spus, fie ALG = $max(L_i |\ i \in \{1,\ldots,m\})$ masina "cea mai incarcata". Avem de arătat că ALG≤2xOPT

Ordered-Scheduling Algorithm

Lema 3.

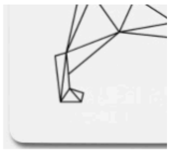
Fie o multime de n activitati cu timpul de procesare $t_1,\ t_2, ...,\ t_n$ astfel incat $t_1 \geq t_2 \geq \ldots t_n$

$Daca\ n > m,\ atunci\ OPT \geq t_{\mathbf{m}} + t_{\mathbf{m+1}}$

TEOREMA 2

Algoritmul descris anterior (Ordered-Scheduling Algorithm) este un algoritm 3/2-aproximativ

Ciclu Hamiltonian (HC-Problem)



Fie *G=(V,E)* un graf neorientat.

Numim *ciclu hamiltonian* un ciclu în *G* cu proprietatea că fiecare nod apare exact o singură dată.

HC-Problem este problema de decizie dacă într-un graf oarecare există sau nu un astfel de ciclu.

HC-Problem este NP-Completa

Traveling Salesman Problem (TSP)

Fie G un graf complet cu ponderi>0 pe muchii.

Evident G este graf hamiltonian, dar se pune problema găsirii ciclului hamiltonian de cost total minim.

Costul unui ciclu este suma costurilor muchiilor din componența sa.

TSP:
”Un vânzător ambulant vrea să își promoveze produsele în *n* locații. El dorește să treacă prin toate localitățile o singură dată, la final ajungând în localitatea de unde a plecat. Pentru a lucra cât mai eficient, vânzătorul dorește să minimizeze costul total al deplasării”

TSP este o problema **NP-hard**. **Găsirea unui algoritm aproximativ este necesară!**

Teorema 1.

Nu există nicio valoare *c* pentru care să existe un algoritm în timp polinomial și care să ofere o soluție cu un factor de aproximare *c* pentru TSP, decât dacă **P=NP**.

Demo: Vom arată că există un asemenea algoritm aproximativ, dacă și numai dacă putem rezolva problema HC în timp polinomial.

Regula triunghiului pe grafuri ne spune că pentru oricare 3 noduri interconectate *u,v,w* avem:

$\text{len}((u,v)) \leq \text{len}((v,w)) + \text{len}((w,u))$

Altfel spus, odată ce am traversat nodurile *u,v,w* - în această ordine, este mai eficient ca să ne întoarcem în *u* direct din *w* decât via *v*.

Observaţie 2:

Fie G un graf complet, ponderat, care respectă regula triunghiului. Şi fie $v_1, v_2, v_3, \ldots, v_k$ un lanţ în graful G. Atunci avem $\text{len}((v_1,v_k)) \leq \text{len}(v_1, v_2, v_3, \ldots, v_k)$

Lema 3:

Fie OPT costul soluției optime pentru TSP, iar MST - ponderea totală a unui Arbore parţial de cost minim pe baza aceluiași graf. Avem relația:

$OPT \geq MST$

Vertex cover problem

Problema:
Fie o rețea de calculatoare în care trebuie să testăm toate conexiunile.

Pentru a testa conexiunile, trebuie să instalăm un program software pe mai multe calculatoare. Acest program poate testa toate conexiunile directe care pleacă din respectivul calculator.

Evident, putem instala acest program pentru a monitoriza întreaga rețea, dar dorim să minimizam intervenția. Deci se pune problema găsirii unei submulțimi de calculatoare de cardinal minim care să poată monitoriza întreaga rețea.

Problema formală:
Fie un graf neorientat $G=(V,E)$.

Numim "acoperire" o submulţime $S \subset V$ cu oriorietatea ca pentru orice $(x,y) \in E$ avem

$x \in S$ sau $y \in S$ (sau $x,y \in S$)

Se pune problema găsirii unei acoperiri S de cardinal minim!

Această problemă este NP-hard.

Algoritmi Genetici

Algoritmi Genetici: Noțiuni

Generație = etapă în evoluția populației
Selecție = proces prin care sunt promovați indivizii cu grad ridicat de adaptare la mediu
Operatori genetici:

- **încrucişare** (combinare, crossover) - indivizii din noua generație moştenesc caracteristicile părinților
- **mutaţie** – indivizii din noua generație pot dobândi şi caracteristici noi

Algoritm

- t=0
- Consideră, **o populație inițială** P(0): **alegem aleator indivizi** din intervalul D
- Cât timp nu există **condiția de terminare**:
  - construim o populație nouă P(t+1) pe baza indivizilor din P(t) astfel:
    - **selecție**: generează o populație intermediară $P^1(t)$ selectând indivizi din **P(t)** după un anumit **criteriu de selecție**
    - aplicăm **operatorul de încrucișare** pentru (unii) indivizi din $P^1(t)$ obținând populația intermediară $P^2(t)$
    - aplicăm operatorul de mutație peste (unii) indivizi din $P^2(t)$ obținând populația **P(t+1)**
    - *opțional: la P(t+1) se adaugă elementul/elementele elitiste din P(t)*
- *t=t+1*

Algoritmi probabilisti

Algoritmi probabilisti

Algoritmii probabilisti pot fi impartiti in 2 (sau 3) clase:

- Algoritmi Monte Carlo:
  - rulează în timp polinomial (rapid) și oferă un răspuns ”probabil” corect
- Algoritmi Las Vegas:
  - oferă mereu răspunsul corect în timp ”probabil” rapid
- Algoritmi *Atlantic City*:
  - rulează în timp ”probabil” rapid și oferă un rezultat ”probabil” corect.

Monte Carlo: Frievald's Algorithm

Problemă: A,B,C - 3 matrici pătrate de dimensiune *nxn*; Trebuie să verificăm dacă AxB=C.

Soluție:

Complexitate: $O(n^2)$

1. **Generam un vector binar *r* de lungime *n* cu $Pr[r_i=1]=½$.**
2. **Dacă Ax(Br)=Cr, return "DA"**
3. **Altfel return "NU"**

Observație: Dacă AxB≠C, atunci Pr[Ax(Br)≠Cr]≥1/2

Primality Testing:
The Solovay-Strassen Test

- **Este utilizat pentru a determina dacă un număr este compus sau *probabil* prim**
- **se bazează pe proprietățile simbolului Jacobi și ale criteriului lui Euler**
- **Dat fiind un număr impar *n* pentru a testa primalitatea, testul alege un număr aleator a în intervalul *[2, n-1]* și calculează simbolul Jacobi *(a/n)*. Dacă *n* este un număr prim, atunci simbolul Jacobi va fi egal cu simbolul Legendre și va satisface criteriul lui Euler**

- **Dacă simbolul Jacobi calculat nu satisface criteriul lui Euler, atunci n este compus. Testul este rulat pentru mai multe iterații pentru a-i crește acuratețea.**

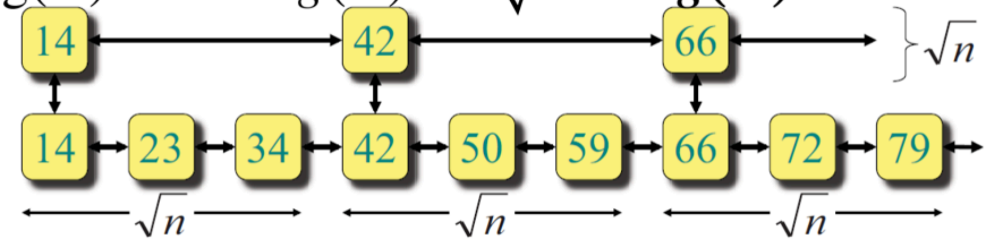
Skip Lists: Numarul de elemente per nivel

$$2\ nivele:\ 2\sqrt{n}$$

$$3\ nivele:\ 3\sqrt[3]{n}$$

$$k\ nivele:\ k\sqrt[k]{n}$$

$$\lg(k)\ nivele:\ lg(k)\sqrt[lg(k)]{n} = 2\mathbf{lg(n)}$$



Skip Lists: Insert (X)

Pentru a insera un nou element X in lista:

- **ii cautam pozitia in nivelul inferior (search(x))**
- **il inseram pe nivelul inferior**
- **il inseram si pe *unele* nivele superioare**

OBSERVATIE: Nivelul inferior va contine intoteauna toate elementele

Q: Pe cate alte nivele inserez X?

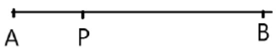
A: Dau cu banul! Daca pica pajura, inserez pe inca un nivel, altfel ma opresc!

Algortimi geometrici

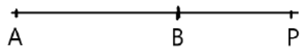
Conceptul de raport

- **Lemă** Fie A și B două puncte distincte în $\mathbb{R}^n$. Pentru orice punct $P \in AB$, $P \neq B$ există un unic scalar $r \in \mathbb{R} \setminus \{-1\}$ astfel ca $\overrightarrow{AP} = r\ \overrightarrow{PB}$. Reciproc, fiecărui scalar $r \in \mathbb{R} \setminus \{-1\}$, îi corespunde un unic punct $P \in AB$.

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



$\overrightarrow{AP} = r\ \overrightarrow{PB}, r > 0$



$\overrightarrow{AP} = r\ \overrightarrow{PB}, r < 0$

- **Observație importantă.** În calcularea raportului, ordinea punctelor este esențială. Modul în care este definită această noțiune (mai precis ordinea în care sunt considerate punctele) diferă de la autor la autor.

(i) În $\mathbb{R}^2$ considerăm punctele $A = (1, 1)$, $B = (2, 2)$, $C = (7, 7)$. Determinăm raportul $r(A, B, C)$.

$$r = ?\ \text{a.î.}\ \overrightarrow{AB} = r\ \overrightarrow{BC}.$$

$$\overrightarrow{AB} = B - A = (x_B - x_A, y_B - y_A) = (1, 1)$$

$$\overrightarrow{BC} = C - B = (x_C - x_B, y_C - y_B) = (5, 5)$$

$$\overrightarrow{AB} = \frac{1}{5}\ \overrightarrow{BC},\ \text{deci}\ r(A, B, C) = \frac{1}{5}.$$

(ii) În $\mathbb{R}^3$ considerăm punctele $A = (1, 2, 3)$, $B = (2, 1, -1)$, $C = (0, 3, 7)$. Atunci punctele A, B, C sunt coliniare și avem $r(A, C, B) = -\frac{1}{2}$, $r(B, C, A) = -2$, $r(C, A, B) = 1$, $r(C, B, A) = -2$.

(iii) Fie A, B două puncte din $\mathbb{R}^n$ și $M = \frac{1}{2}A + \frac{1}{2}B$. Atunci $r(A, M, B) = 1$, $r(M, A, B) = -\frac{1}{2}$.

Testul de orientare

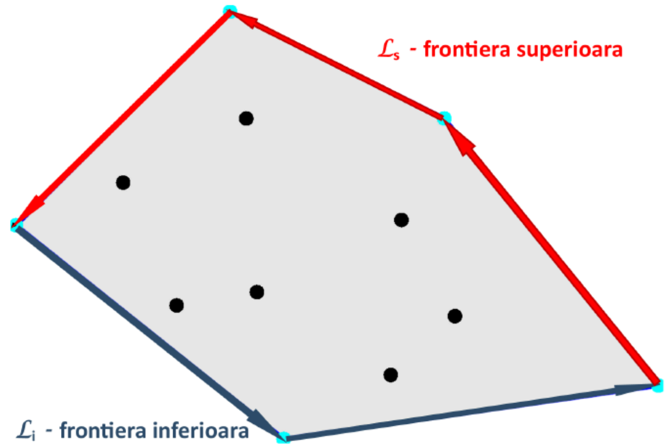
- **Propoziție.** *Fie* $P = (p_1, p_2), Q = (q_1, q_2)$ *două puncte distincte din planul* $\mathbf{R}^2$, *fie* $R = (r_1, r_2)$ *un punct arbitrar și*

$$\Delta(P, Q, R) = \begin{vmatrix} 1 & 1 & 1 \\ p_1 & q_1 & r_1 \\ p_2 & q_2 & r_2 \end{vmatrix}.$$

  Atunci R *este situat:*
  (i) pe dreapta $PQ \Leftrightarrow \Delta(P, Q, R) = 0$ *("ecuația dreptei");*
  (ii) "în dreapta" segmentului orientat $\overrightarrow{PQ} \Leftrightarrow \Delta(P, Q, R) < 0$;
  (iii) "în stânga" segmentului orientat $\overrightarrow{PQ} \Leftrightarrow \Delta(P, Q, R) > 0$.
- **Obs.** Testul de orientare se bazează pe calculul unui polinom de gradul II ($\Delta(P, Q, R)$).

Acoperire convexă a unei mulțimi finite $\mathcal{P}$ (practic)

Graham's scan, varianta Andrew: idee de lucru



- Punctele sunt mai întâi sortate și 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.

Input: O mulțime de puncte $\mathcal{P}$ din $\mathbf{R}^2$.
Output: O listă $\mathcal{L}$ care conține vârfurile ce determină frontiera acoperirii convexe, parcursă în sens trigonometric.

1. Sortare lexicografică, renumerotare $P_1, P_2, \ldots, P_n$ conform ordonării
2. $\mathcal{L} \leftarrow (P_1, P_2)$
3. **for** $i \leftarrow 3$ **to** n
4.   **do** adaugă P_i la sfârșitul lui $\mathcal{L}$
5.   **while** $\mathcal{L}$ are mai mult de două puncte
           and ultimele trei <u>nu</u> determină un viraj la stânga
6.   **do** șterge penultimul punct
7. **return** $\mathcal{L}_i$
8. Parcurge pași analogi pentru a determina $\mathcal{L}_s$
9. Concatenează $\mathcal{L}_i$ și $\mathcal{L}_s$

- Complexitatea: $O(n \log n)$.
- Tratarea cazurilor degenerate: corect.
- Robustețea: datorită erorilor de rotunjire este posibil ca algoritmul să returneze o listă eronată (dar coerentă) de muchii.

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din $\mathbf{R}^2$ $(n \geq 3)$.
Output: O listă $\mathcal{L}$ care conține vârfurile ce determină frontiera acoperirii convexe, parcursă în sens trigonometric.

```
Determinarea unui punct din 𝒫 care aparține frontierei (de exemplu cel mai mic,
  folosind ordinea lexicografică); acest punct este notat cu A₁.
k ← 1; ℒ ← (A₁); valid ← true
while valid= true
    do alege un pivot arbitrar S ∈ 𝒫, diferit de A_k
    for i ← 1 to n
        do if P_i este la dreapta muchiei orientate A_k S
           then S ← P_i
    if S ≠ A₁
       then  k ← k + 1;
             A_k = S
             adaugă A_k la ℒ
       else  valid ← false
return ℒ
```

Rezolvarea problemei galeriei de artă

- Fie $\mathcal{P}$ un poligon plan.
- (i) O **diagonală** a lui $\mathcal{P}$ este un segment ce unește două vârfuri ale acestuia și care este situat în interiorul lui $\mathcal{P}$.
- (ii) O **triangulare** $\mathcal{T}_P$ a lui $\mathcal{P}$ este o descompunere a lui $\mathcal{P}$ în triunghiuri, dată de o mulțime maximală de diagonale ce nu se intersectează.

- **Lemă.** Orice poligon admite o diagonală.
- **Teoremă.** *Orice poligon admite o triangulare. Orice triangulare a unui poligon cu n vârfuri conține exact $n-2$ triunghiuri.*

- **Teoremă.** [Chvátal, 1975; Fisk, 1978] *Pentru un poligon cu n vârfuri,* $\left[\frac{n}{3}\right]$ *camere sunt* **uneori necesare** *și întotdeauna* **suficiente** *pentru ca fiecare punct al poligonului să fie vizibil din cel puțin una din camere.*

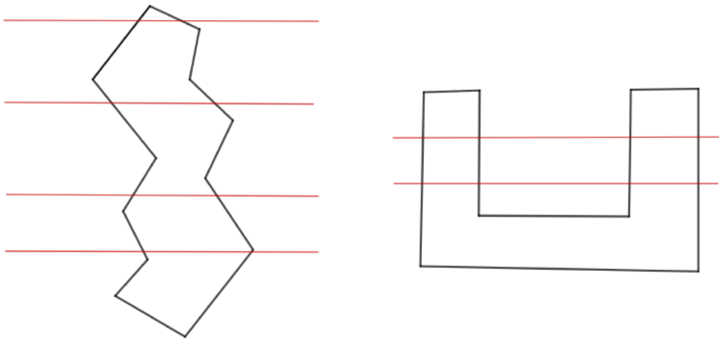
Metode de triangulare: ear cutting / clipping / trimming

- Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \ldots, P_n)$:
  - **Vârf convex/concav("reflex"):** se stabilește cu testul de orientare. Un vârf este convex $\Leftrightarrow$ are același tip de viraj ca vârful "cel mai din stânga".
  - **Vârf principal:** P_i este principal dacă $[P_{i-1}P_{i+1}]$ este diagonală (echivalent: nu există un alt vârf în interiorul sau pe laturile $\Delta P_{i-1}P_iP_{i+1}$).
  - **Ear (vârf / componentă de tip E):** este un vârf principal convex [Meisters, 1975]. Dacă P_i este componentă de tip E, atunci segmentul $[P_{i-1}P_{i+1}]$ nu intersectează laturile poligonului și este situat în interiorul acestuia, adică este "diagonală veritabilă", iar $\Delta P_{i-1}P_iP_{i+1}$ 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 puțin* 4 *vârfuri admite cel puțin două componente de tip E care nu se suprapun.*
- **Corolar.** *Orice poligon admite (cel puțin) 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 $O(n^2)$.

Metode de triangulare: descompunerea în poligoane monotone

- Concept: **poligon y-monoton**

Exemplul 1 Exemplul 2



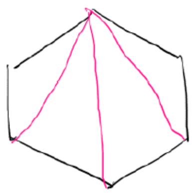
- Poligonul din Exemplul 1 **este y-monoton**: (i) poate fi parcurs de sus în jos în exact două moduri, fără întoarceri - există două lanțuri pe care se poate realiza parcurgerea, (ii) orice dreaptă orizontală intersectează reuniunea dintre poligon și interior după o mulțime conexă ($\emptyset$, punct sau segment). Poligonul din Exemplul 2 **nu este y-monoton**.

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

- **Definiție.** O **triangulare** a unei mulțimi $\mathcal{P}$ este o subdivizare maximală a acoperirii convexe $\mathrm{Conv}(\mathcal{P})$ a lui $\mathcal{P}$ cu triunghiuri ale căror vârfuri sunt elemente ale lui $\mathcal{P}$ (fără autointersecții!)
- Trebuie făcută distincție între triangulare a unui poligon $(P_1, P_2, \ldots, P_n)$ și triangulare a mulțimii subdiacente $\{P_1, P_2, \ldots, P_n\}$ (coincid dacă poligonul este convex!)
- **Comentariu:** Triangulările mulțimilor de puncte sunt esențiale în grafica pe calculator.

Elemente ale unei triangulări

- Dată o mulțime de puncte $\mathcal{P}$ și o triangulare $\mathcal{T}_P$ a sa: **vârfuri, muchii, triunghiuri**.
- Legătură cantitativă între aceste elemente?
- **Propoziție.** *Fie $\mathcal{P}$ o mulțime de n puncte din plan nesituate toate pe o aceeași dreaptă. Notăm cu k numărul de puncte de pe frontiera acoperirii convexe $\mathrm{Conv}(\mathcal{P})$. Orice triangulare a lui $\mathcal{P}$ are $(2n-k-2)$ triunghiuri și $(3n-k-3)$ muchii.*
- **Exemplu:** Cazul unui (unor puncte care formează un) poligon convex: un poligon convex cu n vârfuri poate fi triangulat cu $(n-2)$ triunghiuri, având $(2n-3)$ muchii.



$\underline{n=6}$

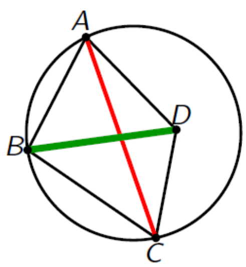
4 Δ
9 muchii

Terminologie, triangulări legale

- Fixată: o mulțime de puncte $\mathcal{P}$. **În cele ce urmează vom presupune că $\mathcal{P}$ este o mulțime de puncte din planul $\mathbb{R}^2$.**
- Fie $\mathcal{T}$ o triangulare a lui $\mathcal{P}$ cu m triunghiuri. Fie $\alpha_1, \alpha_2, \ldots, \alpha_{3m}$ unghiurile lui $\mathcal{T}$, ordonate crescător. **Vectorul unghiurilor lui $\mathcal{T}$ este** $A(\mathcal{T}) = (\alpha_1, \alpha_2, \ldots, \alpha_{3m})$.
- **Relație de ordine pe mulțimea triangulărilor lui $\mathcal{P}$:** ordinea lexicografică pentru vectorii unghiurilor. Fie $\mathcal{T}$ și $\mathcal{T}'$ două triangulări ale lui $\mathcal{P}$. Atunci $A(\mathcal{T}) > A(\mathcal{T}')$ dacă $\exists i$ astfel ca $\alpha_j = \alpha'_j$, $\forall 1 \leq j < i$ și $\alpha_i > \alpha'_i$.
- **Triangulare unghiular optimă:** $\mathcal{T}$ astfel ca $A(\mathcal{T}) \geq A(\mathcal{T}')$, pentru orice triangulare $\mathcal{T}'$.

Muchii ilegale

- **Conceptul de muchie ilegală.** Fie $A, B, C, D \in \mathbb{R}^2$ fixate astfel ca $ABCD$ să fie un patrulater convex; fie $\mathcal{T}_{AC}$, $\mathcal{T}_{BD}$ triangulările date de diagonalele AC, respectiv BD. Muchia AC este **ilegală** dacă
$$\min A(\mathcal{T}_{AC}) < \min A(\mathcal{T}_{BD}).$$
- **Criteriu geometric** pentru a testa dacă o muchie este legală: muchia AC, adiacentă cu triunghiurile ΔACB și ΔACD este ilegală dacă și numai dacă punctul D este situat în interiorul cercului circumscris ΔABC.



- **Criteriu numeric / analitic** pentru a testa dacă o muchie este ilegală.
  - Pentru puncte $A = (x_A, y_A), B = (x_B, y_B), C = (x_C, y_C), D = (x_D, y_D)$:
$$\Theta(A, B, C, D) = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix}$$
  - (i) Punctele A, B, C, D sunt conciclice $\Leftrightarrow \Theta(A, B, C, D) = 0$.
(ii) Fie A, B, C astfel ca ABC să fie un viraj la stânga. Un punct D este situat în interiorul cercului circumscris $\Delta ABC \Leftrightarrow \Theta(A, B, C, D) > 0$.

- **Triangulare legală:** nu are muchii ilegale. **Fapt:** O triangulare legală a unei mulțimi cu n puncte poate fi determinată printr-un algoritm incremental randomizat, cu complexitate-timp medie $O(n \log n)$.

Determinarea complexității algebrice (IIa)

- Pentru a **determina explicit** punctul de intersecție dintre două segmente $[AB]$ și $[CD]$: un punct M este atât pe segmentul $[AB]$, cât și pe segmentul $[CD]$ $\Leftrightarrow$ există $\lambda, \mu \in [0,1]$ astfel ca
$$M = (1-\lambda)A + \lambda B = (1-\mu)C + \mu D \tag{1}$$
(am scris punctul M ca fiind atât o combinație convexă a lui A și B, cât și o combinație convexă a lui C și D).
- Ecuația (1) se scrie în coordonate
$$\begin{cases} x_M = (1-\lambda)x_A + \lambda x_B = (1-\mu)x_C + \mu x_D \\ y_M = (1-\lambda)y_A + \lambda y_B = (1-\mu)y_C + \mu y_D \end{cases} \tag{2}$$
Am obținut sistemul (2) de două ecuații cu două necunoscute (λ, μ), care poate fi rescris sub forma
$$\begin{cases} (x_B - x_A)\lambda + (x_D - x_C)\mu = x_C - x_A \\ (y_B - y_A)\lambda + (y_D - y_C)\mu = y_C - y_A \end{cases} \tag{3}$$

- Pentru sistemul (2), determinantul
$$\Delta = \begin{vmatrix} x_B - x_A & x_D - x_C \\ y_B - y_A & y_D - y_C \end{vmatrix} = (x_B - x_A)(y_D - y_C) - (y_B - y_A)(x_D - x_C)$$
este un polinom de gradul II. Dreptele AB și CD se intersectează într-un singur punct $\Leftrightarrow \Delta \neq 0$.
- Dacă $\Delta \neq 0$, atunci
$$\lambda = \frac{1}{\Delta}\begin{vmatrix} x_C - x_A & x_D - x_C \\ y_C - y_A & y_D - y_C \end{vmatrix}, \qquad \mu = \frac{1}{\Delta}\begin{vmatrix} x_B - x_A & x_C - x_A \\ y_B - y_A & y_C - y_A \end{vmatrix},$$
deci calculul soluțiilor λ și μ ale sistemului (2) revine la evaluarea unor rapoarte de forma $\frac{\textbf{polinom de gradul II}}{\textbf{polinom de gradul II}}$. Mai trebuie verificat faptul că $\lambda, \mu \in [0,1]$.
- În final, prin înlocuirea lui λ și μ în sistemul (2), se găsesc coordonatele punctului de intersecție $\longrightarrow \frac{\textbf{polinom de gradul III}}{\textbf{polinom de gradul II}}$.

Detaliere (scriere în coordonate)

- Cum vrem să extragem obiectul din matriță "în sus", putem presupune f.r.g. că $\vec{d} = (d_x, d_y, 1)$ (de ce?).
- Fie f o față a obiectului, $\vec{\nu}\,(f) = (\nu_x, \nu_y, \nu_z)$. Faptul că fața corespunzătoare $\hat{f}$ a matriței **nu** blochează extragerea în direcția $\vec{d}$ revine la

$$\langle \vec{\nu}\,(f), \vec{d}\,\rangle \leq 0 \Leftrightarrow$$

$$\nu_x \cdot d_x + \nu_y \cdot d_y + \nu_z \leq 0. \qquad (1)$$

- Fixată o față f (informația relevantă fiind ν_x, ν_y, ν_z) căutăm $\vec{d} = (d_x, d_y)$ astfel încât să fie verificată relația (1). Aceasta este o inegalitate care descrie un hiperplan.
- Condițiile indicate mai sus trebuie verificate pentru **toate** fețele!

- Unui punct $p = (p_x, p_y)$ din planul $\mathbb{R}^2$ (plan primal) i se asociază o dreaptă din planul dual, notată p^*:

$$p^* : (y = p_x x - p_y) \text{ duala lui } p.$$

- Unei drepte neverticale $d : (y = m_d x + n_d)$ din planul primal i se asociază un punct din planul dual, notat d^*:

$$d^* = (m_d, -n_d) \text{ dualul lui } d.$$

- **Obs.** Această transformare este polaritatea față de parabola $y = \frac{x^2}{2}$.

Plan primal	**Plan dual**
Punct p	Dreaptă neverticală p^*
Dreaptă neverticală d	Punct d^*
Dreaptă determinată de două puncte	Punct de intersecție a două drepte
Punctul p deasupra dreptei d	Punctul d^* deasupra dreptei p^*
Segment	Fascicul de drepte (*wedge*)

semiplan inferior semiplan superior

- Dat un semiplan delimitat de o dreaptă neverticală

$$ax + by + c \leq 0$$

$-x + y + 3 \leq 0$ semiplan inferior

$x - y - 3 \leq 0$ semiplan superior