

Algoritmi avansați

C1 - Acoperiri convexe

Mihai-Sorin Stupariu

Sem. al II-lea, 2022 - 2023

Introducere

Criterii numerice. Raport și test de orientare

Acoperiri convexe

Introducere

- Algoritmii geometrici sunt legați de Geometria Computațională: *complexitatea computațională a problemelor geometrice în contextul analizei algoritmilor* [Lee & Preparata, 1984]

Introducere

- ▶ Algoritmii geometrici sunt legați de Geometria Computațională: *complexitatea computațională 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?

Introducere

- ▶ Algoritmii geometrici sunt legați de Geometria Computațională: *complexitatea computațională 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

Introducere

- ▶ Algoritmii geometrici sunt legați de Geometria Computațională: *complexitatea computațională 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
- ▶ Probleme abordate: acoperiri convexe, proximitate, intersecții, căutare, etc.

Introducere

- ▶ Algoritmii geometrici sunt legați de Geometria Computațională: *complexitatea computațională 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
- ▶ Probleme abordate: acoperiri convexe, proximitate, intersecții, căutare, etc.
- ▶ Tehnici utilizate: construcții incrementale, divide et impera, plane-sweep, transformări geometrice, etc.

Introducere

- ▶ Note istorice:

Introducere

- ▶ Note istorice:
 - ▶ Ideea de construcție geometrică realizată într-un număr finit de pași:
Elementele lui Euclid

Introducere

▶ Note istorice:

- ▶ Ideea de construcție geometrică realizată într-un număr finit de pași:
[Elementele lui Euclid](#)
- ▶ Abordare numerică - sistem de coordonate în plan: Descartes (sec. XVII)

Introducere

▶ Note istorice:

- ▶ Ideea de construcție geometrică realizată într-un număr finit de pași: [Elementele lui Euclid](#)
- ▶ Abordare numerică - sistem de coordonate în plan: Descartes (sec. XVII)
- ▶ “Simplicitatea” construcțiilor geometrice: Lemoine (~ 1900)

Introducere

- ▶ Note istorice:
 - ▶ Ideea de construcție geometrică realizată într-un număr finit de pași: [Elementele lui Euclid](#)
 - ▶ Abordare numerică - sistem de coordonate în plan: Descartes (sec. XVII)
 - ▶ [“Simplicitatea” construcțiilor geometrice](#): Lemoine (~ 1900)
 - ▶ a doua jumătate a sec. XX: formularea / rezolvarea unor probleme de GC; în 1975 este folosit prima dată termenul de *Computational Geometry* (M.I. Shamos, “Geometric Complexity”, Proc. 7th ACM Annual Symposium on Theory of Computing)

Introducere

- ▶ Note istorice:
 - ▶ Ideea de construcție geometrică realizată într-un număr finit de pași: [Elementele lui Euclid](#)
 - ▶ Abordare numerică - sistem de coordonate în plan: Descartes (sec. XVII)
 - ▶ [“Simplicitatea” construcțiilor geometrice](#): Lemoine (~ 1900)
 - ▶ a doua jumătate a sec. XX: formularea / rezolvarea unor probleme de GC; în 1975 este folosit prima dată termenul de *Computational Geometry* (M.I. Shamos, “Geometric Complexity”, Proc. 7th ACM Annual Symposium on Theory of Computing)
- ▶ Domenii de aplicabilitate: grafică pe calculator, pattern recognition, robotică, statistică, gestionarea bazelor de date numerice, cercetări operaționale

Bibliografie

- ▶ M. de Berg, M. van Kreveld, M. Overmars si O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer, 2008.
(Site: <http://www.cs.uu.nl/geobook/>)
- ▶ F. Preparata si M. Shamos, *Computational Geometry: An Introduction*, Springer, 1985.
- ▶ S. Devadoss, J. O'Rourke, *Discrete and Computational Geometry*, Princeton University Press, 2011.
(Site: <http://cs.smith.edu/~orourke/DCG/>)

Bibliografie

- ▶ M. de Berg, M. van Kreveld, M. Overmars si O. Schwarzkopf, *Computational Geometry, Algorithms and Applications*, Springer, 2008.
(Site: <http://www.cs.uu.nl/geobook/>)
- ▶ F. Preparata si M. Shamos, *Computational Geometry: An Introduction*, Springer, 1985.
- ▶ S. Devadoss, J. O'Rourke, *Discrete and Computational Geometry*, Princeton University Press, 2011.
(Site: <http://cs.smith.edu/~orourke/DCG/>)
- ▶ D. Lee, F. Preparata, *Computational Geometry - A Survey*, IEEE Transactions on Computers, 33 (1984), 1072-1101.
- ▶ B. Gärtner, M. Hoffmann, *Computational Geometry. Lecture Notes*, ETH Zürich, 2013.
- ▶ J. Goodman, J. O'Rourke, C. Tóth (eds.) *Handbook of Discrete and Computational Geometry*, 2017.

Criterii numerice – poziția relativă a unor puncte

- ▶ Stabilirea unor relații între puncte / ordonare

Criterii numerice – poziția relativă a unor puncte

- ▶ Stabilirea unor relații între puncte / ordonare
- ▶ **Context 1D**

Criterii numerice – poziția relativă a unor puncte

- ▶ Stabilirea unor relații între puncte / ordonare
- ▶ **Context 1D**
 - ▶ ordonare (**relativ la un sistem de coordonate**)

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**)

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**

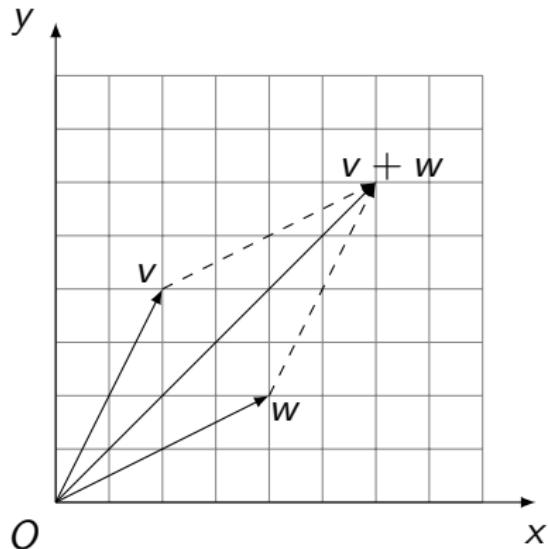
Criterii numerice – poziția relativă a unor puncte

- ▶ 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)

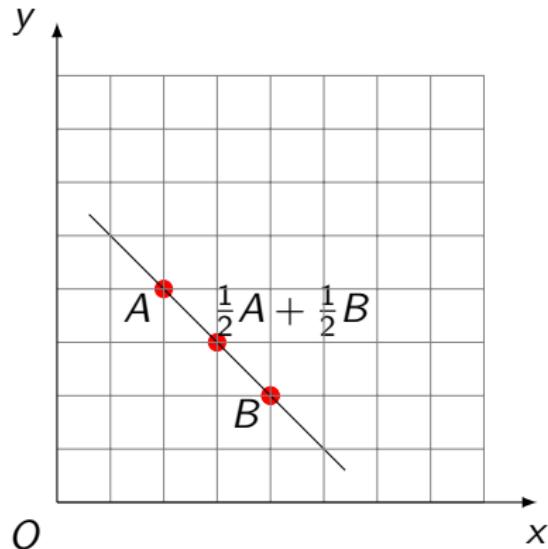
Criterii numerice – poziția relativă a unor puncte

- ▶ 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**)

Vectori și puncte



Combinații liniare
 $\alpha v + \beta w$ ($\alpha, \beta \in \mathbb{R}$)



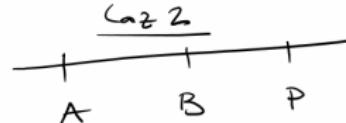
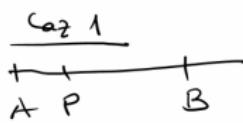
Combinații afine
 $\lambda A + \mu B$ ($\lambda, \mu \in \mathbb{R}$ și $\lambda + \mu = 1$)

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 $P \in AB$.

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 $r(A, P, B)$.

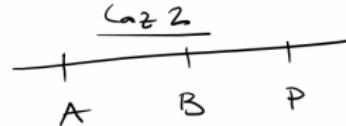
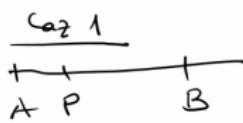


$$\overrightarrow{AP} = r \overrightarrow{PB}, r > 0$$

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

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.

Raport- exemple

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

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

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

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

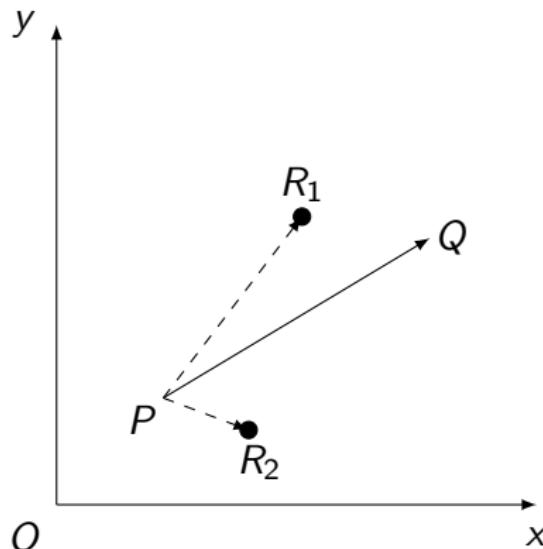
$$\overrightarrow{AB} = \frac{1}{5} \cdot \overrightarrow{BC}$$

$$r(A, B, C)$$

Raport- exemple

- (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 - motivație



Poziția relativă a două puncte față de un vector / o muchie orientată

Enunț principal

- **Propoziție.** Fie $P = (p_1, p_2)$, $Q = (q_1, q_2)$ două puncte distincte din planul \mathbb{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 $\overrightarrow{PQ} \Leftrightarrow \Delta(P, Q, R) > 0$.

Enunț principal

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

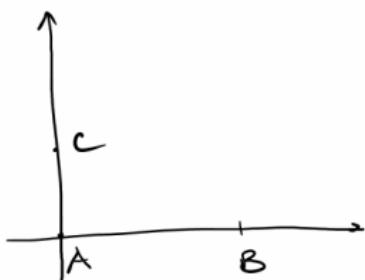
$$\Delta(P, Q, R) = \begin{vmatrix} 1 & 1 & 1 \\ p_1 & q_1 & r_1 \\ p_2 & q_2 & r_2 \end{vmatrix}.$$

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 ($\Delta(P, Q, R)$).

Testul de orientare - exemplu



$$A = (0, 0)$$

$$B = (1, 0)$$

$$C = (0, 1)$$

$$\Delta(A, B, C) = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} = 1 > 0$$

deci cf. criteriului C este în stânga muchiei
orientate \vec{AB}

Limitări - robustețe și erori de rotunjire

L. Kettner et al. / Computational Geometry 40 (2008) 61–78

65

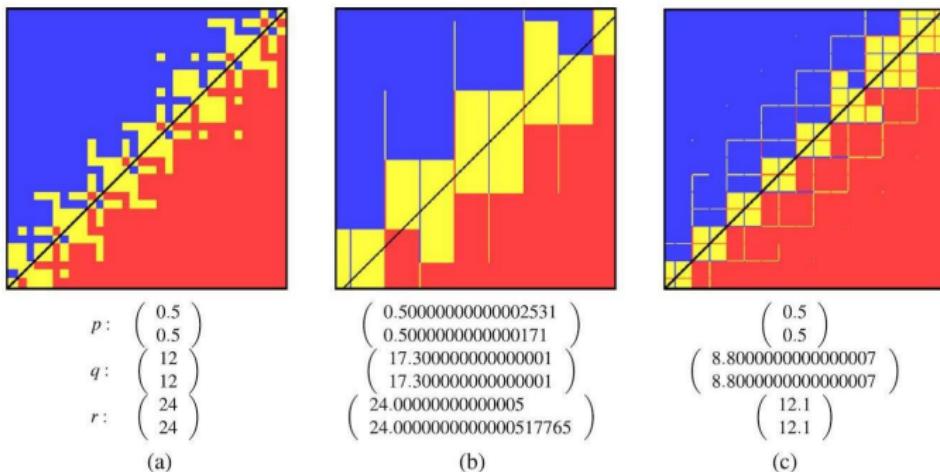
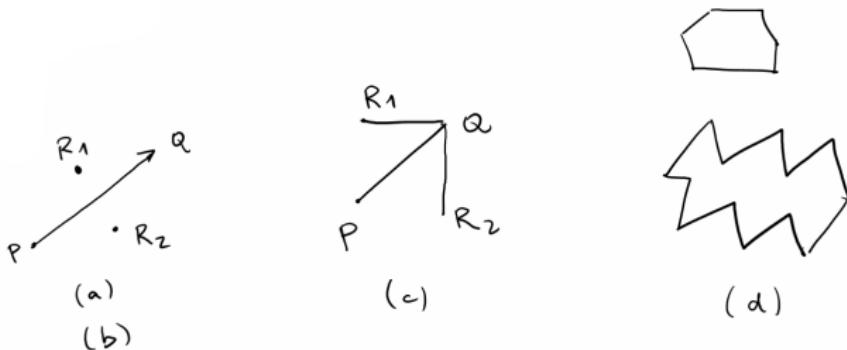


Fig. 2. The weird geometry of the float-orientation predicate: The figure shows the results of `float_orient(px + Xux, py + Yuy, q, r)` for $0 \leq X, Y \leq 255$, where $u_x = u_y = 2^{-53}$ is the increment between adjacent floating-point numbers in the considered range. The result is color coded: Yellow (red, blue, resp.) pixels represent collinear (negative, positive, resp.) orientation. The line through q and r is shown in black.

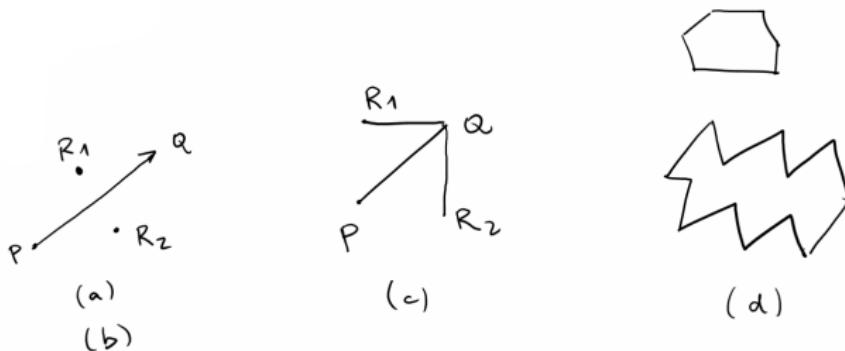
Sursa: Kettner et al, *Classroom examples of robustness problems in geometric computations*, 2008

Aplicații



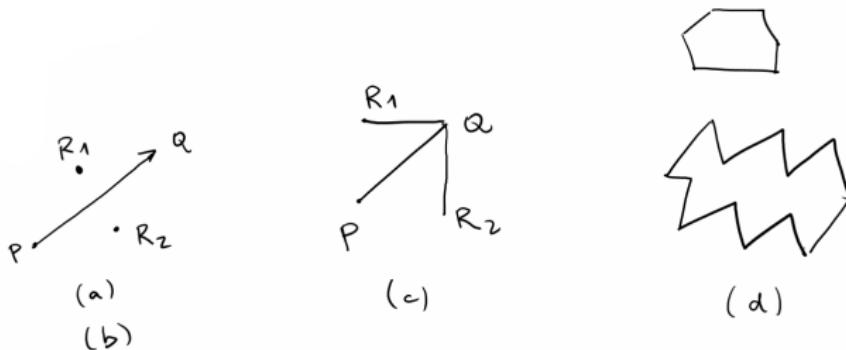
- (a) dacă un punct este în dreapta / stânga unei muchii orientate;
- (b) dacă două puncte sunt de o parte și de alta a unui segment / a unei drepte;

Aplicații



- (a) dacă un punct este în dreapta / stânga unei muchii orientate;
- (b) dacă două puncte sunt de o parte și de alta a unui segment / a unei drepte;
- (c) natura unui viraj în parcurgerea unei linii poligonale (la dreapta / la stânga);

Aplicații



- dacă un punct este în dreapta / stânga unei muchii orientate;
- dacă două puncte sunt de o parte și de alta a unui segment / a unei drepte;
- natura unui viraj în parcurgerea unei linii poligonale (la dreapta / la stânga);
- natura unui poligon (convex / concav).

Mulțimi convexe: generalități

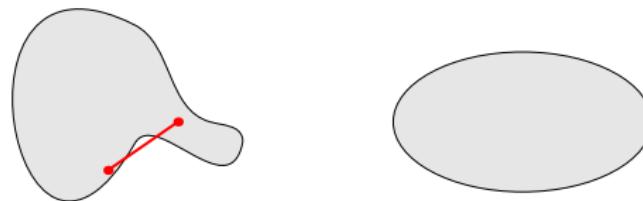
► **Conceptul de mulțime convexă:**

O mulțime $M \subset \mathbb{R}^m$ este convexă dacă oricare ar fi $p, q \in M$, segmentul $[pq]$ este inclus în M .

Mulțimi convexe: generalități

► Conceptul de mulțime convexă:

O mulțime $M \subset \mathbb{R}^m$ este convexă dacă oricare ar fi $p, q \in M$, segmentul $[pq]$ este inclus în M .

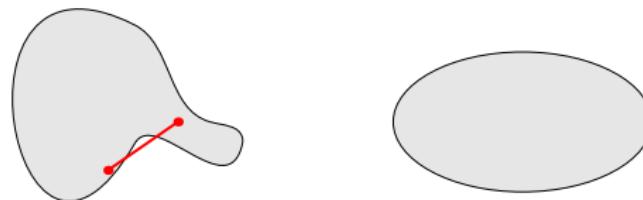


Mulțimea din stânga nu este convexă, întrucât **există** două puncte, pentru care segmentul determinat nu este inclus în mulțime (punctele cu această proprietate nu sunt unice!).

Mulțimi convexe: generalități

► Conceptul de mulțime convexă:

O mulțime $M \subset \mathbb{R}^m$ este convexă dacă oricare ar fi $p, q \in M$, segmentul $[pq]$ este inclus în M .

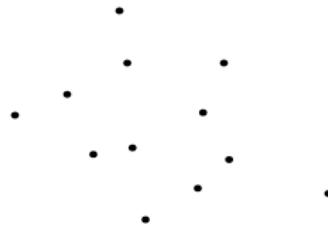


Mulțimea din stânga nu este convexă, întrucât **există** două puncte, pentru care segmentul determinat nu este inclus în mulțime (punctele cu această proprietate nu sunt unice!).

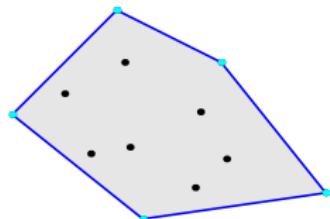
► Problematizare:

Mulțimile finite cu cel puțin două elemente nu sunt convexe \longrightarrow necesară **acoperirea convexă**.

Acoperire convexă a unei multimi (finite) \mathcal{P} : concept

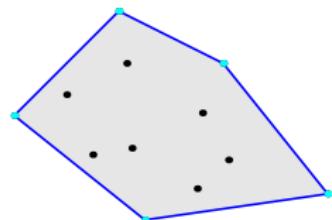


Acoperire convexă a unei multimi finite \mathcal{P} : concept



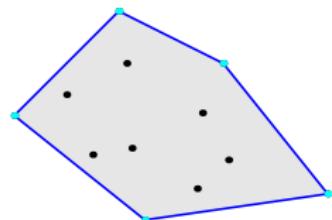
- ▶ Caracterizări echivalente:

Acoperire convexă a unei multimi finite \mathcal{P} : concept



- ▶ Caracterizări echivalente:
 - ▶ Cea mai "mică" (în sensul incluziunii) mulțime convexă care conține \mathcal{P} .

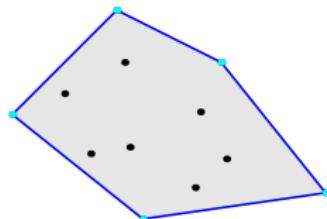
Acoperire convexă a unei multimi finite \mathcal{P} : concept



► Caracterizări echivalente:

- ▶ Cea mai "mică" (în sensul incluziunii) mulțime convexă care conține \mathcal{P} .
- ▶ Intersecția tuturor mulțimilor convexe care conțin \mathcal{P} .

Acoperire convexă a unei multimi finite \mathcal{P} : concept

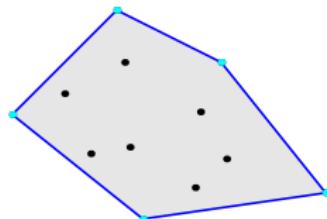


► Caracterizări echivalente:

- ▶ Cea mai "mică" (în sensul incluziunii) mulțime convexă care conține \mathcal{P} .
- ▶ Intersecția tuturor mulțimilor convexe care conțin \mathcal{P} .
- ▶ Mulțimea tuturor combinațiilor convexe ale punctelor din \mathcal{P} . O **combinație convexă** a punctelor P_1, P_2, \dots, P_n este un punct P de forma

$$P = \alpha_1 P_1 + \dots + \alpha_n P_n, \quad \alpha_1, \dots, \alpha_n \in [0, 1], \quad \alpha_1 + \dots + \alpha_n = 1.$$

Acoperire convexă a unei multimi finite \mathcal{P} : concept



- ▶ Caracterizări echivalente:
 - ▶ Cea mai "mică" (în sensul incluziunii) mulțime convexă care conține \mathcal{P} .
 - ▶ Intersecția tuturor mulțimilor convexe care conțin \mathcal{P} .
 - ▶ Mulțimea tuturor combinațiilor convexe ale punctelor din \mathcal{P} . O **combinație convexă** a punctelor P_1, P_2, \dots, P_n este un punct P de forma

$$P = \alpha_1 P_1 + \dots + \alpha_n P_n, \quad \alpha_1, \dots, \alpha_n \in [0, 1], \quad \alpha_1 + \dots + \alpha_n = 1.$$

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

Acoperire convexă a unei multimi finite \mathcal{P} : problematizare

- Dacă $\mathcal{P} \subset \mathbb{R}^d$ este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un **politop convex**.

Acoperire convexă a unei multimi finite \mathcal{P} : problematizare

- ▶ Dacă $\mathcal{P} \subset \mathbb{R}^d$ este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un **politop convex**.
- ▶ Cazuri particulare: $d = 1$ (segment); $d = 2$ (poligon); $d = 3$ (poliedru).

Acoperire convexă a unei multimi finite \mathcal{P} : problematizare

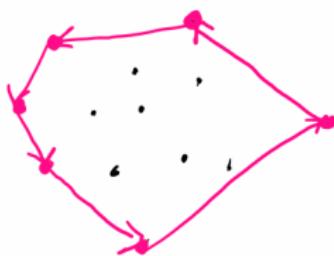
- ▶ Dacă $\mathcal{P} \subset \mathbb{R}^d$ este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un **politop convex**.
- ▶ Cazuri particulare: $d = 1$ (segment); $d = 2$ (poligon); $d = 3$ (poliedru).
- ▶ Cazul $d = 1$: acoperirea convexă este un segment; algoritmic: parcursere a punctelor (complexitate $O(n)$).

Acoperire convexă a unei multimi finite \mathcal{P} : problematizare

- ▶ Dacă $\mathcal{P} \subset \mathbb{R}^d$ este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un **politop convex**.
- ▶ Cazuri particulare: $d = 1$ (segment); $d = 2$ (poligon); $d = 3$ (poliedru).
- ▶ Cazul $d = 1$: acoperirea convexă este un segment; algoritmic: parcursere a punctelor (complexitate $O(n)$).
- ▶ În continuare: $d = 2$.

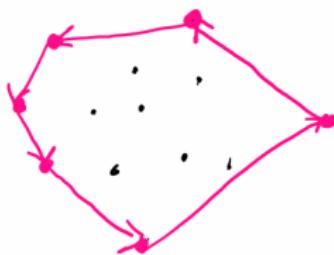
Acoperire convexă a unei multimi finite \mathcal{P} (practic)

- De fapt, dacă \mathcal{P} este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un poligon convex.



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

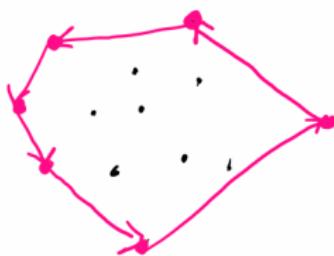
- De fapt, dacă \mathcal{P} este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un poligon convex.



- **Problemă:**
Cum determinăm, algoritmice, vîrfurile acestui poligon?

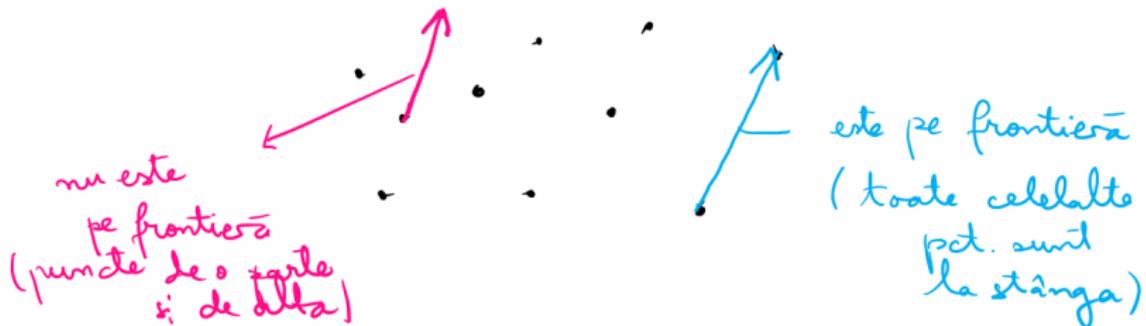
Acoperire convexă a unei multimi finite \mathcal{P} (practic)

- De fapt, dacă \mathcal{P} este finită, acoperirea sa convexă, $\text{Conv}(\mathcal{P})$ este un poligon convex.



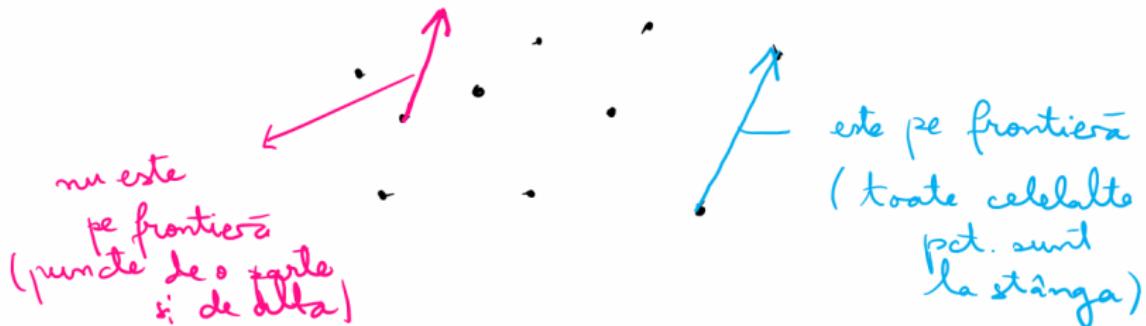
- **Problemă:**
Cum determinăm, algoritmice, vîrfurile acestui poligon?
- **Convenție:** Sensul de parcursere a frontierei este cel trigonometric.

Un algoritm "lent": idee de lucru



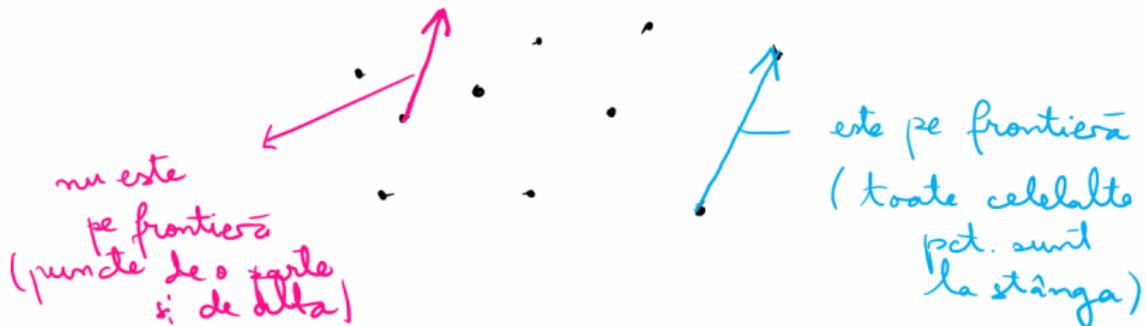
- Sunt considerate **muchiile orientate**.

Un algoritm "lent": idee de lucru



- ▶ Sunt considerate **muchiile orientate**.
- ▶ **Q:** Cum se decide dacă o muchie orientată fixată este pe **frontieră**?

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").

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$
4. **for** $R \in \mathcal{P} \setminus \{P, Q\}$

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$
4. **for** $R \in \mathcal{P} \setminus \{P, Q\}$
5. **do if** R "în dreapta" lui \overrightarrow{PQ}

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$
4. **for** $R \in \mathcal{P} \setminus \{P, Q\}$
5. **do if** R "în dreapta" lui \overrightarrow{PQ}
6. **then** $valid \leftarrow \text{false}$

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$
4. **for** $R \in \mathcal{P} \setminus \{P, Q\}$
5. **do if** R "în dreapta" lui \overrightarrow{PQ}
6. **then** $valid \leftarrow \text{false}$
7. **if** $valid = \text{true}$ **then** $E = E \cup \{\overrightarrow{PQ}\}$

Un algoritm lent

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sensul trigonometric.

1. $E \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset$ /* E este lista muchiilor orientate*/
2. **for** $(P, Q) \in \mathcal{P} \times \mathcal{P}$ cu $P \neq Q$
3. **do** $valid \leftarrow \text{true}$
4. **for** $R \in \mathcal{P} \setminus \{P, Q\}$
5. **do if** R "în dreapta" lui \overrightarrow{PQ}
6. **then** $valid \leftarrow \text{false}$
7. **if** $valid = \text{true}$ **then** $E = E \cup \{\overrightarrow{PQ}\}$
8. din E se construiește lista \mathcal{L} a vârfurilor acoperirii convexe /*este necesar ca E să fie **coerentă***/

Algoritmul "lent": comentarii

- Complexitatea: $O(n^3)$

Algoritmul "lent": comentarii

- ▶ Complexitatea: $O(n^3)$
- ▶ Complexitate algebrică: polinoame de gradul II

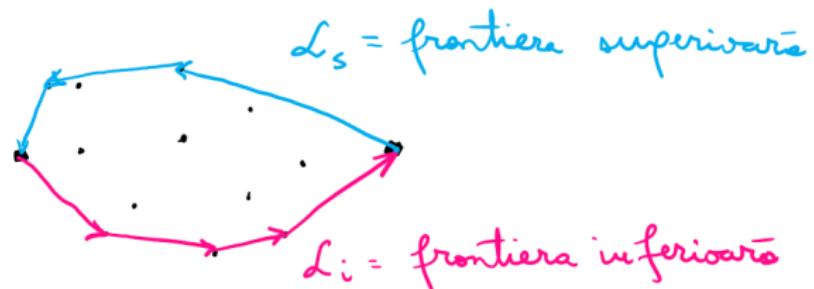
Algoritmul "lent": comentarii

- ▶ Complexitatea: $O(n^3)$
- ▶ Complexitate algebrică: polinoame de gradul II
- ▶ Tratarea cazurilor degenerate: poate fi adaptat.

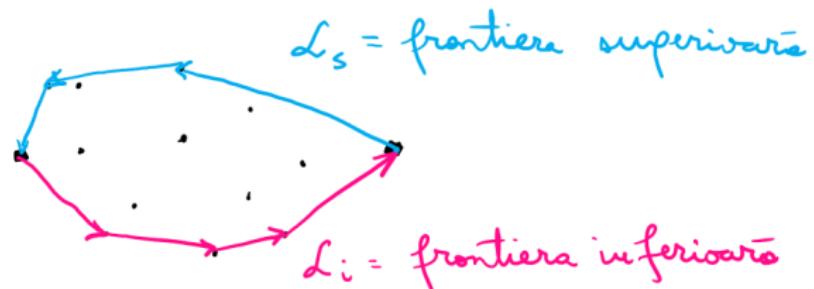
Algoritmul "lent": comentarii

- ▶ Complexitatea: $O(n^3)$
- ▶ Complexitate algebrică: polinoame de gradul II
- ▶ Tratarea cazurilor degenerate: poate fi adaptat.
- ▶ Robustețea: datorită erorilor de rotunjire este posibil ca algoritmul să nu returneze o listă coerentă de muchii.

Graham's scan, varianta Andrew: idee de lucru

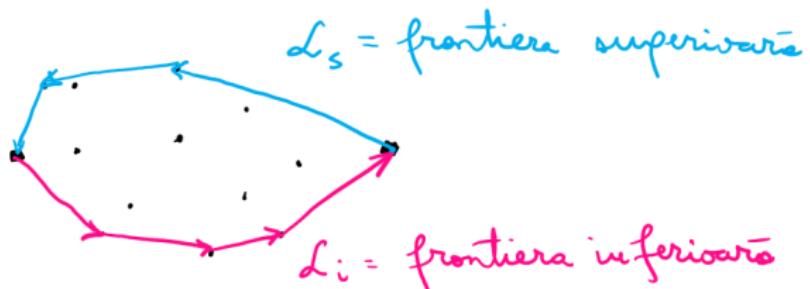


Graham's scan, varianta Andrew: idee de lucru



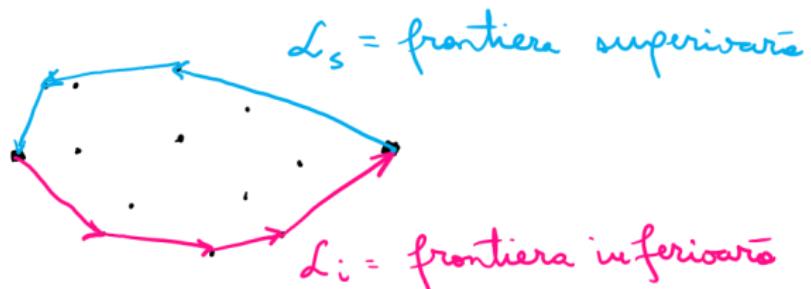
- ▶ Punctele sunt mai întâi sortate și renumerate **lexicografic**.

Graham's scan, varianta Andrew: idee de lucru



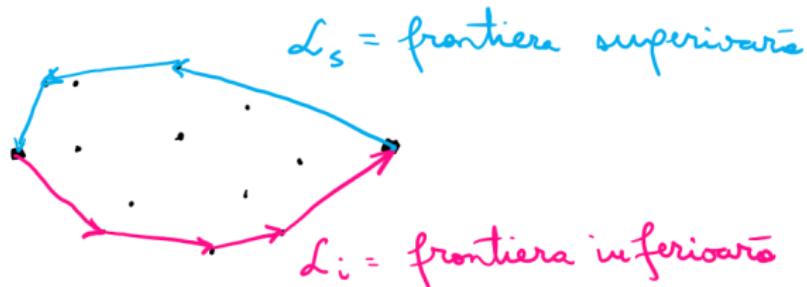
- ▶ Punctele sunt mai întâi sortate și renumerate **lexicografic**.
- ▶ Algoritmul este de tip **incremental**, punctele fiind adăugate unul câte unul, fiind apoi eliminate anumite puncte.

Graham's scan, varianta Andrew: idee de lucru

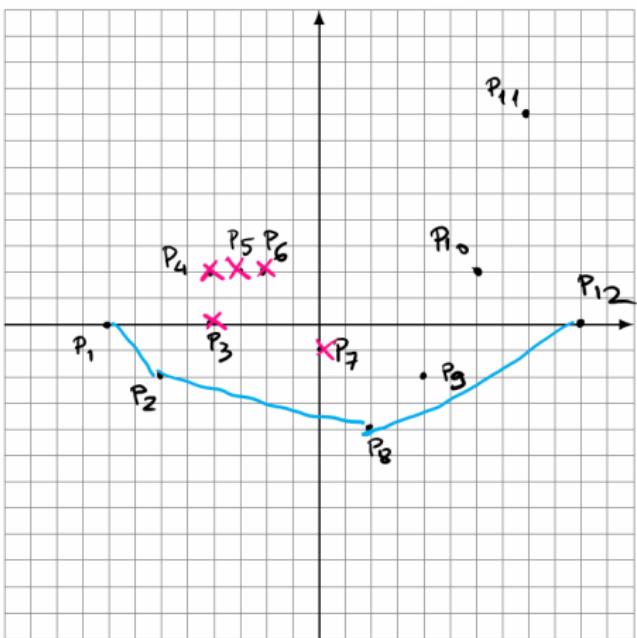


- ▶ Punctele sunt mai întâi sortate și renumerate **lexicografic**.
- ▶ Algoritmul este de tip **incremental**, punctele fiind adăugate unul câte unul, fiind apoi eliminate anumite puncte.
- ▶ **Q:** Cum se decide dacă trei puncte sunt vârfuri consecutive ale acoperirii convexe?

Graham's scan, varianta Andrew: idee de lucru



- ▶ Punctele sunt mai întâi sortate și renumerate **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.



$P_1 = (-8, 0)$; $P_2 = (-6, -2)$; $P_3 = (-4, 0)$;
 $P_4 = (-4, 2)$; $P_5 = (-3, 2)$; $P_6 = (-2, 2)$;
 $P_7 = (0, -1)$; $P_8 = (2, -4)$; $P_9 = (4, -2)$
 $P_{10} = (6, 2)$; $P_{11} = (8, 8)$; $P_{12} = (10, 0)$

Cum evoluază L_i pe parcursul

Graham's scan - varianta Andrew.

P_1, P_2

P_1, P_2, P_3

P_1, P_2, P_3, P_4, P_5

viraj la dreapta
în P_4

P_1, P_2, P_3, P_5, P_6

viraj la stânga în P_5

P_2, P_3, P_6 coliniare

P_1, P_2, P_6, P_7

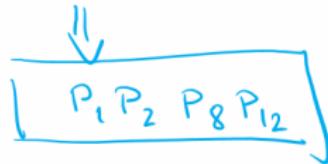
viraj dr. P_6

P_1, P_2, P_7, P_8

viraj dr. P_7

$P_1, P_2, P_8, P_9, P_{10}, P_{11}, P_{12}$

viraj dr.
 P_{11}, P_{10}, P_9



frontiera
inferioară

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Sortare lexicografică, renumerotare P_1, P_2, \dots, P_n conform ordonării

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Sortare lexicografică, renumerotare P_1, P_2, \dots, P_n conform ordonării
2. $\mathcal{L} \leftarrow (P_1, P_2)$

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Sortare lexicografică, renumerotare P_1, P_2, \dots, P_n conform ordonării
2. $\mathcal{L} \leftarrow (P_1, P_2)$
3. **for** $i \leftarrow 3$ **to** n

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{R}^2 .

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Sortare lexicografică, renumerotare P_1, P_2, \dots, 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}

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{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, \dots, 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 nu determină un viraj la stânga

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{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, \dots, 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 nu determină un viraj la stânga
 6. **do** șterge penultimul punct

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{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, \dots, 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 nu determină un viraj la stânga
6. **do** șterge penultimul punct
7. **return** \mathcal{L}_i

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{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, \dots, 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 nu 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

Graham's scan, varianta Andrew (algoritm)

Input: O mulțime de puncte \mathcal{P} din \mathbb{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, \dots, 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 nu 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

Graham's scan, varianta Andrew: comentarii

- ▶ Complexitatea: $O(n \log n)$.

Graham's scan, varianta Andrew: comentarii

- ▶ Complexitatea: $O(n \log n)$.
- ▶ Tratarea cazurilor degenerate: corect.

Graham's scan, varianta Andrew: comentarii

- ▶ 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 / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.

Jarvis' march / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.
- ▶ Inițializare: un punct care este sigur un vârf al acoperirii convexe (e.g. punctul cel mai de jos / din stânga / stânga jos).

Jarvis' march / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.
- ▶ Inițializare: 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.

Jarvis' march / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.
- ▶ Inițializare: 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.

Jarvis' march / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.
- ▶ Inițializare: 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 frontieră acoperirii convexe.

Jarvis' march / Jarvis' wrap [1973]

- ▶ Algoritm de tip **incremental**. Nu necesită sortare prealabilă.
- ▶ Inițializare: 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 frontieră acoperirii convexe.
- ▶ **Algoritmul lui Chan** "combină" ideile celor doi algoritmi, ajungând la complexitatea-timp $O(n \log h)$.

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate $A_k S$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate $A_k S$
7. **then** $S \leftarrow P_i$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate A_kS
7. **then** $S \leftarrow P_i$
8. **if** $S \neq A_1$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate $A_k S$
7. **then** $S \leftarrow P_i$
8. **if** $S \neq A_1$
9. **then** $k \leftarrow k + 1$;
 $A_k = S$
adaugă A_k la \mathcal{L}

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate $A_k S$
7. **then** $S \leftarrow P_i$
8. **if** $S \neq A_1$
9. **then** $k \leftarrow k + 1$;
 $A_k = S$
 adaugă A_k la \mathcal{L}
10. **else** $valid \leftarrow \text{false}$

Jarvis' march (algoritm)

Input: O mulțime de puncte necoliniare $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ din \mathbb{R}^2 ($n \geq 3$).

Output: O listă \mathcal{L} care conține vârfurile ce determină frontieră acoperirii convexe, parcursă în sens trigonometric.

1. Determinarea unui punct din \mathcal{P} care aparține frontierei (de exemplu cel mai mic, folosind ordinea lexicografică); acest punct este notat cu A_1 .
2. $k \leftarrow 1$; $\mathcal{L} \leftarrow (A_1)$; $valid \leftarrow \text{true}$
3. **while** $valid = \text{true}$
4. **do** alege un pivot arbitrar $S \in \mathcal{P}$, diferit de A_k
5. **for** $i \leftarrow 1$ **to** n
6. **do if** P_i este la dreapta muchiei orientate $A_k S$
7. **then** $S \leftarrow P_i$
8. **if** $S \neq A_1$
9. **then** $k \leftarrow k + 1$;
 $A_k = S$
 adaugă A_k la \mathcal{L}
10. **else** $valid \leftarrow \text{false}$
11. **return** \mathcal{L}

Alte direcții de lucru

- ▶ **Aplicații:** grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.

Alte direcții de lucru

- ▶ **Aplicații:** grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.
- ▶ Pot fi stabilite legături cu algoritmi studiați 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 - [exemplu](#).

Alte direcții de lucru

- ▶ **Aplicații:** grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.
- ▶ Pot fi stabilite legături cu algoritmi studiați 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 - [exemplu](#).
- ▶ Algoritmi pentru spații euclidiene de dimensiune $m \geq 3$.

Alte direcții de lucru

- ▶ **Aplicații:** grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.
- ▶ Pot fi stabilite legături cu algoritmi studiați 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 - [exemplu](#).
- ▶ Algoritmi pentru spații euclidiene de dimensiune $m \geq 3$.
- ▶ Algoritmi eficienți pentru determinarea acoperirii convexe pentru vârfurile unui poligon arbitrar.

Alte direcții de lucru

- ▶ **Aplicații:** grafică pe calculator, robotică, GIS, recunoașterea formelor, gestionarea bazelor de date multi-dimensionale, etc.
- ▶ Pot fi stabilite legături cu algoritmi studiați 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 - [exemplu](#).
- ▶ Algoritmi pentru spații euclidiene de dimensiune $m \geq 3$.
- ▶ Algoritmi eficienți pentru determinarea acoperirii convexe pentru vârfurile unui poligon arbitrar.
- ▶ Algoritmi dinamici (on-line, real-time, convex hull maintenance).

Algoritmi avansați

C2 - Triangularea poligoanelor

Mihai-Sorin Stupariu

Sem. al II-lea, 2022 - 2023

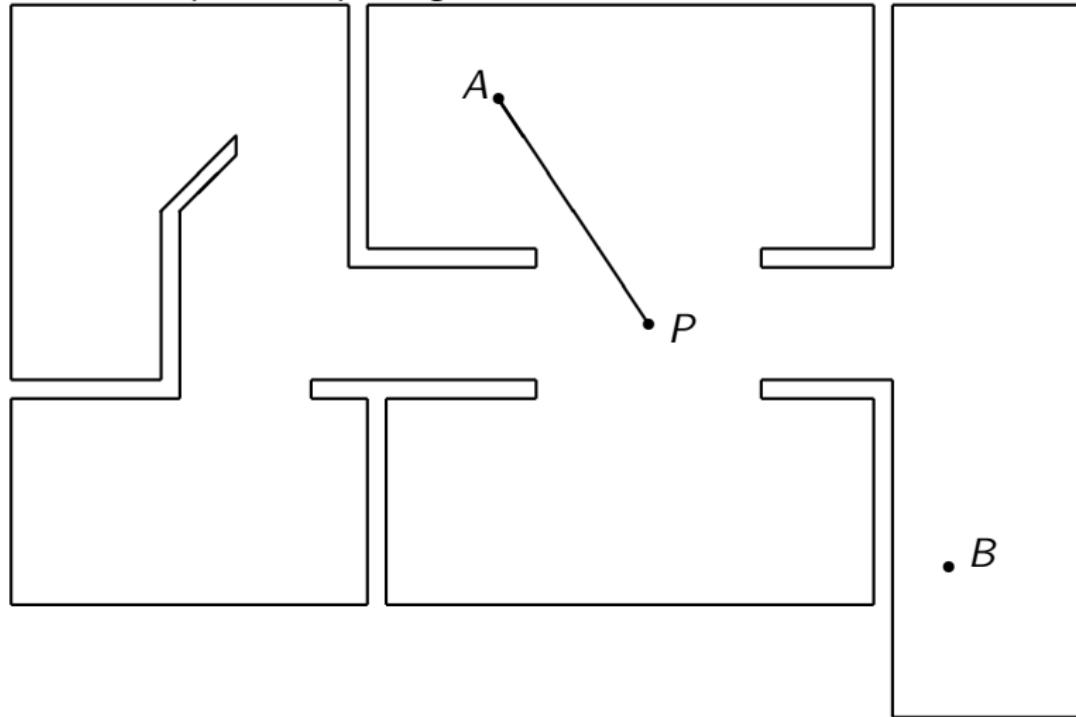
Problema galeriei de artă

Algoritmi de triangulare - “Ear clipping”

Triangularea poligoanelor - un algoritm eficient

Supravegherea unei galerii de artă

Camera din P poate supraveghea A , dar nu B .



Formalizare

- ▶ O galerie de artă poate fi interpretată (în contextul acestei probleme) ca un poligon \mathcal{P} (adică o linie poligonală fără autointersecții) având n vârfuri.

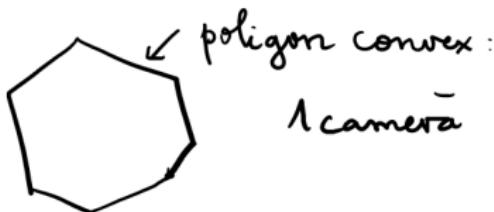
Formalizare

- ▶ O galerie de artă poate fi interpretată (în contextul acestei probleme) ca un poligon \mathcal{P} (adică o linie poligonală fără autointersecții) având n vârfuri.
- ▶ O cameră video (vizibilitate 360^0) poate fi identificată cu un punct din interiorul lui \mathcal{P} ; ea poate supraveghea acele puncte cu care poate fi unită printr-un segment inclus în interiorul poligonului.

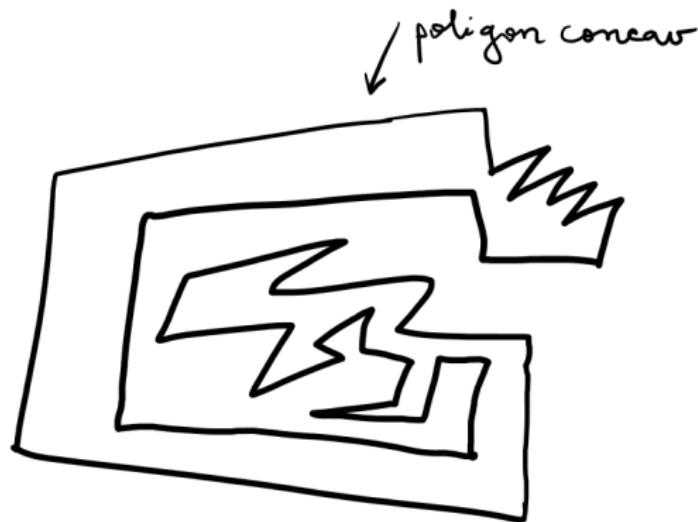
Formalizare

- ▶ O galerie de artă poate fi interpretată (în contextul acestei probleme) ca un poligon \mathcal{P} (adică o linie poligonală fără autointersecții) având n vârfuri.
- ▶ O cameră video (vizibilitate 360^0) poate fi identificată cu un punct din interiorul lui \mathcal{P} ; ea poate supraveghea acele puncte cu care poate fi unită printr-un segment inclus în interiorul poligonului.
- ▶ **Problema galeriei de artă:** *câte camere video sunt necesare pentru a supraveghea o galerie de artă și unde trebuie amplasate acestea?*

Comentarii



1 cameră



Numărul de camere vs. forma poligonului

- ▶ Se dorește exprimarea numărului de camere necesare pentru supraveghere în funcție de n (sau controlarea acestuia de către n).

Numărul 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 spațiu având forma unui poligon convex, este suficientă o singură cameră.

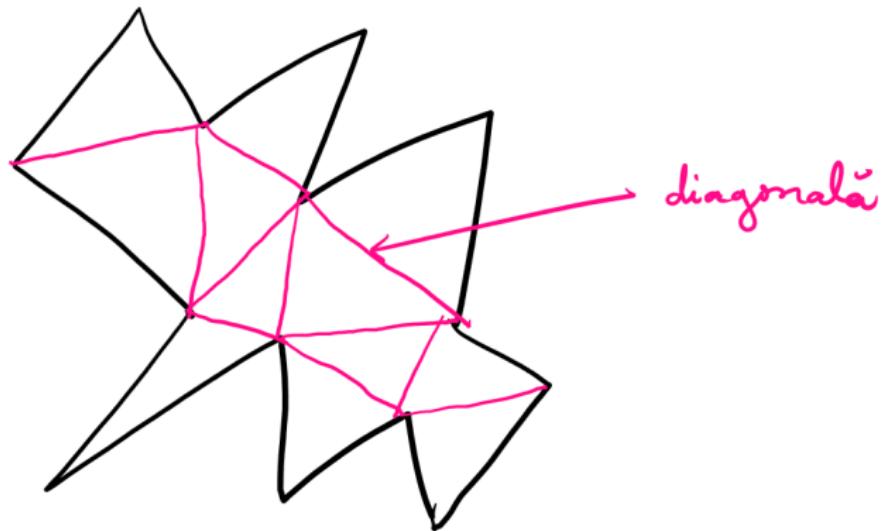
Numărul de camere vs. forma poligonului

- ▶ Se dorește exprimarea numărului de camere necesare pentru supraveghere în funcție de n (sau controlarea acestuia de către n).
- ▶ Pentru a supraveghea un spațiu 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.

Numărul 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 spațiu 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).

Despre triangulări



Definiție formală

- ▶ Fie \mathcal{P} un poligon plan.

Definiție formală

- ▶ 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} .

Definiție formală

- ▶ 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** $T_{\mathcal{P}}$ a lui \mathcal{P} este o descompunere a lui \mathcal{P} în triunghiuri, dată de o mulțime maximală de diagonale ce nu se intersectează.

Rezultate

- **Lemă.** Orice poligon admite o diagonală.

Rezultate

- ▶ **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.*

Rezolvarea problemei galeriei de artă

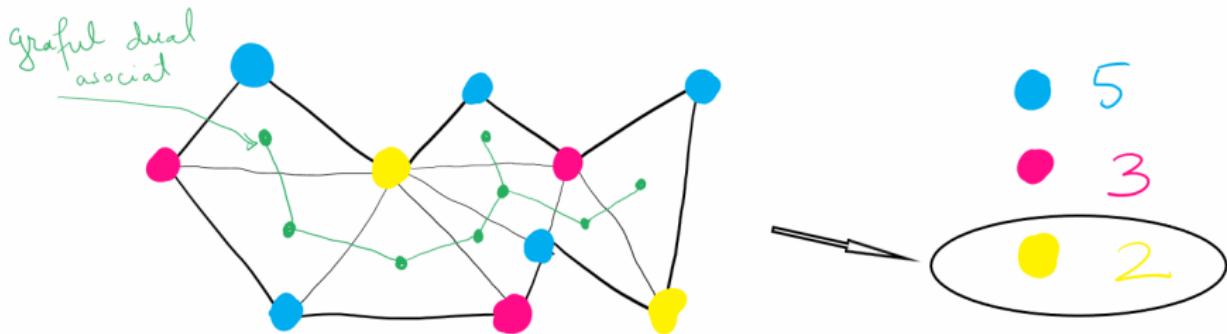
- ▶ Amplasarea camerelor se poate face în vîrfurile poligonului.

Rezolvarea problemei galeriei de artă

- ▶ Amplasarea camerelor se poate face în vîrfurile poligonului.
- ▶ Date o pereche $(\mathcal{P}, \mathcal{T}_P)$ se consideră o 3-colorare a acesteia: fiecărui vîrf îi corespunde o culoare dintr-un set de 3 culori și pentru fiecare triunghi, cele 3 vîrfuri au culori distincte.

Rezolvarea problemei galeriei de artă

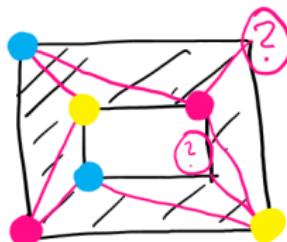
- ▶ Amplasarea camerelor se poate face în vârfurile poligonului.
- ▶ Date o pereche $(\mathcal{P}, \mathcal{T}_P)$ se consideră o 3-colorare a acesteia: fiecărui vârf îi corespunde o culoare dintr-un set de 3 culori și pentru fiecare triunghi, cele 3 vârfuri au culori distincte.
- ▶ **Observație.** Dacă \mathcal{P} este linie poligonală fără autointersecții o astfel de colorare există, deoarece graful dual asociat perechii $(\mathcal{P}, \mathcal{T}_P)$ este arbore.



Rezolvarea problemei galeriei de artă

- ▶ Amplasarea camerelor se poate face în vârfurile poligonului.
- ▶ Date o pereche $(\mathcal{P}, \mathcal{T}_P)$ se consideră o 3-colorare a acesteia: fiecărui vârf îi corespunde o culoare dintr-un set de 3 culori și pentru fiecare triunghi, cele 3 vârfuri au culori distincte.
- ▶ **Observație.** Dacă \mathcal{P} este linie poligonală fără autointersecții o astfel de colorare există, deoarece graful dual asociat perechii $(\mathcal{P}, \mathcal{T}_P)$ este arbore.

Contraexemplu - 3 colorare



Teorema galeriei de artă

- ▶ **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.*

Teorema galeriei de artă

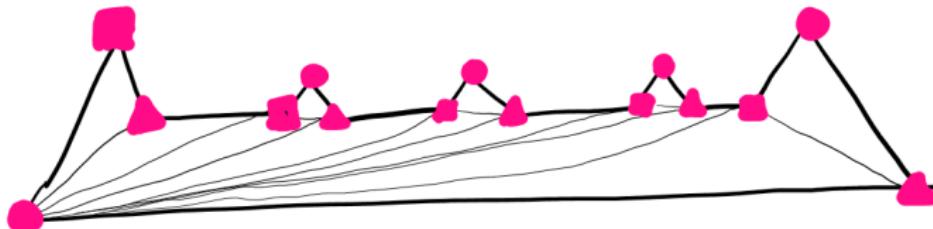
- ▶ **Teoremă.** [Chvátal, 1975; Fisk, 1978] Pentru un poligon cu n vârfuri, $\left[\frac{n}{3}\right]$ camere sunt **uneori necesare** și întotdeauna **suficiente** pentru ca fiecare punct al poligonului să fie vizibil din cel puțin una din camere.
- ▶ Despre Teorema Galeriei de Artă: J. O'Rourke, *Art Gallery Theorems and Algorithms*

Teorema galeriei de artă

- ▶ **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.
- ▶ Despre Teorema Galeriei de Artă: J. O'Rourke, *Art Gallery Theorems and Algorithms*
- ▶ Despre numărul de culori utilizat: L. Erickson, S. LaValle, *A chromatic art gallery problem*

Teorema galeriei de artă - justificare, exemplu

• unori necesare



- 5
- 5
- ▲ 5

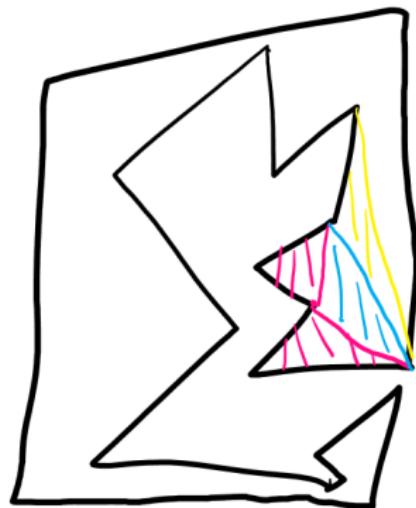
Teorema galeriei de artă - justificare, exemplu

- întotdeauna suficiente

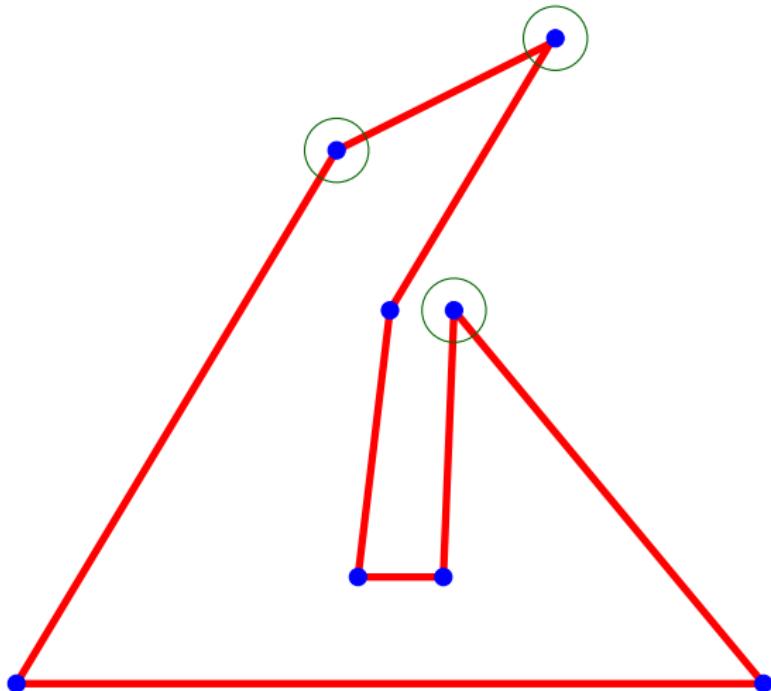
notăm cu n_1, n_2, n_3 numărul de vârfuri colorate cu cele 3 culori : $n_1 + n_2 + n_3 = n$

$$\begin{aligned} \text{Pp. abs. } n_1 > \left[\frac{n}{3} \right] \Rightarrow n_1 > \frac{n}{3} &\quad \left(\begin{array}{l} \text{def.} \\ \text{partii} \\ \text{întregi} \end{array} \right) \\ n_2 > \left[\frac{n}{3} \right] \Rightarrow n_2 > \frac{n}{3} & \Rightarrow n_1 + n_2 + n_3 \\ n_3 > \left[\frac{n}{3} \right] \Rightarrow n_3 > \frac{n}{3} & > n \\ \Rightarrow \exists i \text{ a.i. } n_i \leq \left[\frac{n}{3} \right] & \text{contradicție} \end{aligned}$$

Triangularea unui poligon - intuiție

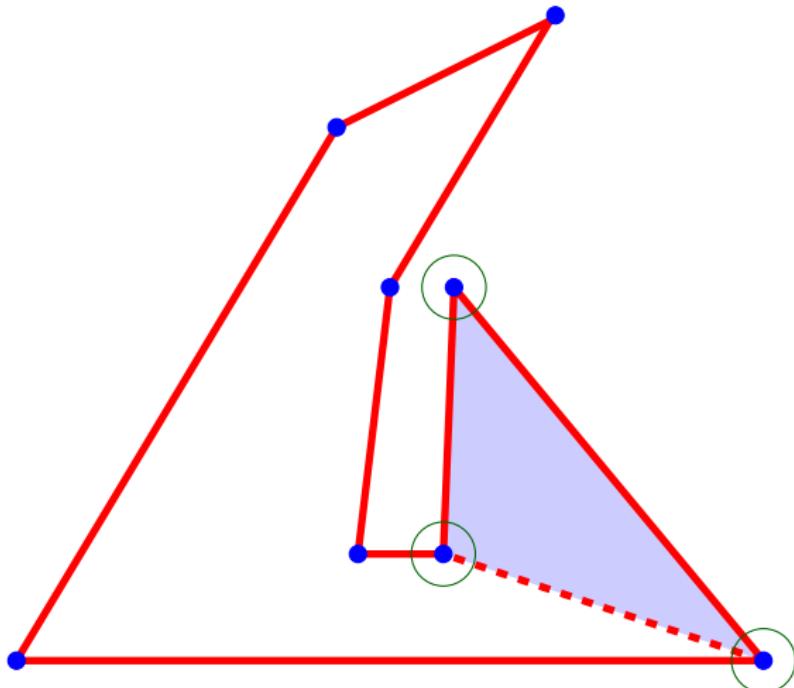


Triangularea unui poligon - algoritmul “Ear clipping”



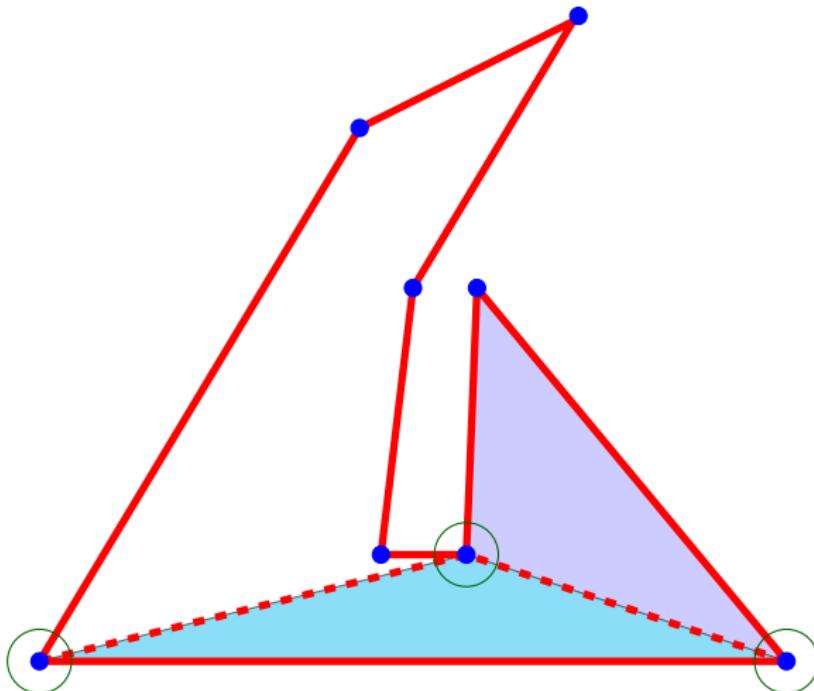
Vârfuri care pot fi selectate pentru start.

Triangularea unui poligon - algoritmul “Ear clipping”



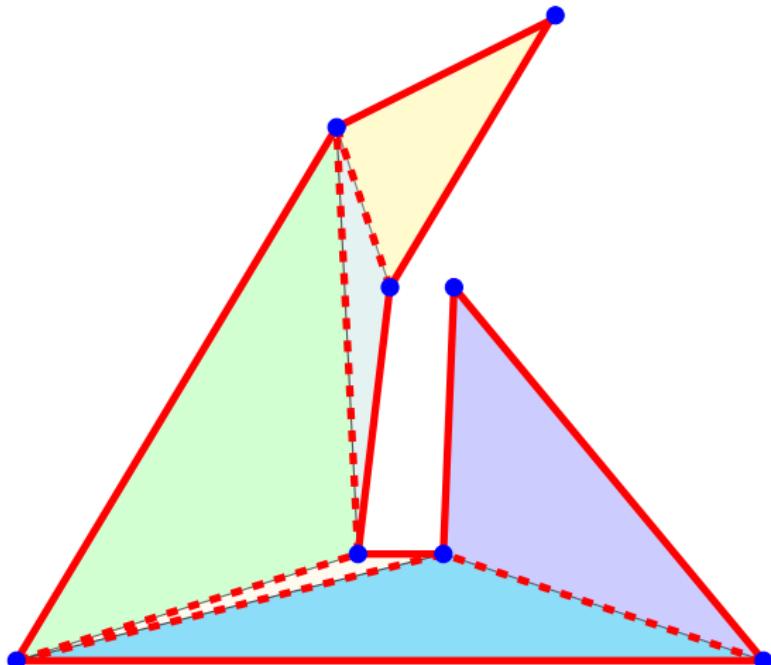
Ales un vârf, este considerat triunghiul determinat cu predecesorul și succesorul.

Triangularea unui poligon - algoritmul “Ear clipping”



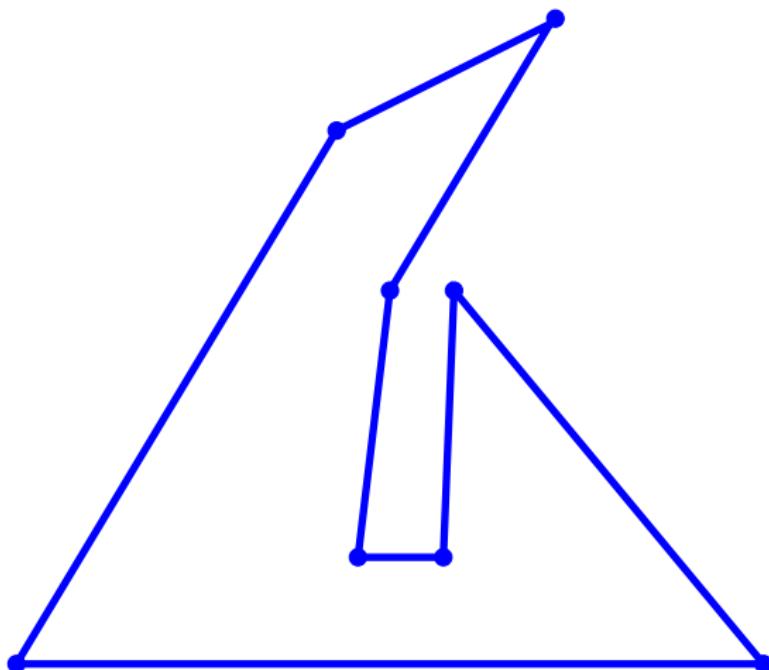
Procedura continuă....

Triangularea unui poligon - algoritmul “Ear clipping”



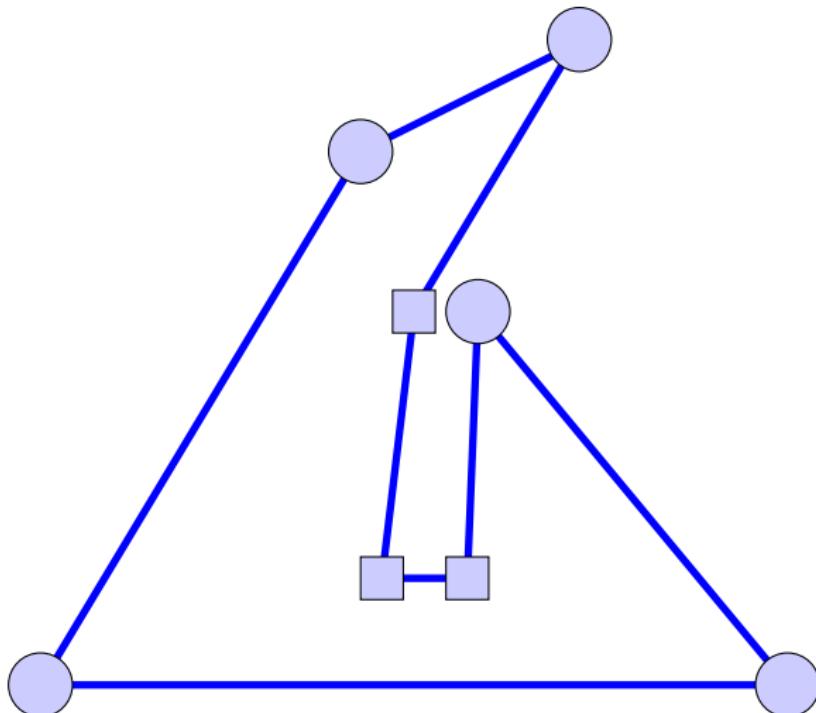
... se obține o triangulare a poligonului.

Clasificarea vârfurilor unui poligon - convexe/concave



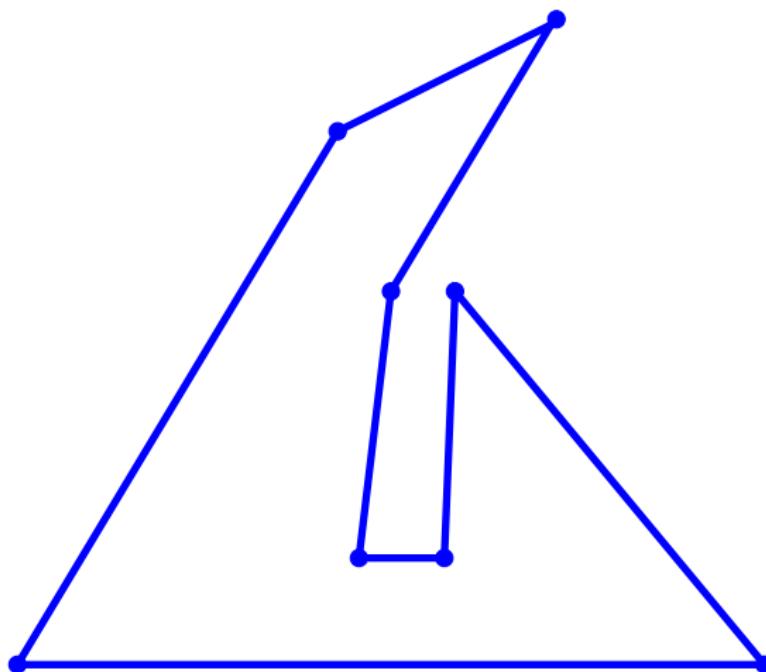
Vârfurile convexe / concave.

Clasificarea vârfurilor unui poligon - convexe/concave



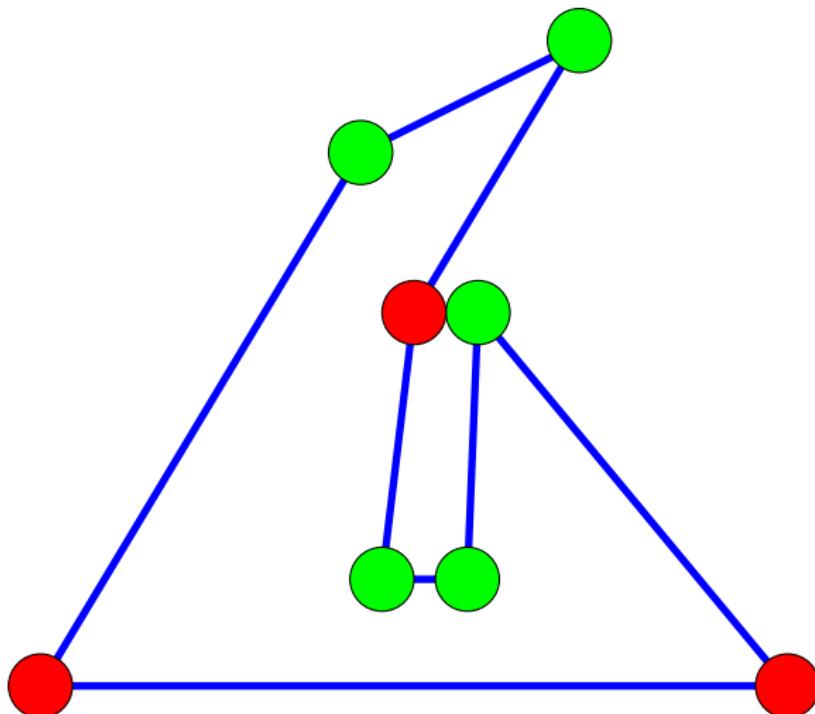
Vârfurile convexe (cerc) / concave (pătrat).

Clasificarea vârfurilor unui poligon - principale/neprincipale



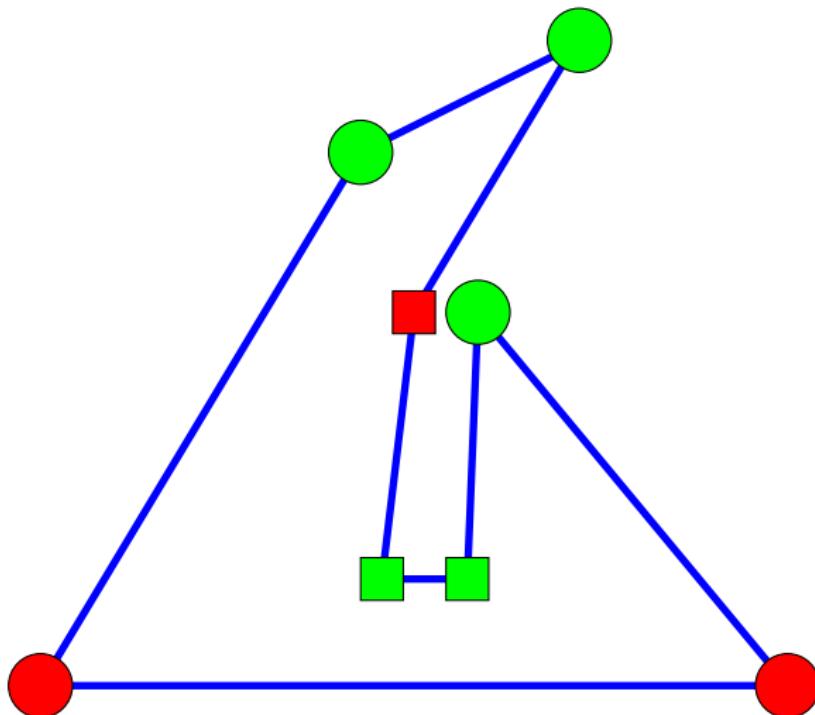
Vârfurile principale.

Clasificarea vârfurilor unui poligon - principale/neprincipale



Vârfurile principale (verde) / neprincipale (roșu).

Clasificarea vârfurilor unui poligon



Patru tipuri de vârfuri.

Metode de triangulare: ear cutting / clipping / trimming

- Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:

Metode de triangulare: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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 $\Delta P_{i-1}P_iP_{i+1}$).

Metode de triangulare: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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].

Metode de triangulare: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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.

Metode de triangulare: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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.*

Metode de triangulare: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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: ear cutting / clipping / trimming

- ▶ Concepte pentru un poligon $\mathcal{P} = (P_1, P_2, \dots, P_n)$:
 - **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 $O(n^2)$.
- ▶ Link despre triangulări. [Link pentru algoritmul Ear cutting](#)

Metode de triangulare: descompunerea în poligoane monotone

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

Metode de triangulare: descompunerea în poligoane monotone

- ▶ Algoritmi de triangulare eficienți: complexitate $O(n)$ pentru poligoane y -monotone [Garey et al., 1978] (algoritmul este descris pe slide-urile următoare).
 - ▶ Descompunerea unui poligon oarecare în 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 $O(n \log n)$.*

Metode de triangulare: descompunerea în poligoane monotone

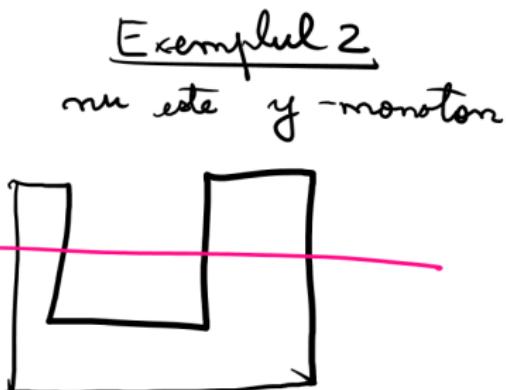
- ▶ Algoritmi de triangulare eficienți: complexitate $O(n)$ pentru poligoane y -monotone [Garey et al., 1978] (algoritmul este descris pe slide-urile următoare).
- ▶ Descompunerea unui poligon oarecare în componente y -monotone poate fi realizată cu un algoritm de complexitate $O(n \log n)$ [Lee, Preparata, 1977]. În concluzie, avem următoarea
Teoremă. *Un poligon poate fi triangulat folosind un algoritm de complexitate $O(n \log n)$.*
- ▶ Există și alte clase de algoritmi mai rapizi; [Chazelle, 1990]: algoritm liniar.

Metode de triangulare: descompunerea în poligoane monotone

- ▶ Algoritmi de triangulare eficienți: complexitate $O(n)$ pentru poligoane y -monotone [Garey et al., 1978] (algoritmul este descris pe slide-urile următoare).
- ▶ Descompunerea unui poligon oarecare în 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 $O(n \log n)$.*
- ▶ Există și alte clase de algoritmi mai rapizi; [Chazelle, 1990]: algoritm liniar.
- ▶ Găsirea unui algoritm liniar "simplu" [Problemă în The Open Problems Project](#)

Metode de triangulare: descompunerea în poligoane monotone

► 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.

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”).

Metoda - paradigma dreptei de baleiere (*line sweep*)

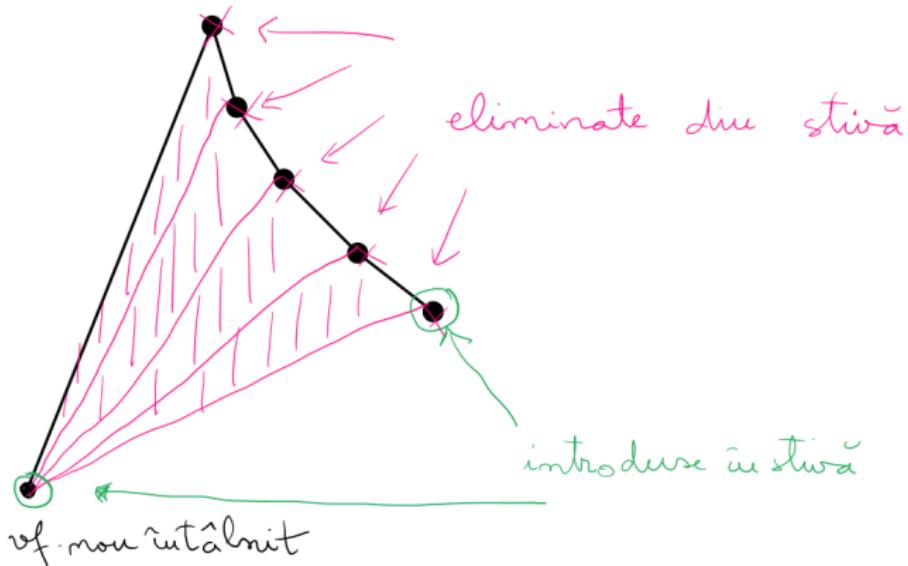
- ▶ Se consideră o dreaptă de baleiere orizontală. Algoritmul reține o serie de informații legate de structura geometrică analizată.
- ▶ **Statut** al dreptei de baleiere: stivă a vârfurilor deja întâlnite, dar care "mai au nevoie de diagonale" / "mai pot să apară în triunghiuri".
(Clarificare. **Q:** 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 știm dacă este pe lanțul din stânga sau pe cel din dreapta.

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 știm dacă este pe lanțul 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.

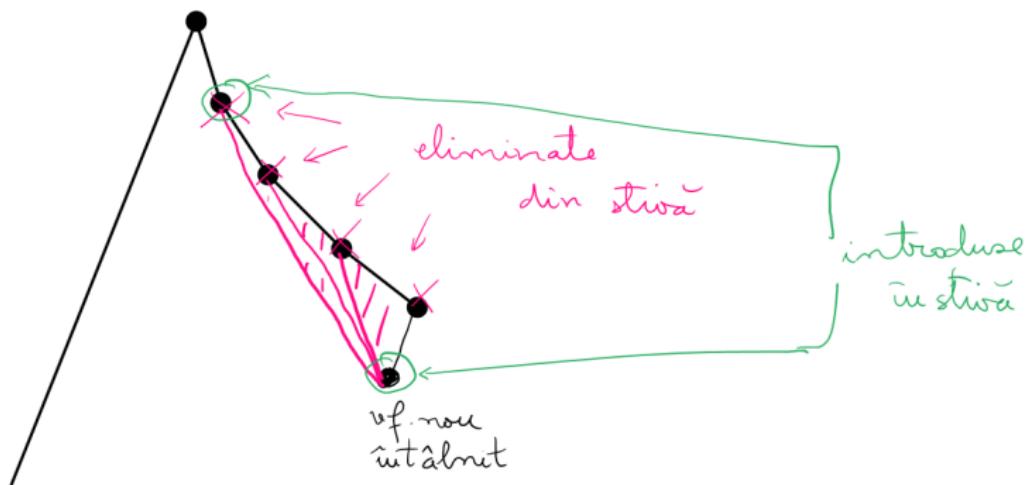
Evenimente - cazul 1

1. Vârful nou întâlnit este pe lanțul opus ultimului vârf din stivă.



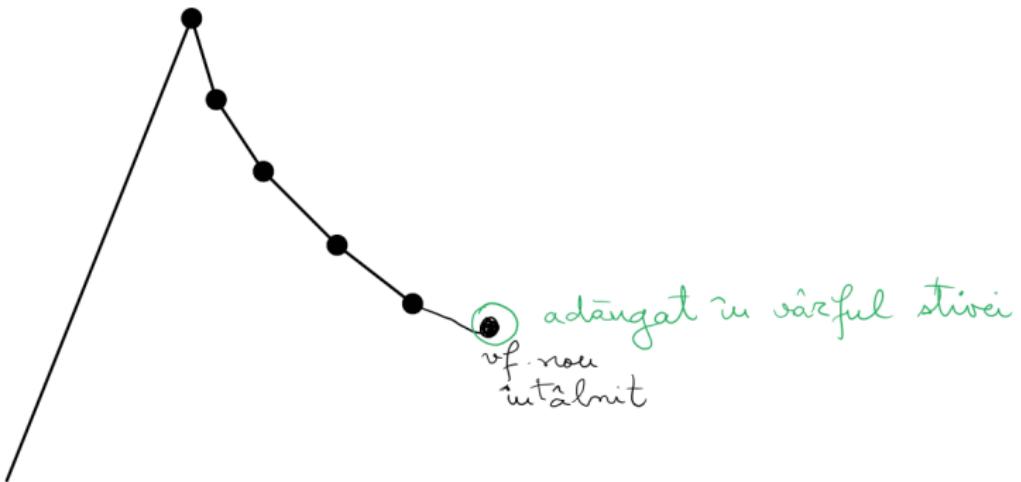
Evenimente - cazul 2a

2a. Vârful nou întâlnit este pe același lanț cu ultimul vârf din stivă.

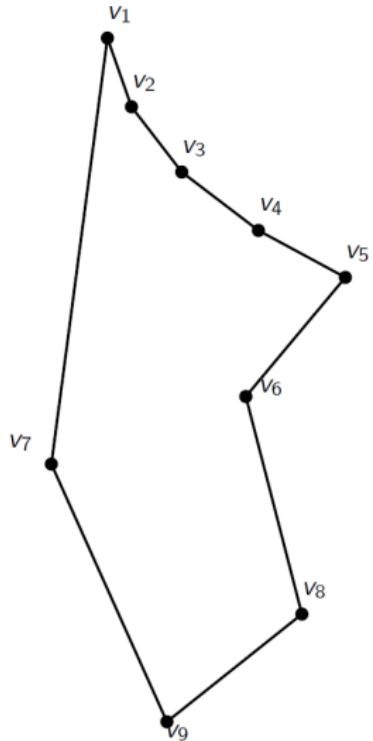


Evenimente - cazul 2b

2b. Vârful nou întâlnit este pe același lanț cu ultimul vârf din stivă.

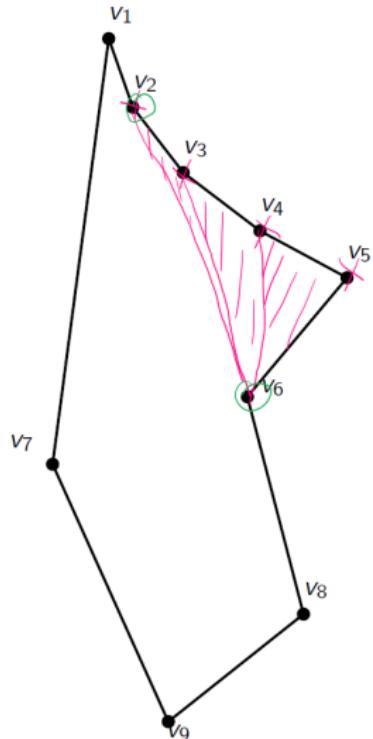


Exemplu



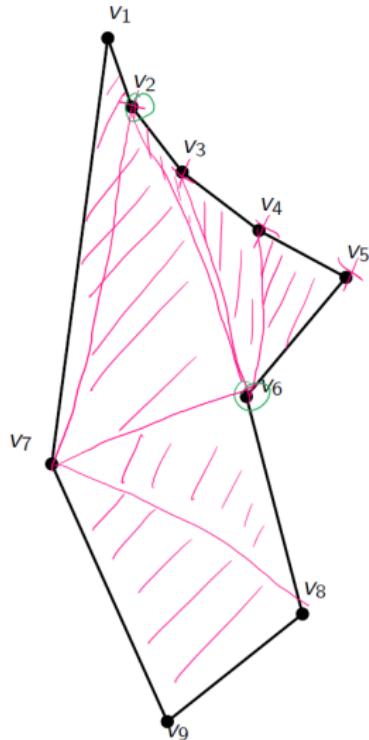
| Eveniment | Stătut |
|-------------------|----------------------------------|
| | v_2 v_1 |
| v_3 (caz 2b) | v_3 v_2 v_1 |
| v_4 (caz 2b) | v_4 v_3 v_2 v_1 |

Exemplu



| Eveniment | Stalut | | | | | | | | |
|-----------------------------|---|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-------|-------|-------|-------|
| v_5 (ω_2 2b) | <table border="1"> <tr><td>v_5</td></tr> <tr><td>v_4</td></tr> <tr><td>v_3</td></tr> <tr><td>v_2</td></tr> <tr><td>v_1</td></tr> </table> | v_5 | v_4 | v_3 | v_2 | v_1 | | | |
| v_5 | | | | | | | | | |
| v_4 | | | | | | | | | |
| v_3 | | | | | | | | | |
| v_2 | | | | | | | | | |
| v_1 | | | | | | | | | |
| v_6 (ω_2 2a) | <table border="1"> <tr><td>v_5</td></tr> <tr><td>v_4</td></tr> <tr><td>v_3</td></tr> <tr><td>v_2</td></tr> <tr><td>v_1</td></tr> </table> <table border="1"> <tr><td>v_6</td></tr> <tr><td>v_2</td></tr> <tr><td>v_1</td></tr> </table> | v_5 | v_4 | v_3 | v_2 | v_1 | v_6 | v_2 | v_1 |
| v_5 | | | | | | | | | |
| v_4 | | | | | | | | | |
| v_3 | | | | | | | | | |
| v_2 | | | | | | | | | |
| v_1 | | | | | | | | | |
| v_6 | | | | | | | | | |
| v_2 | | | | | | | | | |
| v_1 | | | | | | | | | |

Exemplu



| Eveniment | Stalut |
|------------------|--|
| v_7 (caz 1) | <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₆</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₂</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₁</div> </div> |
| v_8 (caz 1) | <div style="display: flex; align-items: center;"> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₇</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₆</div> <div style="border: 1px solid black; padding: 5px; margin-right: 10px;">v₇</div> </div> |
| v_9 | Δ |

La fiecare pas : diagonale în Δ în mod adecvat.

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă S și inserează v_1, v_2 .

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă S și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă S și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui S sunt în lanțuri diferite

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă S și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui S sunt în lanțuri diferite
5. **then** extrage toate vârfurile din S
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7. inserează v_{j-1} și v_j în \mathcal{S}

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7. inserează v_{j-1} și v_j în \mathcal{S}
8. **else** extrage un vârf din \mathcal{S}

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7. inserează v_{j-1} și v_j în \mathcal{S}
8. **else** extrage un vârf din \mathcal{S}
9. extrage celelalte vârfuri din \mathcal{S} dacă diagonalele formate cu v_j sunt în interiorul lui \mathcal{P} ; inserează aceste diagonale; inserează înapoi ultimul vârf extras

Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7. inserează v_{j-1} și v_j în \mathcal{S}
8. **else** extrage un vârf din \mathcal{S}
9. extrage celelalte vârfuri din \mathcal{S} dacă diagonalele formate cu v_j sunt în interiorul lui \mathcal{P} ; inserează aceste diagonale; inserează înapoi ultimul vârf extras
10. inserează v_j în \mathcal{S}

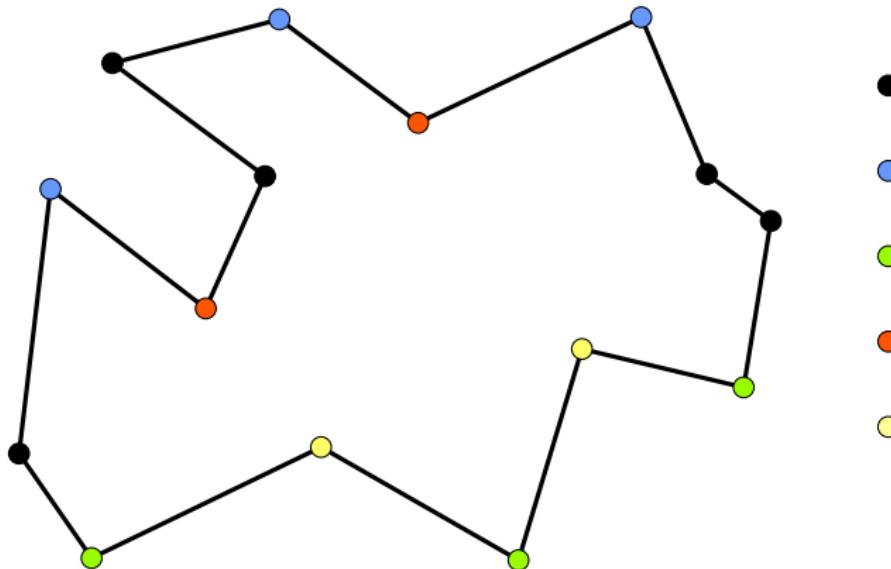
Triangularea poligoanelor monotone

Input: Un poligon y -monoton \mathcal{P} .

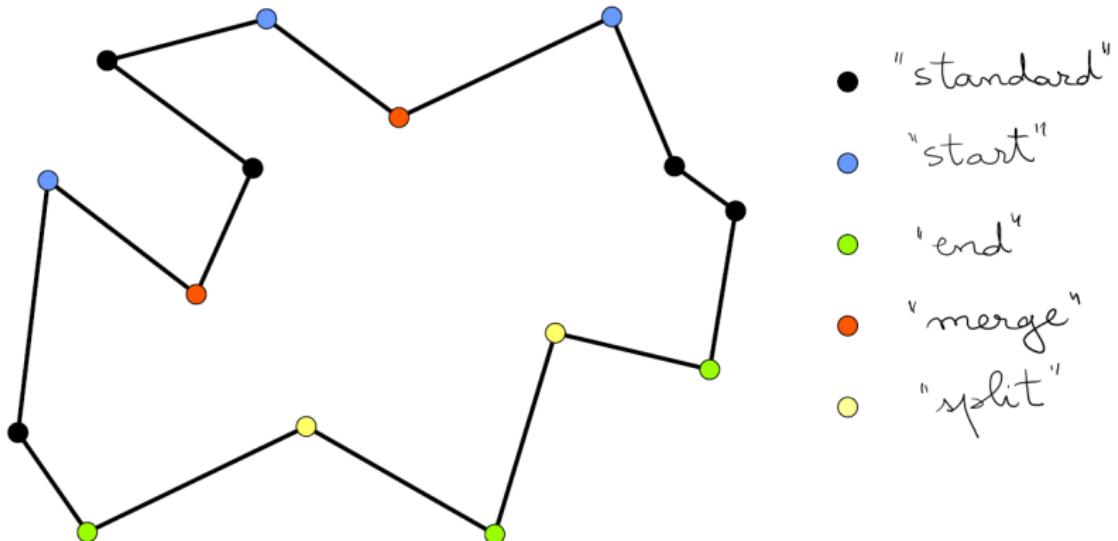
Output: O triangulare a lui \mathcal{P} .

1. Lanțul vârfurilor din partea stângă și al celor din partea dreaptă sunt unite într-un singur sir, ordonat descrescător, după y (dacă ordonata este egală, se folosește abscisa). Fie v_1, v_2, \dots, v_n sirul ordonat.
2. Inițializează o stivă vidă \mathcal{S} și inserează v_1, v_2 .
3. **for** $j = 3$ **to** $n - 1$
4. **do if** v_j și vârful din top al lui \mathcal{S} sunt în lanțuri diferite
5. **then** extrage toate vârfurile din \mathcal{S}
6. inserează diagonale de la v_j la vf. extrase, exceptând ultimul
7. inserează v_{j-1} și v_j în \mathcal{S}
8. **else** extrage un vârf din \mathcal{S}
9. extrage celelalte vârfuri din \mathcal{S} dacă diagonalele formate cu v_j sunt în interiorul lui \mathcal{P} ; inserează aceste diagonale; inserează înapoi ultimul vârf extras
10. inserează v_j în \mathcal{S}
11. adaugă diagonale de la v_n la vf. stivei (exceptând primul și ultimul)

Tipuri de vârfuri



Tipuri de vârfuri



Rezultate

- ▶ Folosind un algoritm bazat pe paradigma dreptei de baleiere și clasificarea vârfurilor indicată, un poligon cu n vârfuri poate fi descompus în poligoane y -monotone cu un algoritm având complexitatea-timp $O(n \log n)$.

Rezultate

- ▶ Folosind un algoritm bazat pe paradigma dreptei de baleiere și clasificarea vârfurilor indicată, un poligon cu n vârfuri poate fi descompus în poligoane y -monotone cu un algoritm având complexitatea-timp $O(n \log n)$.
- ▶ Conform algoritmului descris, un poligon y -monoton poate fi triangulat în timp liniar.

Rezultate

- ▶ Folosind un algoritm bazat pe paradigma dreptei de baleiere și clasificarea vârfurilor indicată, un poligon cu n vârfuri poate fi descompus în poligoane y -monotone cu un algoritm având complexitatea-timp $O(n \log n)$.
- ▶ Conform algoritmului descris, un poligon y -monoton poate fi triangulat în timp liniar.
- ▶ **Teoremă (rezultatul principal)** *Un poligon cu n vârfuri poate fi triangulat cu complexitatea-timp $O(n \log n)$ și complexitatea-spațiu $O(n)$.*

Algoritmi avansați

C3 - Triangularea mulțimilor de puncte

Mihai-Sorin Stupariu

Sem. al II-lea, 2022 - 2023

Triangularea unei mulțimi arbitrară de puncte

Triangulări legale și triangulări unghiular optime

Problematizare

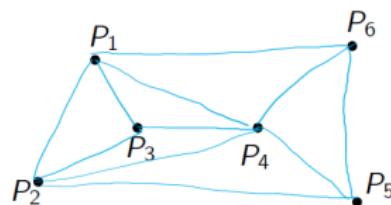
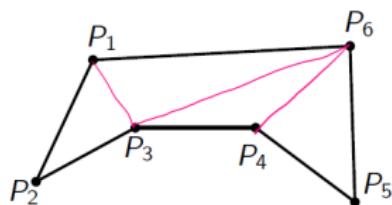
- ▶ Tema anterioară: triangularea unui poligon (listă ordonată de puncte (P_1, P_2, \dots, P_n)).

Problematizare

- ▶ Tema anterioară: triangularea unui poligon (listă ordonată de puncte (P_1, P_2, \dots, P_n)).
- ▶ Are sens să vorbim de triangulare pentru mulțimea $\{P_1, P_2, \dots, P_n\}$?

Problematizare

- ▶ Tema anterioară: triangularea unui poligon (listă ordonată de puncte (P_1, P_2, \dots, P_n)).
- ▶ Are sens să vorbim de triangulare pentru mulțimea $\{P_1, P_2, \dots, P_n\}$?
- ▶ **Exemplu:**



- ▶ În cele ce urmează vom considera doar mulțimi de puncte din planul \mathbb{R}^2 .

Problematizare

- **Definiție.** O **triangulare** a unei mulțimi \mathcal{P} este o subdivizare maximală a acoperirii convexe $\text{Conv}(\mathcal{P})$ a lui \mathcal{P} cu triunghiuri ale căror vârfuri sunt elemente ale lui \mathcal{P} (fără autointersecții!)

Problematizare

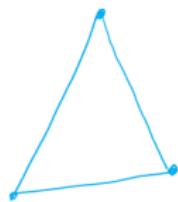
- ▶ **Definiție.** O **triangulare** a unei mulțimi \mathcal{P} este o subdivizare maximală a acoperirii convexe $\text{Conv}(\mathcal{P})$ a lui \mathcal{P} cu triunghiuri ale căror vârfuri sunt elemente ale lui \mathcal{P} (fără autointersecții!)
- ▶ Trebuie făcută distincție între triangulare a unui poligon (P_1, P_2, \dots, P_n) și triangulare a mulțimii subdiacente $\{P_1, P_2, \dots, P_n\}$ (coincid dacă poligonul este convex!)

Problematizare

- ▶ **Definiție.** O **triangulare** a unei mulțimi \mathcal{P} este o subdivizare maximală a acoperirii convexe $\text{Conv}(\mathcal{P})$ 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, \dots, P_n) și triangulare a mulțimii subdiacente $\{P_1, P_2, \dots, P_n\}$ (coincid dacă poligonul este convex!)
- ▶ **Comentariu:** Triangulările mulțimilor de puncte sunt esențiale în grafica pe calculator.

Exemple

(i) 3 puncte necoliniare

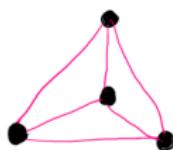


3 vârfuri
3 muchii
1 față

Exemple

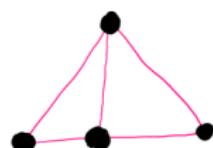
(ii) 4 puncte necoliniare, nesituate toate pe o aceeași dreaptă

(a)



4 vârfuri
6 muchii
3 fete (3 Δ)

(b)



4 vârfuri
5 muchii
2 fete (2 Δ)

(c)



4 vârfuri
5 muchii
2 fete (2 Δ)

Elemente ale unei triangulări

- ▶ Dată o mulțime de puncte \mathcal{P} și o triangulare \mathcal{T}_P a sa:
vârfuri, muchii, triunghiuri.

Elemente ale unei triangulări

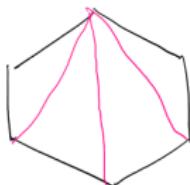
- ▶ Dată o mulțime de puncte \mathcal{P} și o triangulare \mathcal{T}_P a sa:
vârfuri, muchii, triunghiuri.
- ▶ Legătură cantitativă între aceste elemente?

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 frontieră acoperirii convexe $\text{Conv}(\mathcal{P})$. Orice triangulare a lui \mathcal{P} are $(2n - k - 2)$ triunghiuri și $(3n - k - 3)$ muchii.

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 frontieră acoperirii convexe $\text{Conv}(\mathcal{P})$. Orice triangulare a lui \mathcal{P} are $(2n - k - 2)$ triunghiuri și $(3n - k - 3)$ muchii.
- ▶ **Exemplu:** Cazul unui poligon convex: un poligon convex cu n vârfuri poate fi triangulat cu $(n - 2)$ triunghiuri, având $(2n - 3)$ muchii.

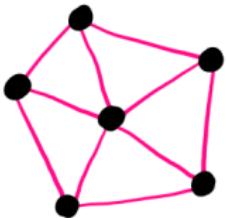


$$\underline{n = 6}$$

4 \triangle

9 muchii

Demonstrație



Graf:

noduri le: punctele inițiale (n)

muchiiile: laturile Δ ($n_m = ?$)

fetele: fețele Δ + fata exterioară
 $(n_t + 1)$
 ||?
 ?

- Relatia lui Euler: $n - n_m + (n_t + 1) = 2$

- Incidente dintre muchii și fețe

$$\underbrace{2 \cdot n_m}_{\text{"perspectiva muchiilor"}} = \underbrace{\frac{n_t}{3 \cdot n_t} + \frac{1}{k}}_{\text{"perspectiva fețelor"}}$$

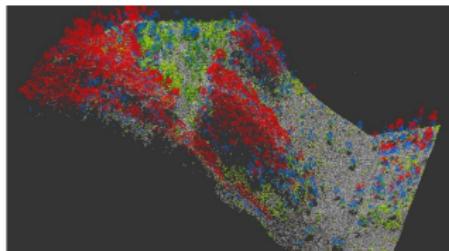
$$\begin{cases} \Rightarrow \dots \\ \Rightarrow \dots \\ n_m = \dots \\ n_t = \dots \end{cases}$$

Problematizare

- **Problemă.** Se fac măsurători ale altitudinii pentru un teren. Se dorește reprezentarea tridimensională (cât mai sugestivă) . Alternativ: se dorește generarea unui teren pentru o aplicație.

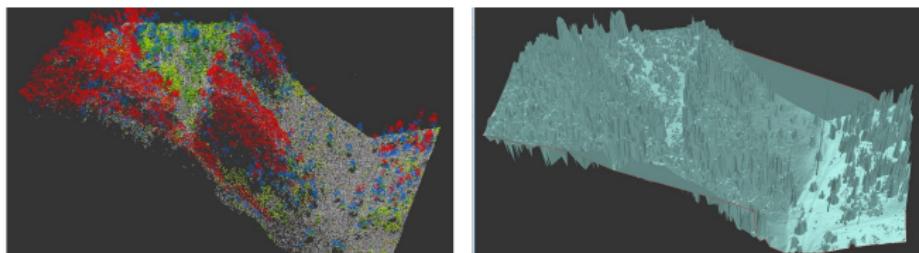
Problematizare

- **Problemă.** Se fac măsurători ale altitudinii pentru un teren. Se dorește reprezentarea tridimensională (cât mai sugestivă) . Alternativ: se dorește generarea unui teren pentru o aplicație.



Problematizare

- **Problemă.** Se fac măsurători ale altitudinii pentru un teren. Se dorește **reprezentarea tridimensională** (cât mai sugestivă) . Alternativ: se dorește **generarea unui teren** pentru o aplicație.

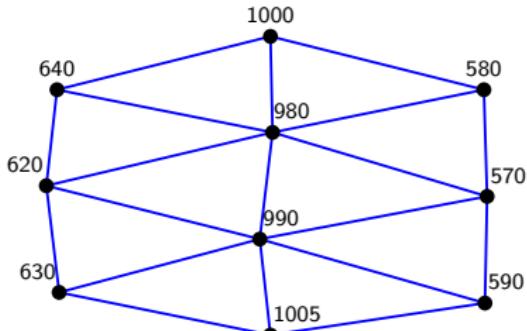


Problematizare - continuare

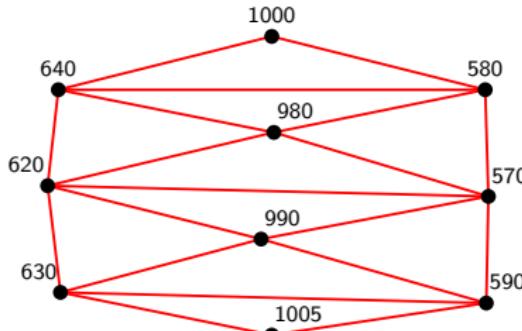
- **Problemă (reformulată).** Cum "comparăm triangulările" unei multimi de puncte fixate?

Problematizare - continuare

- ▶ **Problemă (reformulată).** Cum "comparăm triangulările" unei multimi de puncte fixate?
- ▶ **Exemplu.** Măsurători ale altitudinii.



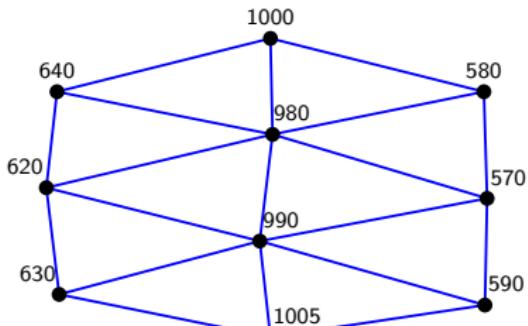
Triangulare 1



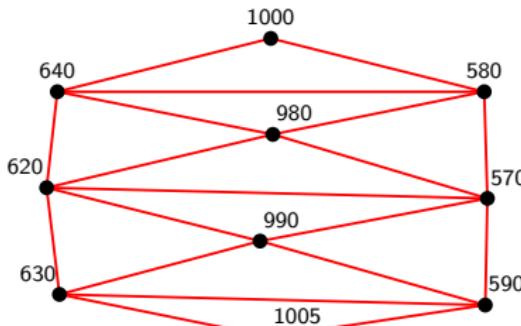
Triangulare 2

Problematizare - continuare

- ▶ **Problemă (reformulată).** Cum "comparăm triangulările" unei mulțimi de puncte fixate?
- ▶ **Exemplu.** Măsurători ale altitudinii.



Triangulare 1



Triangulare 2

- ▶ **Întrebări naturale:** (i) Există o triangulare "convenabilă" a unei mulțimi de puncte? (ii) Cum poate fi determinată eficient o astfel de triangulare?

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 .**

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 $\alpha_1, \alpha_2, \dots, \alpha_{3m}$ unghiiurile lui \mathcal{T} , ordonate crescător. **Vectorul unghiurilor lui \mathcal{T} este $A(\mathcal{T}) = (\alpha_1, \alpha_2, \dots, \alpha_{3m})$.**

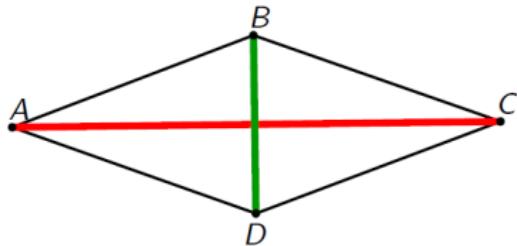
Terminologie, triangulări legale

- ▶ 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 $\alpha_1, \alpha_2, \dots, \alpha_{3m}$ unghiiurile lui \mathcal{T} , ordonate crescător. **Vectorul unghiurilor lui \mathcal{T} este $A(\mathcal{T}) = (\alpha_1, \alpha_2, \dots, \alpha_{3m})$.**
- ▶ **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 $\alpha_i > \alpha'_i$.

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 $\alpha_1, \alpha_2, \dots, \alpha_{3m}$ unghiiurile lui \mathcal{T} , ordonate crescător. **Vectorul unghiurilor lui \mathcal{T} este $A(\mathcal{T}) = (\alpha_1, \alpha_2, \dots, \alpha_{3m})$.**
- ▶ **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}' .

Exemplu - cazul unui patrulater convex



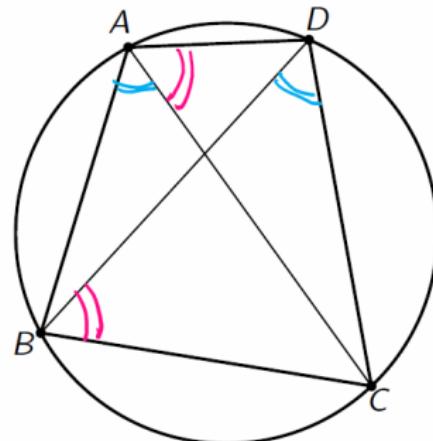
$$\text{diagonal } AC \\ \Rightarrow \triangle BAC \cong \triangle DAC$$

diagonala BD
→ $\triangle ABD \cong \triangle CBD$

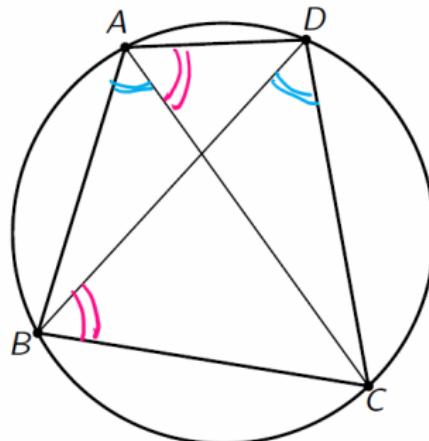
diagonala $AC \Rightarrow$ vectorul unghiurilor (α) $\alpha < \beta$
 $BD \Rightarrow$ $__ - -$ (β)

"cel mai mic unghi care apare în triunghiul dată de AC este mai mic decât cel mai mic unghi care apare în triunghiul data de BD"

Exemplu - cazul unui patrulater inscriptibil



Exemplu - cazul unui patrulater inscriptibil



În acest caz triunghiurile formate de diagonale au “cele mai mici unghiuri” congruente, deci nu putem distinge între cele două diagonale.

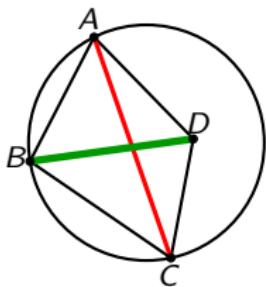
Muchii ilegale

- ▶ **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
$$\min A(\mathcal{T}_{AC}) < \min A(\mathcal{T}_{BD}).$$

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 $\mathcal{T}_{AC}, \mathcal{T}_{BD}$ triunghiurile 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 .



Muchii ilegale

- ▶ **Criteriu numeric / analitic** pentru a testa dacă o muchie este ilegală.
- ▶ Pentru puncte $A = (x_A, y_A)$, $B = (x_B, y_B)$, $C = (x_C, y_C)$, $D = (x_D, y_D)$:

$$\Theta(A, B, C, D) = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix}$$

Muchii ilegale

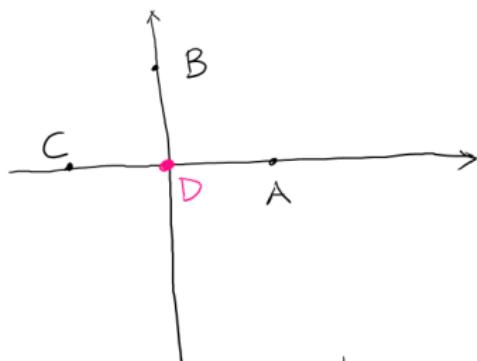
- ▶ **Criteriu numeric / analitic** pentru a testa dacă o muchie este ilegală.

- ▶ 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 $\Delta ABC \Leftrightarrow \Theta(A, B, C, D) > 0$.

Exemplu



$$A = (1, 0)$$

$$B = (0, 1)$$

$$C = (-1, 0)$$

- ABC este viraj la stânga

$$D = (0, 0)$$

$$\bullet \text{ (H)} (A, B, C, D) = \begin{vmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 0 & 0 & 0 & -1 \end{vmatrix} = (-1)^{4+4} \begin{vmatrix} 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ -1 & 1 \end{vmatrix} =$$

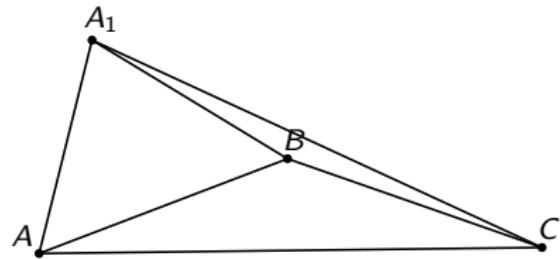
$$= \begin{vmatrix} 1 & 1 \\ -1 & 1 \end{vmatrix} \geq 0$$

D este în interiorul cercului circumscris $\triangle ABC$.

Exercițiu: calculați pt. $E = (0, -1)$ și $F = (0, -2)$

Muchii ilegale, triangulări legale

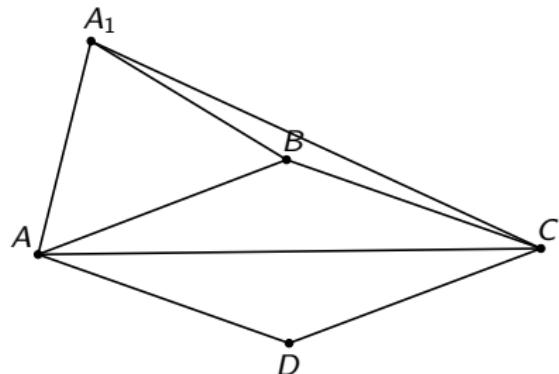
- **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local).



D

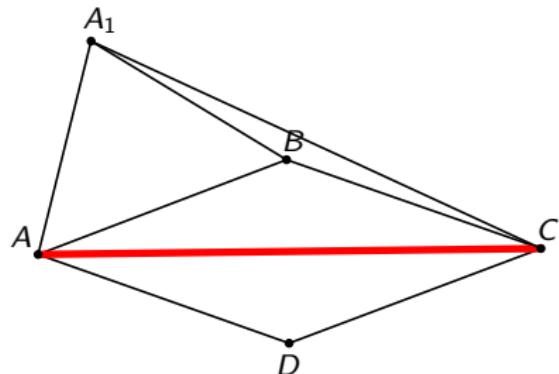
Muchii ilegale, triangulări legale

- **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local).



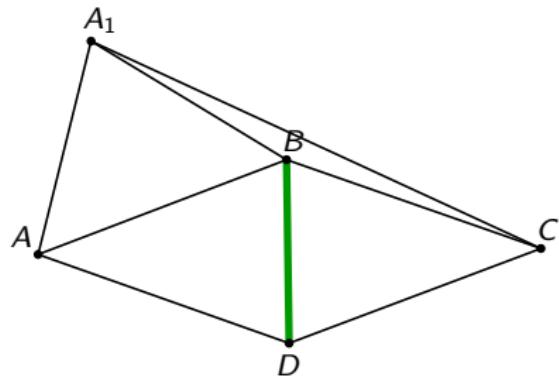
Muchii ilegale, triangulări legale

- **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local).



Muchii ilegale, triangulări legale

- **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local).



Muchii ilegale, triangulări legale

- **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local). Prințr-un flip, vectorul unghiurilor crește.

Muchii ilegale, triangulări legale

- ▶ **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local). Prințr-un flip, vectorul unghiurilor crește.
- ▶ **Concluzie (reformulare):** Fie \mathcal{T} o triangulare cu o muchie ilegală e , fie \mathcal{T}' triangularea obținută din \mathcal{T} prin *flip*-ul muchiei e . Atunci $A(\mathcal{T}') > A(\mathcal{T})$.

Muchii ilegale, triangulări legale

- ▶ **Concluzie:** Dacă muchia AC este ilegală, printr-un *flip* (înlocuirea ei cu BD), cel mai mic unghi poate fi mărit (local). Prințr-un flip, vectorul unghiurilor crește.
- ▶ **Concluzie (reformulare):** Fie \mathcal{T} o triangulare cu o muchie ilegală e , fie \mathcal{T}' triangularea obținută din \mathcal{T} prin *flip*-ul muchiei e . Atunci $A(\mathcal{T}') > A(\mathcal{T})$.
- ▶ **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 $O(n \log n)$.

Triangulări unghiular optime vs. triangulări legale

► **Propoziție.** Fie \mathcal{P} o mulțime de puncte din plan.

- (i) Orice triangulare unghiular optimă este legală.
- (ii) Dacă \mathcal{P} este în poziție generală (oricare patru puncte nu sunt conciclice), atunci există o unică triangulare legală, iar aceasta este unghiular optimă.

Triangulări unghiular optime vs. triangulări legale

► **Propoziție.** Fie \mathcal{P} o mulțime de puncte din plan.

- (i) Orice triangulare unghiular optimă este legală.
- (ii) Dacă \mathcal{P} este în poziție generală (oricare patru puncte nu sunt conciclice), atunci există o unică triangulare legală, iar aceasta este unghiular optimă.

► **Teoremă.** Fie \mathcal{P} o mulțime de n puncte din plan, în poziție generală. Triangularea unghiular optimă poate fi construită, folosind un algoritm incremental randomizat, în timp mediu $O(n \log n)$, folosind $O(n)$ memorie medie.

Scurt rezumat

- Triangulare a unei mulțimi de puncte. Rezultat referitor la numărul de triunghiuri și muchii ale unei triangulări (și la seminar).

Scurt rezumat

- ▶ Triangulare a unei mulțimi de puncte. Rezultat referitor la numărul de triunghiuri și muchii ale unei triangulări (și la seminar).
- ▶ Compararea triangulărilor - criteriu folosind unghiurile. Triangulări unghiular optime.

Scurt rezumat

- ▶ Triangulare a unei mulțimi de puncte. Rezultat referitor la numărul de triunghiuri și muchii ale unei triangulări (și la seminar).
- ▶ Compararea triangulărilor - criteriu folosind unghiurile. Triangulări unghiular optime.
- ▶ Conceptul de muchie ilegală. Criteriu numeric referitor la configurația a patru puncte.

Scurt rezumat

- ▶ Triangulare a unei mulțimi de puncte. Rezultat referitor la numărul de triunghiuri și muchii ale unei triangulări (și la seminar).
- ▶ Compararea triangulărilor - criteriu folosind unghiurile. Triangulări unghiular optime.
- ▶ Conceptul de muchie ilegală. Criteriu numeric referitor la configurația a patru puncte.
- ▶ Triangulări legale.

Scurt rezumat

- ▶ Triangulare a unei mulțimi de puncte. Rezultat referitor la numărul de triunghiuri și muchii ale unei triangulări (și la seminar).
- ▶ Compararea triangulărilor - criteriu folosind unghiurile. Triangulări unghiular optime.
- ▶ Conceptul de muchie ilegală. Criteriu numeric referitor la configurația a patru puncte.
- ▶ Triangulări legale.
- ▶ Legătura dintre triangulările unghiular optime și triangulările legale.

Algoritmi avansați

C4 - Diagrame Voronoi

Mihai-Sorin Stupariu

Sem. al II-lea, 2022 - 2023

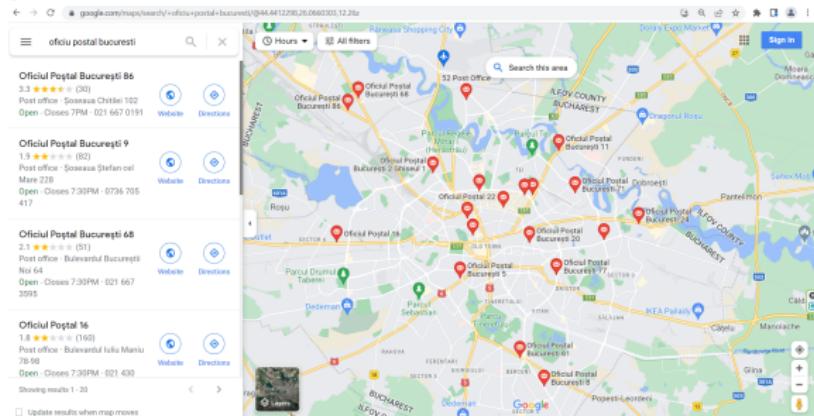
Definiție, proprietăți elementare

Diagrame Voronoi și triangulări Delaunay

Un algoritm eficient

Motivație

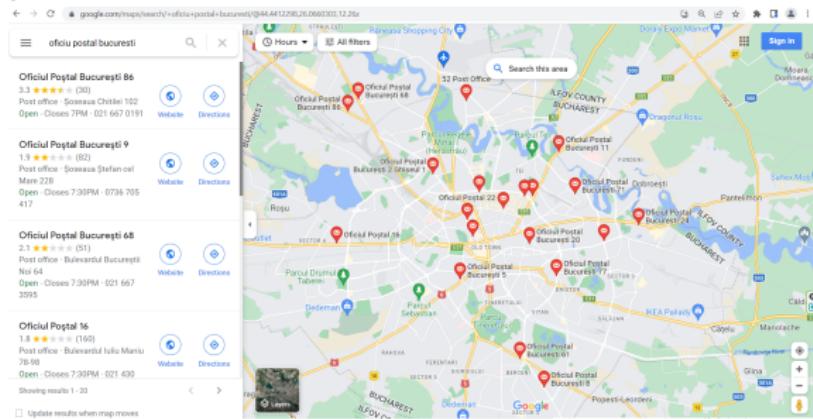
- ▶ *Problema oficiilor poștale:* Se consideră o mulțime de puncte (oficiile poștale) din plan. Ce zone vor deservi aceste oficii?



Sursa: [Google Maps](#)

Motivație

- ▶ *Problema oficiilor poștale:* Se consideră o mulțime de puncte (oficiile poștale) din plan. Ce zone vor deservi aceste oficii?



Sursa: [Google Maps](#)

- ▶ Ideea de a delimita “zone de influență” a apărut cu multă vreme în urmă (de exemplu în [lucrările lui Descartes](#), dar și în [legătură cu alte probleme](#); este utilizată în mod curent în varii domenii. În plus, astfel de “împărțiri” apar [în natură](#). Conceptul aferent este cel de **diagramă Voronoi** - există o multitudine de aplicații.

Aplicații - rețele de transport

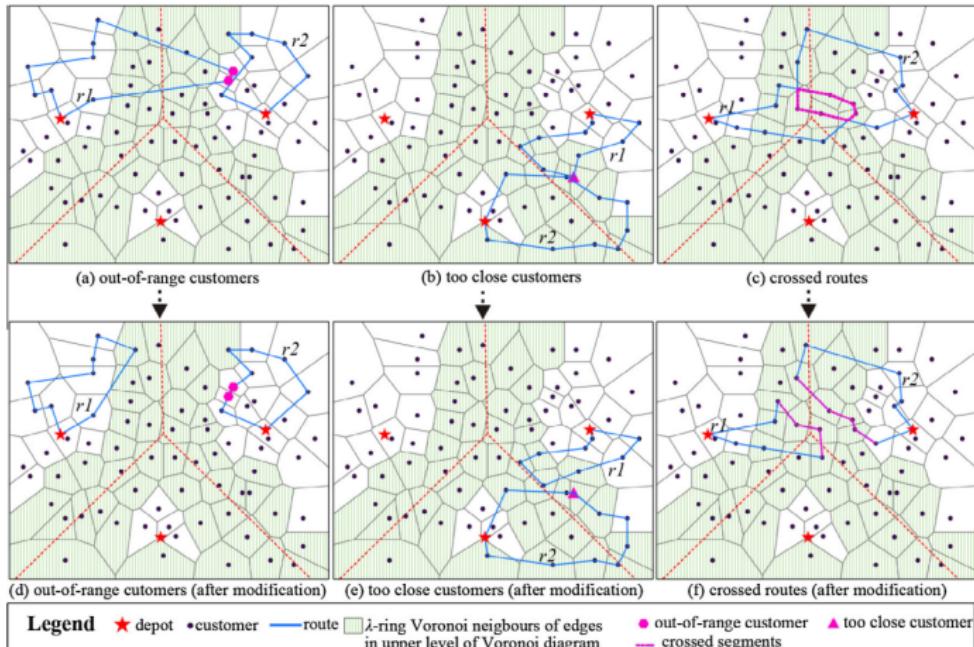


Fig. 3. Inappropriate routes between depots and their modifications.

Sursa: [Tu et al., 2014]

Aplicații - rețele de transport

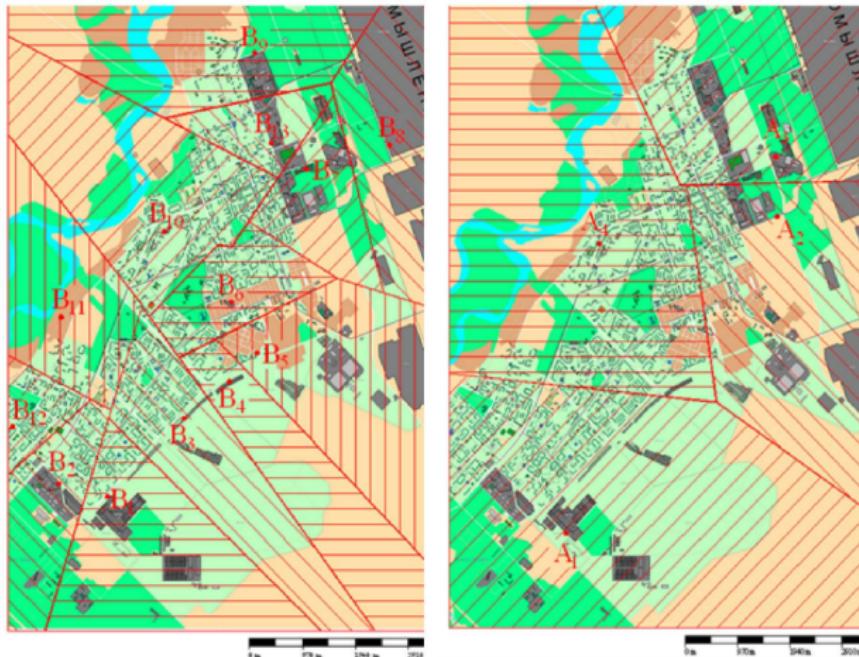


Fig. 4. Voronoi diagram to optimize: (a) the zones serviced by transport terminals of large city districts;(b) freight delivery zones from the logistic centers of a large city.

Sursa: [Lebedeva et al., 2018]

Aplicații - medicină

ANTIGA *et al.*: COMPUTATIONAL GEOMETRY FOR PATIENT-SPECIFIC RECONSTRUCTION AND MESHING OF BLOOD VESSELS

681

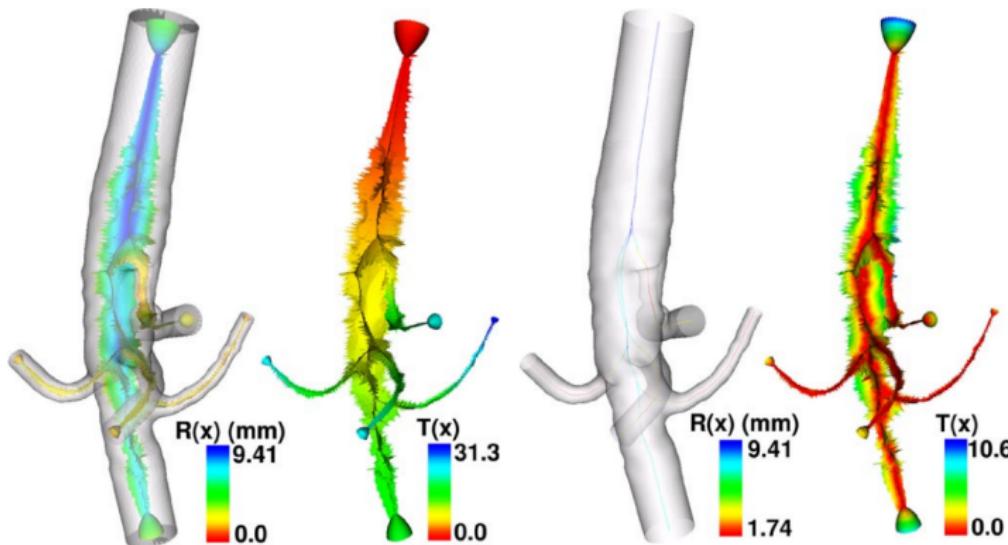


Fig. 8. Voronoi diagram colored according to maximal inscribed sphere radius values $R(x)$ (left), and solution $T(x)$ of Eikonal equation from aorta inlet (right) for the abdominal aorta model (Surface: 8096 points, 16188 triangles. Voronoi diagram: 26463 points, 18370 polygons. Voronoi diagram computation time: 10 s. Solution of Eikonal equation time: 8 s).

Fig. 9. Calculated central paths (left) and solution of Eikonal equation from central path points (right) for subsequent surface characterization (total five central paths backtracing time: 1 s. Solution of Eikonal equation time: 8 s). $R(x)$ is maximal inscribed ball radius values defined on central paths, and $T(x)$ is maximal traveltime (since in this case $F(x) = 1$ everywhere, $T(x)$ also represents geodesic distance).

Sursa: [Antiga et al., 2003]

Aplicații - medicină

C. Wang, H.R. Roth, T. Kitasaka et al. / Computerized Medical Imaging and Graphics 77 (2019) 101642

3

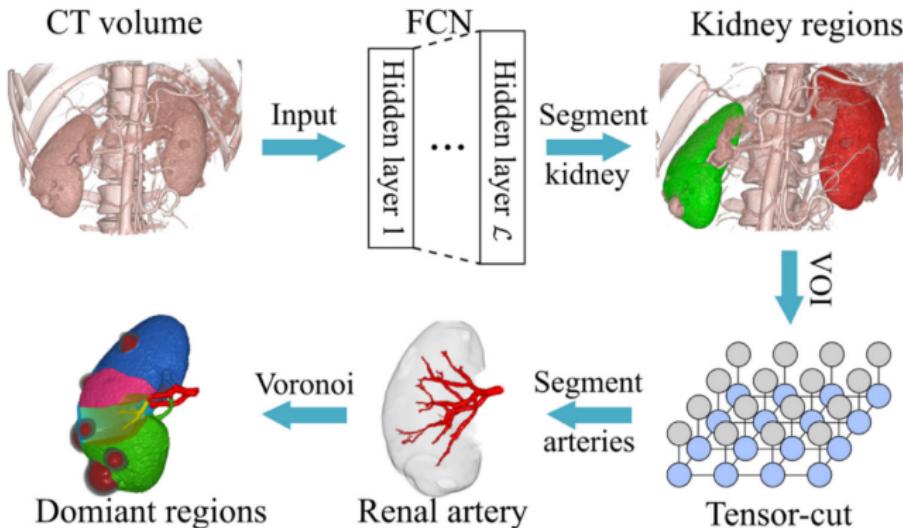
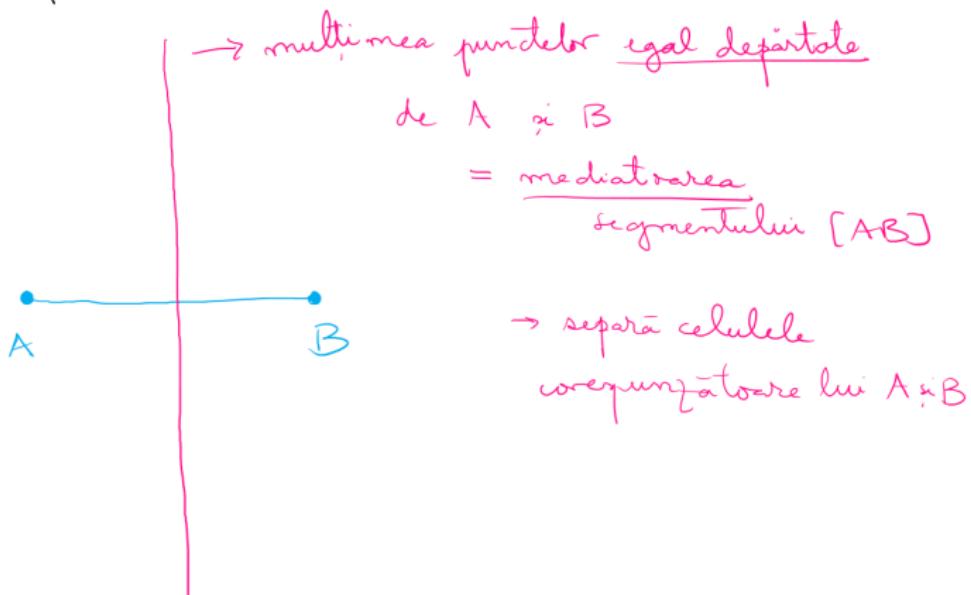


Fig. 1. Workflow: Our precise estimation approach can be divided into three parts: kidney segmentation, renal artery segmentation and estimation of vascular dominant regions.

Sursa: [Wang et al., 2019]

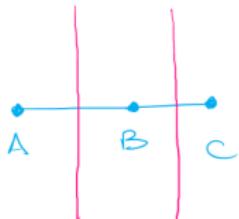
Exemple - diagrame Voronoi în context 2D

1)

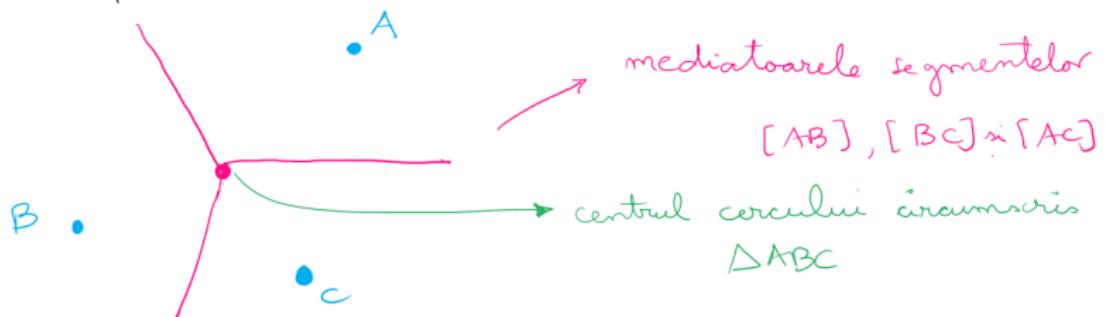
2 puncte distințte

Exemple - diagrame Voronoi în context 2D

2) 3 puncte distincte, coliniare

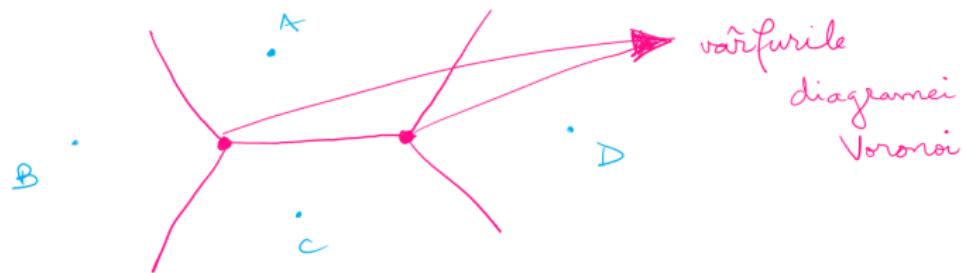


3) 3 puncte necoliniare

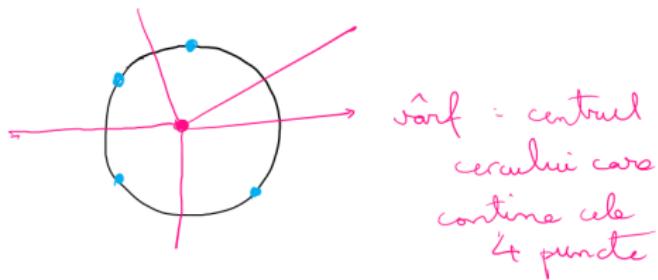


Exemple - diagrame Voronoi în context 2D

4) 4 puncte neclimni ore , necociclice



4') 4 puncte concidice



Formalizare

- Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.

Formalizare

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.
- ▶ **Diagrama Voronoi** a lui \mathcal{P} (notată $\text{Vor}(\mathcal{P})$) este o divizare a planului \mathbb{R}^2 în n celule $\mathcal{V}(P_1), \dots, \mathcal{V}(P_n)$ cu proprietatea că

$$P \in \mathcal{V}(P_i) \Leftrightarrow d(P, P_i) \leq d(P, P_j), \quad \forall j = 1, \dots, n.$$

Formalizare

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.
- ▶ **Diagrama Voronoi** a lui \mathcal{P} (notată $\text{Vor}(\mathcal{P})$) este o divizare a planului \mathbb{R}^2 în n celule $\mathcal{V}(P_1), \dots, \mathcal{V}(P_n)$ cu proprietatea că

$$P \in \mathcal{V}(P_i) \Leftrightarrow d(P, P_i) \leq d(P, P_j), \quad \forall j = 1, \dots, n.$$

- ▶ Două celule adiacente au în comun o *muchie* sau un *vârf* (punct de intersecție a muchiilor).

Formalizare

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.
- ▶ **Diagrama Voronoi** a lui \mathcal{P} (notată $\text{Vor}(\mathcal{P})$) este o divizare a planului \mathbb{R}^2 în n celule $\mathcal{V}(P_1), \dots, \mathcal{V}(P_n)$ cu proprietatea că

$$P \in \mathcal{V}(P_i) \Leftrightarrow d(P, P_i) \leq d(P, P_j), \quad \forall j = 1, \dots, n.$$

- ▶ Două celule adiacente au în comun o *muchie* sau un *vârf* (punct de intersecție a muchiilor).
- ▶ **Atenție!** Vârfurile lui $\text{Vor}(\mathcal{P})$ sunt diferite de punctele din \mathcal{P} .

Formalizare

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.
- ▶ **Diagrama Voronoi** a lui \mathcal{P} (notată $\text{Vor}(\mathcal{P})$) este o divizare a planului \mathbb{R}^2 în n celule $\mathcal{V}(P_1), \dots, \mathcal{V}(P_n)$ cu proprietatea că

$$P \in \mathcal{V}(P_i) \Leftrightarrow d(P, P_i) \leq d(P, P_j), \quad \forall j = 1, \dots, n.$$

- ▶ Două celule adiacente au în comun o *muchie* sau un *vârf* (punct de intersecție a muchiilor).
- ▶ **Atenție!** Vârfurile lui $\text{Vor}(\mathcal{P})$ sunt diferite de punctele din \mathcal{P} .
- ▶ Uneori, prin abuz de limbaj, este precizată doar împărțirea în muchii / vârfuri.

Formalizare

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de puncte din planul \mathbb{R}^2 , numite **situri**.
- ▶ **Diagrama Voronoi** a lui \mathcal{P} (notată $\text{Vor}(\mathcal{P})$) este o divizare a planului \mathbb{R}^2 în n celule $\mathcal{V}(P_1), \dots, \mathcal{V}(P_n)$ cu proprietatea că

$$P \in \mathcal{V}(P_i) \Leftrightarrow d(P, P_i) \leq d(P, P_j), \quad \forall j = 1, \dots, n.$$

- ▶ Două celule adiacente au în comun o *muchie* sau un *vârf* (punct de intersecție a muchiilor).
- ▶ **Atenție!** Vârfurile lui $\text{Vor}(\mathcal{P})$ sunt diferite de punctele din \mathcal{P} .
- ▶ Uneori, prin abuz de limbaj, este precizată doar împărțirea în muchii / vârfuri.
- ▶ Diagrame Voronoi pot fi construite pentru **diverse funcții distanță** (e.g. **distanța Manhattan**); forma celulelor depinde de **forma "cercului"** în raport cu funcția distanță respectivă.

Structura unei diagrame Voronoi

- Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de situri (puncte) din planul \mathbb{R}^2 .

Structura unei diagrame Voronoi

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de situri (puncte) din planul \mathbb{R}^2 .
- ▶ Celula asociată unui punct este o intersecție de semiplane:

$$\mathcal{V}(P_i) = \bigcap_{j \neq i} h(P_i, P_j),$$

unde $h(P_i, P_j)$ este semiplanul determinat de mediatoarea segmentului $[P_i P_j]$ care conține punctul P_i . În particular: fiecare celulă este o mulțime convexă.

Aplicabilitate: algoritm (lent) de determinare a diagramei Voronoi.

Structura unei diagrame Voronoi

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de situri (puncte) din planul \mathbb{R}^2 .
- ▶ Celula asociată unui punct este o intersecție de semiplane:

$$\mathcal{V}(P_i) = \bigcap_{j \neq i} h(P_i, P_j),$$

unde $h(P_i, P_j)$ este semiplanul determinat de mediatoarea segmentului $[P_i P_j]$ care conține punctul P_i . În particular: fiecare celulă este o mulțime convexă.

Aplicabilitate: algoritm (lent) de determinare a diagramei Voronoi.

- ▶ Dacă toate punctele sunt coliniare, atunci diagrama Voronoi asociată $\text{Vor}(\mathcal{P})$ conține $n - 1$ drepte paralele între ele (în particular, pentru $n \geq 3$, ea nu este conexă).
- ▶ În caz contrar, diagrama este conexă, iar muchiile sale sunt fie *segmente*, fie *semidrepte* (cui corespund acestea?).

Structura unei diagrame Voronoi

- ▶ Fie $\mathcal{P} = \{P_1, P_2, \dots, P_n\}$ o mulțime de situri (puncte) din planul \mathbb{R}^2 .
- ▶ Celula asociată unui punct este o intersecție de semiplane:

$$\mathcal{V}(P_i) = \bigcap_{j \neq i} h(P_i, P_j),$$

unde $h(P_i, P_j)$ este semiplanul determinat de mediatoarea segmentului $[P_i P_j]$ care conține punctul P_i . În particular: fiecare celulă este o mulțime convexă.

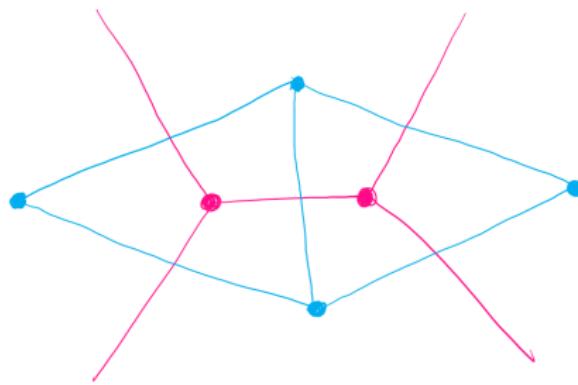
Aplicabilitate: algoritm (lent) de determinare a diagramei Voronoi.

- ▶ Dacă toate punctele sunt coliniare, atunci diagrama Voronoi asociată $\text{Vor}(\mathcal{P})$ conține $n - 1$ drepte paralele între ele (în particular, pentru $n \geq 3$, ea nu este conexă).
- ▶ În caz contrar, diagrama este conexă, iar muchiile sale sunt fie *segmente*, fie *semidrepte* (cui corespund acestea?).
- ▶ **Propoziție.** Fie o mulțime cu n situri. Atunci, pentru diagrama Voronoi asociată au loc inegalitățile

$$n_v \leq 2n - 5, \quad n_m \leq 3n - 6,$$

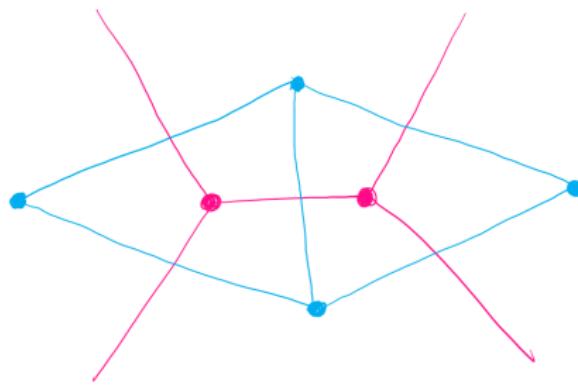
unde n_v este numărul de vârfuri ale diagramei și n_m este numărul de muchii al acesteia.

Legătura cu triangulările legale / unghiular optime



- ▶ Construcție:

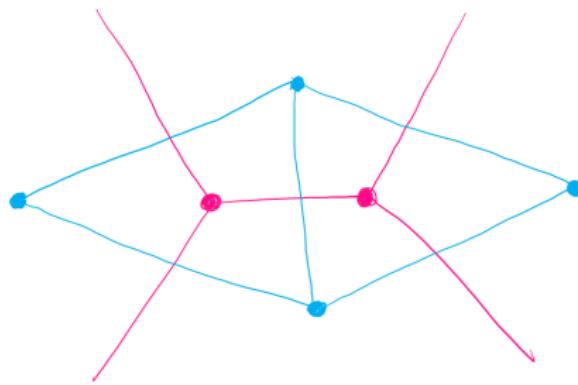
Legătura cu triangulările legale / unghiular optime



► Construcție:

- Multime de puncte \mathcal{P} în planul $\mathbb{R}^2 \implies$

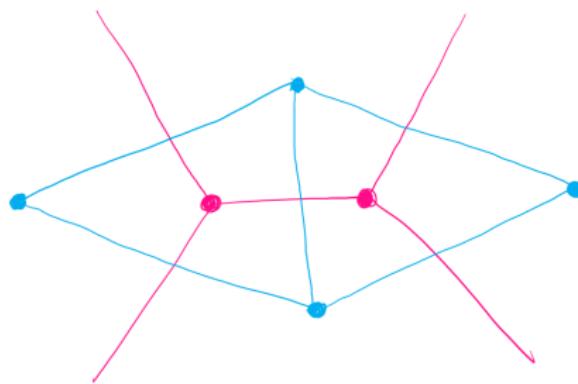
Legătura cu triangulările legale / unghiular optime



► Construcție:

- Mulțime de puncte \mathcal{P} în planul $\mathbb{R}^2 \implies$
- Diagrama Voronoi $\text{Vor}(\mathcal{P}) \implies$

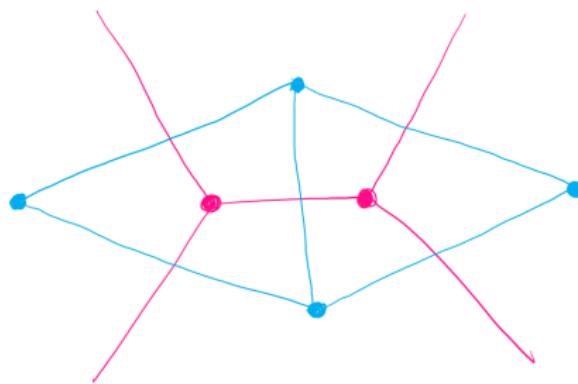
Legătura cu triangulările legale / unghiular optime



► Construcție:

- Mulțime de puncte \mathcal{P} în planul $\mathbb{R}^2 \implies$
- Diagrama Voronoi $\text{Vor}(\mathcal{P}) \implies$
- Graful dual $\mathcal{G}(\mathcal{P})$. **Noduri:** fețele diagramei Voronoi (siturile). **Arce:** dacă celulele (fețele diagramei Voronoi corespunzătoare) au o muchie comună \implies

Legătura cu triangulările legale / unghiular optime

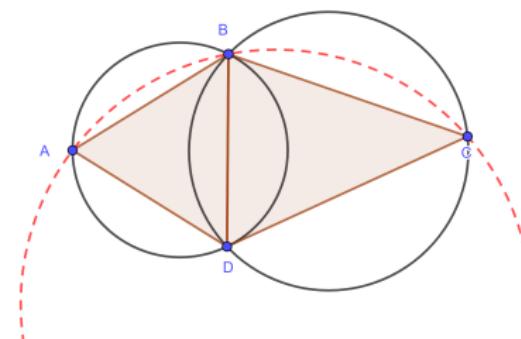


► Construcție:

- Mulțime de puncte \mathcal{P} în planul $\mathbb{R}^2 \implies$
- Diagrama Voronoi $\text{Vor}(\mathcal{P}) \implies$
- Graful dual $\mathcal{G}(\mathcal{P})$. **Noduri:** fețele diagramei Voronoi (siturile). **Arce:** dacă celulele (fețele diagramei Voronoi corespunzătoare) au o muchie comună \implies
- Triangulare $\mathcal{T}_{\mathcal{P}}$ (numită **triangulare Delaunay**)

Legătura cu triangulările legale / unghiular optime

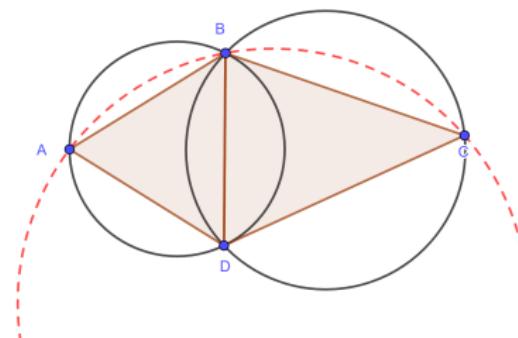
- ▶ **Propoziție.** Fie \mathcal{T} o triangulare a lui \mathcal{P} . Atunci \mathcal{T} este o triangulare Delaunay dacă și numai dacă pentru orice triunghi din \mathcal{T} cercul circumscris nu conține în interiorul său niciun punct al lui \mathcal{P} .



Pentru punctele A, B, C, D din figură, triangularea construită este o triangulare Delaunay. În schimb, triunghiul ΔABC nu poate participa la realizarea unei triangulări Delaunay, deoarece D este în interiorul cercului circumscris acestuia.

Legătura cu triangulările legale / unghiular optime

- ▶ **Propoziție.** Fie \mathcal{T} o triangulare a lui \mathcal{P} . Atunci \mathcal{T} este o triangulare Delaunay dacă și numai dacă pentru orice triunghi din \mathcal{T} cercul circumscris nu conține în interiorul său niciun punct al lui \mathcal{P} .

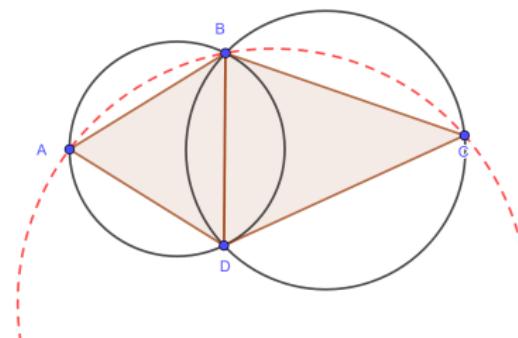


Pentru punctele A, B, C, D din figură, triangularea construită este o triangulare Delaunay. În schimb, triunghiul ΔABC nu poate participa la realizarea unei triangulări Delaunay, deoarece D este în interiorul cercului circumscris acestuia.

- ▶ **Teoremă.** O triangulare este legală dacă și numai dacă este o triangulare Delaunay.

Legătura cu triangulările legale / unghiular optime

- ▶ **Propoziție.** Fie \mathcal{T} o triangulare a lui \mathcal{P} . Atunci \mathcal{T} este o triangulare Delaunay dacă și numai dacă pentru orice triunghi din \mathcal{T} cercul circumscris nu conține în interiorul său niciun punct al lui \mathcal{P} .

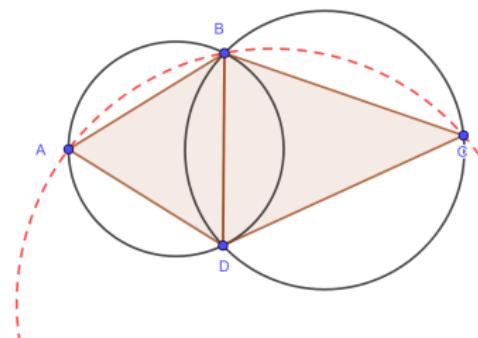


Pentru punctele A, B, C, D din figură, triangularea construită este o triangulare Delaunay. În schimb, triunghiul ΔABC nu poate participa la realizarea unei triangulări Delaunay, deoarece D este în interiorul cercului circumscris acestuia.

- ▶ **Teoremă.** O triangulare este legală dacă și numai dacă este o triangulare Delaunay.
- ▶ **Teoremă.** Orice triangulare unghiular optimă este o triangulare Delaunay. Pentru mulțimile în poziție generală (nu există patru puncte conciclice), orice triangulare Delaunay maximizează cel mai mic unghi, comparativ cu toate triangulările lui \mathcal{P} .

Legătura cu triangulările legale / unghiular optime

- ▶ **Propoziție.** Fie \mathcal{T} o triangulare a lui \mathcal{P} . Atunci \mathcal{T} este o triangulare Delaunay dacă și numai dacă pentru orice triunghi din \mathcal{T} cercul circumscris nu conține în interiorul său niciun punct al lui \mathcal{P} .



Pentru punctele A, B, C, D din figură, triangularea construită este o triangulare Delaunay. În schimb, triunghiul ΔABC nu poate participa la realizarea unei triangulări Delaunay, deoarece D este în interiorul cercului circumscris acestuia.

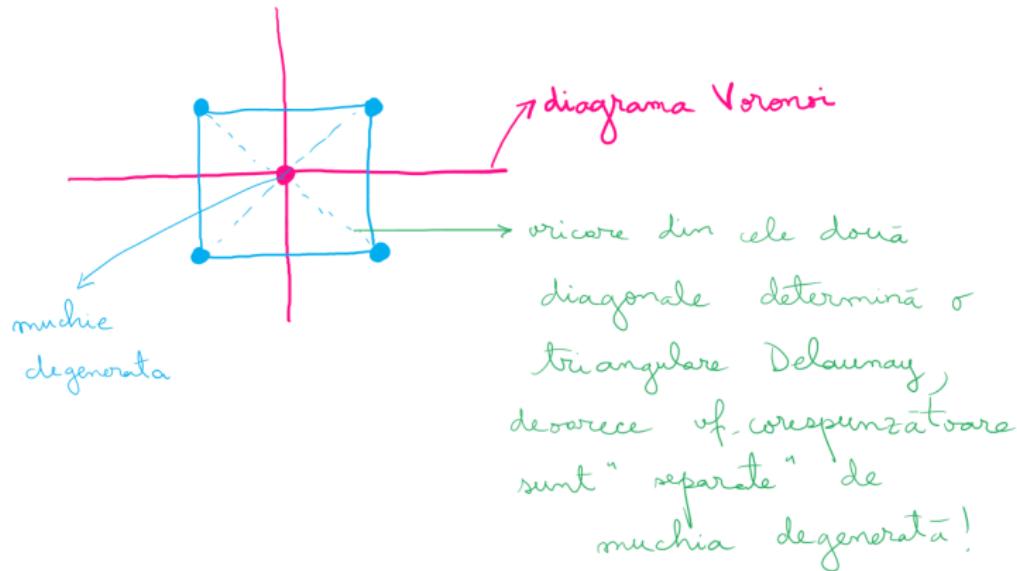
- ▶ **Teoremă.** O triangulare este legală dacă și numai dacă este o triangulare Delaunay.
- ▶ **Teoremă.** Orice triangulare unghiular optimă este o triangulare Delaunay. Pentru mulțimile în poziție generală (nu există patru puncte conciclice), orice triangulare Delaunay maximizează cel mai mic unghi, comparativ cu toate triangulările lui \mathcal{P} .
- ▶ Alte link-uri: <http://www.cs.cornell.edu/info/people/chew/Delaunay.html>
<http://cgm.cs.mcgill.ca/~godfried/teaching/projects.pr.98/tesson/taxi/taxivoro.html>

Legătura cu triangulările legale / unghiular optime - cazul unui pătrat

Întrebare: Cum “funcționează” această construcție când punctele din \mathcal{P} sunt (de exemplu) vârfurile unui pătrat?

Legătura cu triangulările legale / unghiular optime - cazul unui pătrat

Întrebare: Cum "funcționează" această construcție când punctele din \mathcal{P} sunt (de exemplu) vârfurile unui pătrat?



Algoritmul lui Fortune [1987]

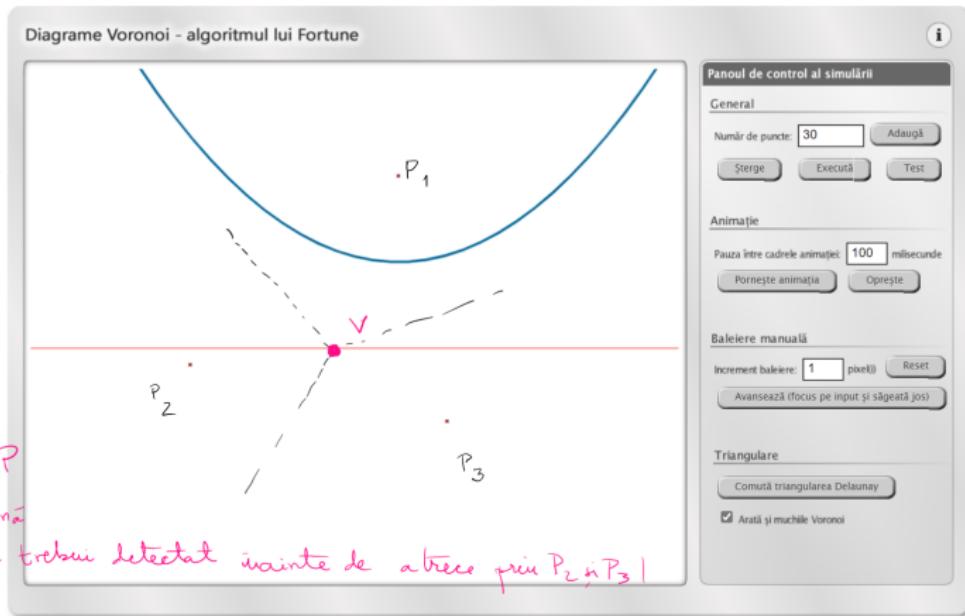
- ▶ Metodă clasică (și eficientă) de determinare a diagramei Voronoi pentru o mulțime de puncte din planul \mathbb{R}^2 . Complexitate: $O(n \log n)$.
Detalii:
<http://www.ams.org/samplings/feature-column/fcarc-voronoi>.
Suport vizual disponibil.

Algoritmul lui Fortune [1987]

- ▶ Metodă clasică (și eficientă) de determinare a diagramei Voronoi pentru o mulțime de puncte din planul \mathbb{R}^2 . Complexitate: $O(n \log n)$.
Detalii:
<http://www.ams.org/samplings/feature-column/fcarc-voronoi>.
Suport vizual disponibil.
- ▶ **Principiu (paradigmă):** sweep line / dreaptă de baleiere.

Adaptarea paradigmei dreptei de baleiere - problematizare

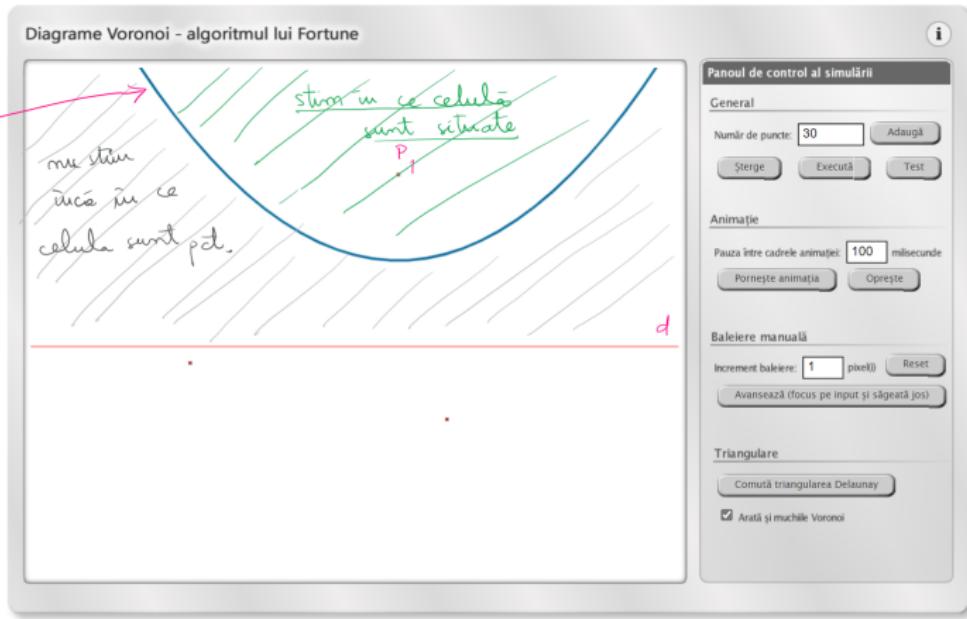
Inconvenient: la întâlnirea unui vîrf al diagramei, dreapta de baleiere nu a întâlnit încă toate siturile (punkte din \mathcal{P}) care determină acest vîrf!



la trecearea
prin vîrful
V al diagramei
Voronoi,
dreapta de
baleiere
nu a
întâlnit
(încă) toate
punktele din \mathcal{P}
care îl determină
(în figura ar trebui detectat înainte de a trece prin P_2 și P_3)

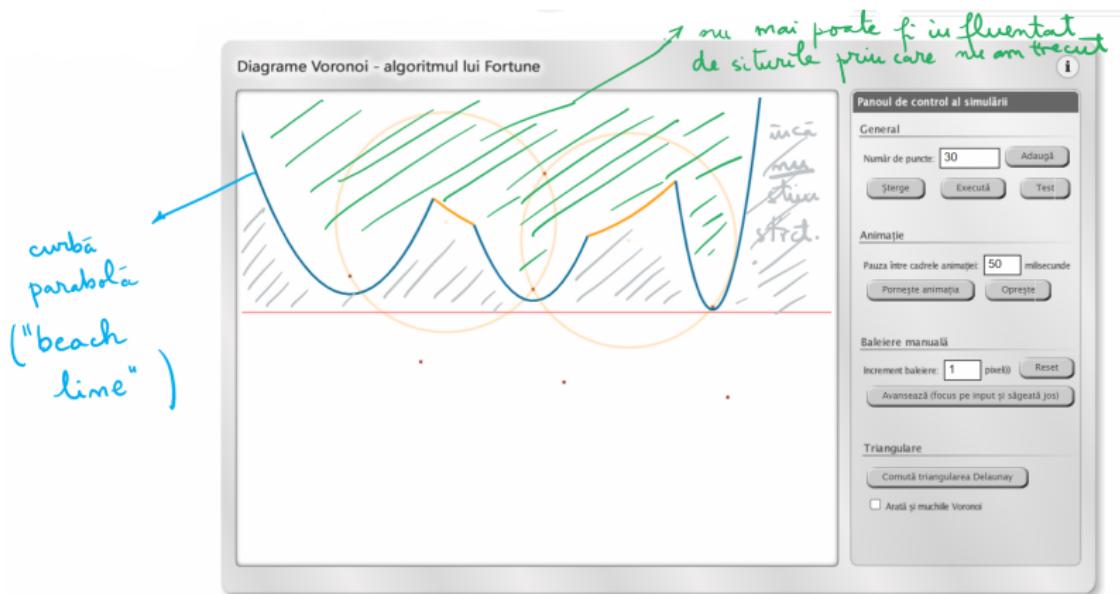
Adaptarea paradigmăi dreptei de baleiere

Adaptare: nu reținem informația legată de intersecția dintre dreapta de baleiere și diagramă, ci doar informația legată de partea diagramei care nu mai poate fi influențată de punctele situate de dincolo de dreapta de baleiere.



Adaptarea paradigmăi dreptei de baleiere - curba parabolică

Din punct de vedere practic, apare o reuniune de arce de parabolă (**curbă parabolică**), ceea ce este situat deasupra acestei curbe nu mai poate fi influențat de evenimentele nedetectate.



Despre curba parabolică (I)

- ▶ **Curba parabolică (*beach line*):**

Despre curba parabolică (I)

► Curba parabolică (*beach line*):

- Este o reuniune de arce de parabolă.
- Un punct de pe curba parabolică este egal depărtat de situl care determină arcul de parabolă și dreapta de baleiere. Presupunem că dreapta de baleiere d are ecuația $y = 0$, iar situl P_i , situat deasupra lui d (adică $y_i > 0$) are coordonatele (x_i, y_i) . Locul geometric al punctelor egal depărtate de d și P_i are ecuația

$$y = \frac{1}{2y_i}x^2 - \frac{x_i}{y_i}x + \frac{x_i^2 + y_i^2}{2y_i}.$$

Despre curba parabolică (I)

► Curba parabolică (*beach line*):

- Este o reuniune de arce de parabolă.
- Un punct de pe curba parabolică este egal depărtat de situl care determină arcul de parabolă și dreapta de baleiere. Presupunem că dreapta de baleiere d are ecuația $y = 0$, iar situl P_i , situat deasupra lui d (adică $y_i > 0$) are coordonatele (x_i, y_i) . Locul geometric al punctelor egal depărtate de d și P_i are ecuația

$$y = \frac{1}{2y_i}x^2 - \frac{x_i}{y_i}x + \frac{x_i^2 + y_i^2}{2y_i}.$$

- Punctele de racord ale arcelor de parabolă aparțin muchiilor diagramei Voronoi;

Despre curba parabolică (I)

► Curba parabolică (*beach line*):

- Este o reuniune de arce de parabolă.
- Un punct de pe curba parabolică este egal depărtat de situl care determină arcul de parabolă și dreapta de baleiere. Presupunem că dreapta de baleiere d are ecuația $y = 0$, iar situl P_i , situat deasupra lui d (adică $y_i > 0$) are coordonatele (x_i, y_i) . Locul geometric al punctelor egal depărtate de d și P_i are ecuația

$$y = \frac{1}{2y_i}x^2 - \frac{x_i}{y_i}x + \frac{x_i^2 + y_i^2}{2y_i}.$$

- Punctele de racord ale arcelor de parabolă aparțin muchiilor diagramei Voronoi;
- Curba parabolică este x -monotonă, adică orice dreaptă verticală o intersectează *exact* într-un punct (la ce folosește această proprietate?).

Despre curba parabolică (II)

- Modificarea curbei parabolice:

Despre curba parabolică (II)

- ▶ Modificarea curbei parabolice:
 - ▶ **Site event / eveniment de tip locație.** (i) La trecerea printr-un sit apare un arc de parabolă (care "la început" este degenerat) și, reciproc, apariția unui nou arc este posibilă doar la trecerea printr-un sit. (ii) În consecință, la un moment fixat, curba parabolică are maxim $(2n - 1)$ arce.

Despre curba parabolică (II)

► Modificarea curbei parabolice:

- **Site event / eveniment de tip locație.** (i) La trecerea printr-un sit apare un arc de parabolă (care “la început” este degenerat) și, reciproc, apariția unui nou arc este posibilă doar la trecerea printr-un sit. (ii) În consecință, la un moment fixat, curba parabolică are maxim $(2n - 1)$ arce.
- **Circle event / eveniment de tip cerc.** (i) La întâlnirea punctului inferior al unui cerc care trece prin cel puțin trei situri și este tangent la dreapta de baleiere dispare un arc de parabolă și, reciproc, arcele de parabolă dispar doar la acest tip de evenimente. (ii) Un eveniment de tip cerc este dat de trei arce de parabolă consecutive de pe curba parabolică, deci trebuie testate toate tripletele consecutive de arce, pe măsură ce ele apar. (iii) Un astfel de eveniment este asociat unui vîrf al diagramei Voronoi. (iv) Există triplete de arce consecutive (muchiile ale diagramei Voronoi) pentru care muchiile nu se întâlnesc. (v) Unele evenimente de tip cerc detectate nu au loc.

Adaptarea paradigmăi dreptei de baleiere - formalizare

- ▶ **Statut:** structura curbei parabolice (succesiunea de arce de parabolă); este modificat de două tipuri de evenimente

Adaptarea paradigmăi dreptei de baleiere - formalizare

- ▶ **Statut:** structura curbei parabolice (succesiunea de arce de parabolă); este modificat de două tipuri de evenimente
- ▶ **Evenimente:**
 - ▶ site event / eveniment de tip locație: întâlnirea unui sit, adică a unui punct din mulțimea \mathcal{P} (apare un arc de parabolă)
 - ▶ circle event / eveniment de tip cerc: întâlnirea unui “punct inferior” al unui cerc care trece prin cel puțin trei situri, tangent la dreapta de baleiere (dispare un arc de parabolă) - **vârf al diagramei Voronoi**

Structuri de date utilizate

- **Diagrama Voronoi:** listă dublu înlănțuită DCEL \mathcal{D}

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlăncuită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlăncuită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.
- ▶ **Statut:** Structura curbei parabolice - arbore de căutare binar echilibrat \mathcal{T} .

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlăncuită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.
- ▶ **Statut:** Structura curbei parabolice - arbore de căutare binar echilibrat \mathcal{T} .
 - ▶ **pe frunze:** siturile care au arce active, la ajungerea într-un sit se inserează un arc, iar la un eveniment de tip cerc se șterge un arc;

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlăncuită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.
- ▶ **Statut:** Structura curbei parabolice - arbore de căutare binar echilibrat \mathcal{T} .
 - ▶ **pe frunze:** siturile care au arce active, la ajungerea într-un sit se inserează un arc, iar la un eveniment de tip cerc se stergă un arc;
 - ▶ **în nodurile interne:** punctele de racord ale arcelor de parabolă (memorate simbolic) - corespund muchiilor diagramei Voronoi

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlăncuită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.
- ▶ **Statut:** Structura curbei parabolice - arbore de căutare binar echilibrat \mathcal{T} .
 - ▶ **pe frunze:** siturile care au arce active, la ajungerea într-un sit se inserează un arc, iar la un eveniment de tip cerc se stergă un arc;
 - ▶ **în nodurile interne:** punctele de racord ale arcelor de parabolă (memorate simbolic) - corespund muchiilor diagramei Voronoi
 - ▶ pointeri (eventual nuli) de la frunze către evenimentele de tip cerc; de la nodurile interne către muchiile diagramei Voronoi

Structuri de date utilizate

- ▶ **Diagrama Voronoi:** listă dublu înlănită DCEL \mathcal{D}
- ▶ **Evenimente:** coadă de priorități / evenimente \mathcal{Q} și arbore de căutare binar echilibrat. Sunt reținute evenimentele de tip sit, precum și evenimentele de tip cerc - detectate pe parcursul algoritmului.
- ▶ **Statut:** Structura curbei parabolice - arbore de căutare binar echilibrat \mathcal{T} .
 - ▶ **pe frunze:** siturile care au arce active, la ajungerea într-un sit se inserează un arc, iar la un eveniment de tip cerc se șterge un arc;
 - ▶ **în nodurile interne:** punctele de racord ale arcelor de parabolă (memorate simbolic) - corespund muchiilor diagramei Voronoi
 - ▶ pointeri (eventual nuli) de la frunze către evenimentele de tip cerc; de la nodurile interne către muchiile diagramei Voronoi
- ▶ **Analiză preliminară a complexității:** (i) Câte evenimente sunt în total? (ii) Care este complexitatea-timp pentru a actualiza coada de evenimente la un eveniment de tip cerc? (iii) Care este complexitatea-timp (totală) pentru a actualiza coada de evenimente? (iv) Întrebări similare pentru statut.

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlăncuită \mathcal{D} .

1. Inițializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlăncuită $\mathcal{D} \leftarrow \emptyset$.

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlăncuită \mathcal{D} .

1. Inițializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlăncuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlăncuită \mathcal{D} .

1. Initializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlăncuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlănțuită \mathcal{D} .

1. Inițializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlănțuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}
4. **if** evenimentul **ev** este un eveniment de tip sit

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlănțuită \mathcal{D} .

1. Initializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlănțuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}
4. **if** evenimentul **ev** este un eveniment de tip sit
5. **then** **PROCESSEVSIT**(p_i), cu $p_i = \text{ev}$

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlănțuită \mathcal{D} .

1. Initializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlănțuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}
4. **if** evenimentul **ev** este un eveniment de tip sit
5. **then** $\text{PROCESSEvSIT}(p_i)$, cu $p_i = \text{ev}$
6. **else** $\text{PROCESSEvCERC}(\gamma)$, cu $\gamma = \text{arc}(\text{ev}) \in \mathcal{T}$

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlănțuită \mathcal{D} .

1. Initializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlănțuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}
4. **if** evenimentul **ev** este un eveniment de tip sit
5. **then** **PROCESSEvSIT**(p_i), cu $p_i = \text{ev}$
6. **else** **PROCESSEvCERC**(γ), cu $\gamma = \text{arc}(\text{ev}) \in \mathcal{T}$
7. Nodurile interne încă prezente în \mathcal{T} corespund semidreptelor diagramei Voronoi.
Consideră un *bounding box* care conține toate vârfurile diagramei Voronoi în interiorul său și leagă semidreptele de acest *bounding box*, prin actualizarea corespunzătoare a lui \mathcal{D} .

Algoritmul

Input. O mulțime de situri $\mathcal{P} = \{p_1, \dots, p_n\}$ de situri în plan.

Output. Diagrama Voronoi $\text{Vor}(\mathcal{P})$ în interiorul unui *bounding box*, descrisă printr-o listă dublu înlănțuită \mathcal{D} .

1. Initializări: coada de evenimente $\mathcal{Q} \leftarrow \mathcal{P}$ (preprocesare: ordonare după y), statut (arbore balansat) $\mathcal{T} \leftarrow \emptyset$; listă dublu înlănțuită $\mathcal{D} \leftarrow \emptyset$.
2. **while** $\mathcal{Q} \neq \emptyset$
3. **do** elimină evenimentul cu cel mai mare y din \mathcal{Q}
4. **if** evenimentul **ev** este un eveniment de tip sit
5. **then** **PROCESSEvSIT**(p_i), cu $p_i = \text{ev}$
6. **else** **PROCESSEvCERC**(γ), cu $\gamma = \text{arc}(\text{ev}) \in \mathcal{T}$
7. Nodurile interne încă prezente în \mathcal{T} corespund semidreptelor diagramei Voronoi.
Consideră un *bounding box* care conține toate vârfurile diagramei Voronoi în interiorul său și leagă semidreptele de acest *bounding box*, prin actualizarea corespunzătoare a lui \mathcal{D} .
8. Traversează muchiile pentru a adăuga celulele diagramei și pointeri corespunzători.

Procedura PROCESSEvSIT (p_i)

1. Dacă \mathcal{T} este vidă, inserează p_i și revine, dacă nu continuă cu 2.–5.

Procedura PROCESSEvSIT (p_i)

1. Dacă \mathcal{T} este vidă, inserează p_i și revine, dacă nu continuă cu 2.–5.
2. Caută în \mathcal{T} arcul α situat deasupra lui p_i . Dacă frunza reprezentând α are un pointer către un eveniment de tip cerc **ev** din \mathcal{Q} , atunci **ev** este o alarmă falsă și trebuie șters.

Procedura PROCESSEvSIT (p_i)

1. Dacă \mathcal{T} este vidă, inserează p_i și revine, dacă nu continuă cu 2.–5.
2. Caută în \mathcal{T} arcul α situat deasupra lui p_i . Dacă frunza reprezentând α are un pointer către un eveniment de tip cerc **ev** din \mathcal{Q} , atunci **ev** este o alarmă falsă și trebuie șters.
3. Înlocuiește frunza lui \mathcal{T} care reprezintă α cu un subarbore cu trei frunze: cea din mijloc reține situl p_i și celelalte două situl p_j asociat lui α . Memorează perechile reprezentând punctele de racord în două noduri interne. Efectuează rebalansări în \mathcal{T} , dacă este necesar.

Procedura PROCESSEvSIT (p_i)

1. Dacă \mathcal{T} este vidă, inserează p_i și revine, dacă nu continuă cu 2.–5.
2. Caută în \mathcal{T} arcul α situat deasupra lui p_i . Dacă frunza reprezentând α are un pointer către un eveniment de tip cerc **ev** din \mathcal{Q} , atunci **ev** este o alarmă falsă și trebuie șters.
3. Înlocuiește frunza lui \mathcal{T} care reprezintă α cu un subarbore cu trei frunze: cea din mijloc reține situl p_i și celelalte două situl p_j asociat lui α . Memorează perechile reprezentând punctele de racord în două noduri interne. Efectuează rebalansări în \mathcal{T} , dacă este necesar.
4. Generează noi înregistrări de tip semi-muchie în structura diagramei Voronoi (\mathcal{D}), pentru muchiile care separă celulele $V(p_i)$ și $V(p_j)$, corespunzând celor două noi puncte de racord.

Procedura PROCESSEvSIT (p_i)

1. Dacă \mathcal{T} este vidă, inserează p_i și revine, dacă nu continuă cu 2.–5.
2. Caută în \mathcal{T} arcul α situat deasupra lui p_i . Dacă frunza reprezentând α are un pointer către un eveniment de tip cerc **ev** din \mathcal{Q} , atunci **ev** este o alarmă falsă și trebuie șters.
3. Înlocuiește frunza lui \mathcal{T} care reprezintă α cu un subarbore cu trei frunze: cea din mijloc reține situl p_i și celelalte două situl p_j asociat lui α . Memorează perechile reprezentând punctele de racord în două noduri interne. Efectuează rebalansări în \mathcal{T} , dacă este necesar.
4. Generează noi înregistrări de tip semi-muchie în structura diagramei Voronoi (\mathcal{D}), pentru muchiile care separă celulele $V(p_i)$ și $V(p_j)$, corespunzând celor două noi puncte de racord.
5. Verifică tripletele de arce consecutive nou create, pentru a verifica dacă muchiile corespunzătoare punctelor de racord se întâlnesc. Dacă da, inserează evenimente de tip cerc în \mathcal{Q} și adaugă pointeri de la nodurile lui \mathcal{T} la evenimentele corespunzătoare din \mathcal{Q} .

Procedura PROCESSEvCERC (γ)

1. Șterge frunza $\gamma \in \mathcal{T}$ care corespunde arcului de cerc α care dispare. Actualizează în nodurile interne perechile care corespund punctelor de racord. Efectuează rebalansări în \mathcal{T} , dacă este necesar. Șterge toate evenimentele de tip cerc care îi corespund lui α (cu ajutorul pointerilor de la predecesorul și succesorul lui γ în \mathcal{T}).

Procedura PROCESSEVERCC(γ)

1. Șterge frunza $\gamma \in \mathcal{T}$ care corespunde arcului de cerc α care dispare. Actualizează în nodurile interne perechile care corespund punctelor de racord. Efectuează rebalansări în \mathcal{T} , dacă este necesar. Șterge toate evenimentele de tip cerc care îi corespund lui α (cu ajutorul pointerilor de la predecesorul și succesorul lui γ în \mathcal{T}).
2. Adaugă centrul cercului care determină evenimentul ca înregistrare de tip vârf în \mathcal{D} . Creează înregistrări de tip semi-muchie corespunzând noului punct de racord de pe linia parabolică și asignează pointeri corespunzători.

Procedura PROCESSEVERCC(γ)

1. Șterge frunza $\gamma \in \mathcal{T}$ care corespunde arcului de cerc α care dispare. Actualizează în nodurile interne perechile care corespund punctelor de racord. Efectuează rebalansări în \mathcal{T} , dacă este necesar. Șterge toate evenimentele de tip cerc care îi corespund lui α (cu ajutorul pointerilor de la predecesorul și succesorul lui γ în \mathcal{T}).
2. Adaugă centrul cercului care determină evenimentul ca înregistrare de tip vârf în \mathcal{D} . Creează înregistrări de tip semi-muchie corespunzând noului punct de racord de pe linia parabolică și asignează pointeri corespunzători.
3. Verifică tripletele de arce consecutive nou create (care au foștii vecini ai lui α în centru), pentru a verifica dacă muchiile corespunzătoare punctelor de racord se întâlnesc. Dacă da, inserează evenimente de tip cerc în \mathcal{Q} și adaugă pointeri de la nodurile lui \mathcal{T} la evenimentele corespunzătoare din \mathcal{Q} .

Rezultate principale

- ▶ **Teoremă.** *Diagrama Voronoi a unei mulțimi de n situri poate fi determinată cu un algoritm bazat pe paradigma dreptei de baleiere având complexitate-temp $O(n \log n)$ și complexitate-spațiu $O(n)$.*

Rezultate principale

- ▶ **Teoremă.** *Diagrama Voronoi a unei mulțimi de n situri poate fi determinată cu un algoritm bazat pe paradigma dreptei de baleiere având complexitate-timp $O(n \log n)$ și complexitate-spațiu $O(n)$.*
- ▶ **Teoremă.** *Triangularea Delaunay a unei mulțimi de n situri poate fi determinată cu un algoritm bazat pe paradigma dreptei de baleiere având complexitate-timp $O(n \log n)$ și complexitate-spațiu $O(n)$.*

Scurt rezumat

- ▶ Conceptul de diagrama Voronoi. Aspecte cantitative legate de numărul de vârfuri/muchii/muchii de tip semidreaptă (și la seminar).

Scurt rezumat

- ▶ Conceptul de diagrama Voronoi. Aspecte cantitative legate de numărul de vârfuri/muchii/muchii de tip semidreaptă (și la seminar).
- ▶ Legătura cu triangulările unghiular optime.

Scurt rezumat

- ▶ Conceptul de diagrama Voronoi. Aspecte cantitative legate de numărul de vârfuri/muchii/muchii de tip semidreaptă (și la seminar).
- ▶ Legătura cu triangulările unghiular optime.
- ▶ Algoritmul lui Fortune.

Algoritmi avansați

C5 - Elemente de programare liniară

Mihai-Sorin Stupariu

Sem. al II-lea, 2022-2023

Motivație: turnarea pieselor în matrițe

Intersecții de semiplane - abordare cantitativă

Dualitate

Intersecții de semiplane - abordare calitativă. Programare liniară

Subdiviziuni planare. DCEL

Turnarea pieselor în matrițe



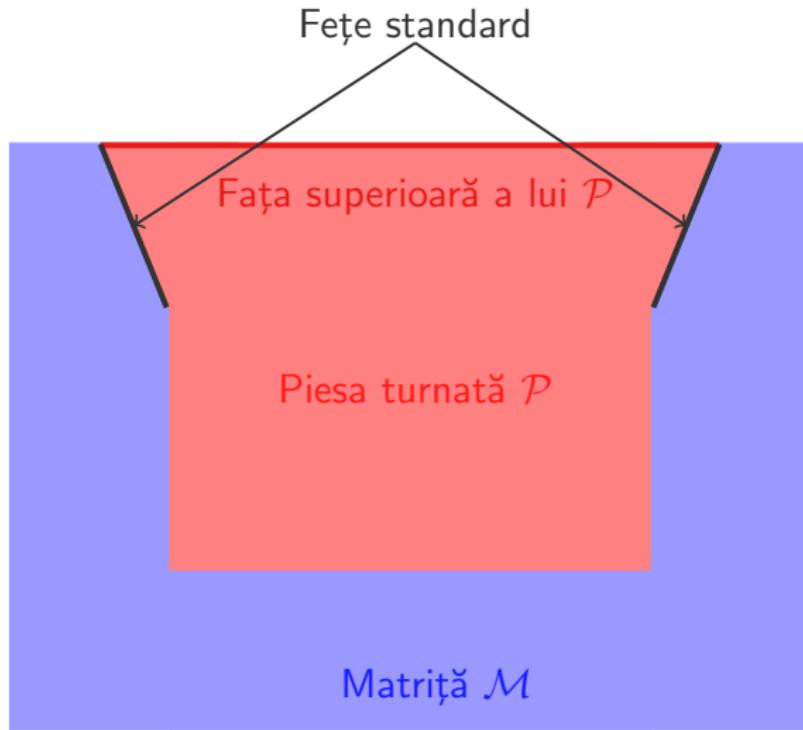
Turnarea pieselor în matrițe



Turnarea pieselor în matrițe



Turnarea pieselor în mătrițe



Problematizare

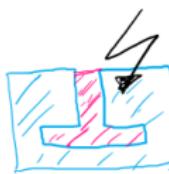
- ▶ Turnarea pieselor în matrițe și extragerea lor fără distrugerea matriței.

Problematizare

- ▶ Turnarea pieselor în matrițe și extragerea lor fără distrugerea matriței.
- ▶ Neajunsuri: unele obiecte pot rămâne blocate; există obiecte pentru care nu există o matriță adecvată.

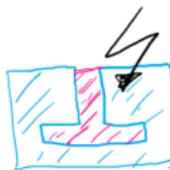
Problematizare

- ▶ Turnarea pieselor în matrițe și extragerea lor fără distrugerea matriței.
- ▶ Neajunsuri: unele obiecte pot rămâne blocate; există obiecte pentru care nu există o matriță adecvată.



Problematizare

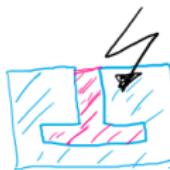
- ▶ Turnarea pieselor în matrițe și extragerea lor fără distrugerea matriței.
- ▶ Neajunsuri: unele obiecte pot rămâne blocați; există obiecte pentru care nu există o matriță adecvată.



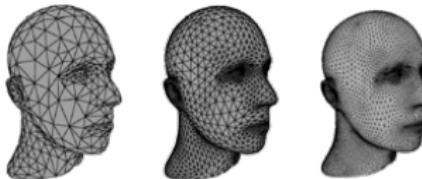
- ▶ **Problema studiată.** Dat un obiect, există o matriță din care să poată fi extras (și dacă da, cu un algoritm eficient?)

Problematizare

- ▶ Turnarea pieselor în matrițe și extragerea lor fără distrugerea matriței.
- ▶ Neajunsuri: unele obiecte pot rămâne blocați; există obiecte pentru care nu există o matriță adecvată.



- ▶ **Problema studiată.** Dat un obiect, există o matriță din care să poată fi extras (și dacă da, cu un algoritm eficient?)



Sursa: <https://www.graphics.rwth-aachen.de/publication/03149/>

Convenții

- Obiectele: **poliedrale**.

Convenții

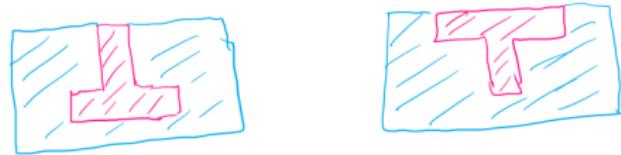
- ▶ Obiectele: **poliedrale**.
- ▶ Matrițele: formate dintr-o singură piesă; fiecărui obiect \mathcal{P} îi este asociată o matriță $\mathcal{M}_{\mathcal{P}}$

Convenții

- ▶ Obiectele: **poliedrale**.
- ▶ Matrițele: formate dintr-o singură piesă; fiecărui obiect \mathcal{P} îi este asociată o matriță $\mathcal{M}_{\mathcal{P}}$
- ▶ Obiectul: extras printr-o singură translație (sau o succesiune de translații)

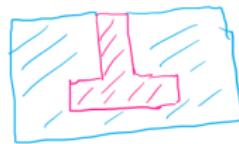
Convenții

- ▶ Obiectele: **poliedrale**.
- ▶ Matrițele: formate dintr-o singură piesă; fiecărui obiect \mathcal{P} îi este asociată o matriță $\mathcal{M}_{\mathcal{P}}$
- ▶ Obiectul: extras printr-o singură translație (sau o succesiune de translații)
- ▶ **Alegerea orientării:** diverse orientări ale obiectului pot genera diverse matrițe...

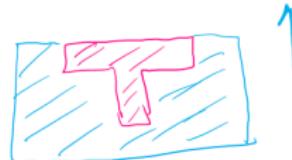


Convenții

- ▶ Obiectele: **poliedrale**.
- ▶ Matrițele: formate dintr-o singură piesă; fiecărui obiect \mathcal{P} îi este asociată o matriță $\mathcal{M}_{\mathcal{P}}$
- ▶ Obiectul: extras printr-o singură translație (sau o succesiune de translații)
- ▶ **Alegerea orientării:** diverse orientări ale obiectului pot genera diverse matrițe...



- ▶ ... astfel încât doar în unele configurații este posibilă extragerea obiectului



Terminologie și convenții

- ▶ **Față superioară:** prin convenție, obiectele au (cel puțin) o față superioară (este orizontală, este singura care nu este adjacente cu matrița). Celelalte fețe: **standard**; orice față standard f a obiectului corespunde unei fețe standard \hat{f} a matriței.

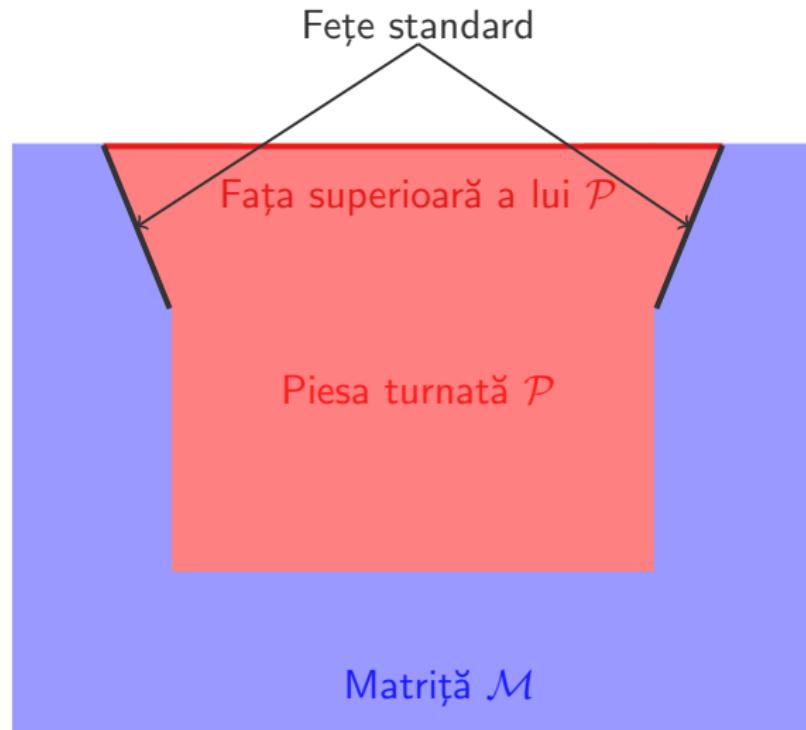
Terminologie și convenții

- ▶ **Față superioară:** prin convenție, obiectele au (cel puțin) o față superioară (este orizontală, este singura care nu este adjacente cu matrița). Celelalte fețe: **standard**; orice față standard f a obiectului corespunde unei fețe standard \hat{f} a matriței.
- ▶ **Obiect care poate fi turnat (castable):** există o orientare pentru care acesta poate fi turnat și apoi extras printr-o translație (succesiune de translații): *direcție admisibilă*.

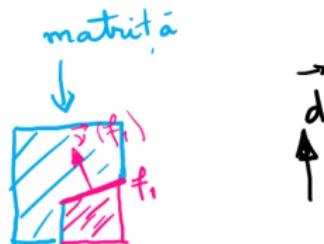
Terminologie și convenții

- ▶ **Fața superioară:** prin convenție, obiectele au (cel puțin) o față superioară (este orizontală, este singura care nu este adjacente cu matrița). Celelalte fețe: **standard**; orice față standard f a obiectului corespunde unei fețe standard \hat{f} a matriței.
- ▶ **Obiect care poate fi turnat (castable):** există o orientare pentru care acesta poate fi turnat și apoi extras printr-o translație (succesiune de translații): *direcție admisibilă*.
- ▶ **Convenții:** Matrița este paralelipipedică și are o cavitate corespunzătoare obiectului; fața superioară a obiectului (și a matriței) este perpendiculară cu planul Oxy .

Turnarea pieselor în matrițe



Descrierea proprietății de a putea extrage o piesă într-o direcție dată



$\vec{v}(f_1)$ = normala exterioară la f_1

fată \hat{f}_1 a mătriței blochează
extragerea în direcția $\vec{d} \Leftrightarrow$
 $m(\vec{v}(f_1), \vec{d}) < 90^\circ$
 $\Leftrightarrow \cos(\vec{v}(f_1), \vec{d}) > 0$



$\vec{v}(f_2)$

\hat{f}_2 nu blochează

$$\Leftrightarrow m(\vec{v}(f_2), \vec{d}) \geq 90^\circ$$

$$\Leftrightarrow \cos(\vec{v}(f_2), \vec{d}) \leq 0$$

Această condiție trebuie verificată pînă tate fetele!

Detaliere (scriere în coordonate)

$$\text{Dacă } v, w \in \mathbb{R}^3 : \cos(\varphi(v, w)) = \frac{\langle v, w \rangle}{\|v\| \|w\|} \left(\begin{array}{l} \langle v, w \rangle = \\ v_1 w_1 + v_2 w_2 + v_3 w_3 \end{array} \right)$$

Cum vrem să extragem obiectul "în sus"; f.r.g. putem pp. că $\vec{d} = (d_x, d_y, 1)$ (de ce?)

Trebuie să fixăm o fază a obiectului; $\vec{r}(f) = (x_x, x_y, x_z)$

Japtul că fază \hat{f} a mătriței nu blochează extragerea în direcția $\vec{d} \Leftrightarrow$

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

$$\boxed{x_x \cdot d_x + x_y \cdot d_y + x_z \leq 0} \quad (\star_f)$$

Fixată $f \rightarrow (x_x, x_y, x_z)$ conțină $\vec{d} (d_x, d_y)$ a.c. să fie verificată (\star_f)

$\boxed{(\star_f)}: \text{inequătate care descrie un semiplan}$

Sinteză - fundamente geometrice

- ▶ **Condiție necesară:** direcția de extragere \vec{d} trebuie să aibă componenta z pozitivă
- ▶ **În general:** o față standard \hat{f} a matriței (corespunzătoare unei fețe f a piesei) pentru care unghiul dintre normala exterioară $\vec{n}(f)$ la față f și \vec{d} este mai mic de 90° împiedică translația în direcția \vec{d}

Sinteză - fundamente geometrice

- ▶ **Condiție necesară:** direcția de extragere \vec{d} trebuie să aibă componenta z pozitivă
- ▶ **În general:** o față standard \hat{f} a matriței (corespunzătoare unei fețe f a piesei) pentru care unghiul dintre normala exteroară $\vec{\nu}(f)$ la față f și \vec{d} este mai mic de 90° împiedică translația în direcția \vec{d}
- ▶ **Propoziție.** *Un poliedru \mathcal{P} poate fi extras din matrița sa $M_{\mathcal{P}}$ prin translație în direcția \vec{d} dacă și numai dacă \vec{d} face un unghi de cel puțin 90° cu normala exteroară a fiecărei fețe standard a lui \mathcal{P} .*
- ▶ **Reformulare.** Dat \mathcal{P} , trebuie găsită o direcție \vec{d} astfel încât, pentru fiecare față standard f , unghiul dintre \vec{d} și $\vec{\nu}(f)$ să fie cel puțin 90° .

Sinteză - fundamente geometrice

- ▶ **Condiție necesară:** direcția de extragere \vec{d} trebuie să aibă componenta z pozitivă
- ▶ **În general:** o față standard \hat{f} a matriței (corespunzătoare unei fețe f a piesei) pentru care unghiul dintre normala exteroară $\vec{\nu}(f)$ la față f și \vec{d} este mai mic de 90° împiedică translația în direcția \vec{d}
- ▶ **Propoziție.** *Un poliedru \mathcal{P} poate fi extras din matrița sa $M_{\mathcal{P}}$ prin translație în direcția \vec{d} dacă și numai dacă \vec{d} face un unghi de cel puțin 90° cu normala exteroară a fiecărei fețe standard a lui \mathcal{P} .*
- ▶ **Reformulare.** Dat \mathcal{P} , trebuie găsită o direcție \vec{d} astfel încât, pentru fiecare față standard f , unghiul dintre \vec{d} și $\vec{\nu}(f)$ să fie cel puțin 90° .
- ▶ **Analitic - pentru o față:** fiecare față definește un semiplan, i.e. dată o față standard f a poliedrului / matriței, a găsi o direcție admisibilă revine la a rezolva o inecuație ($*_f$), care corespunde unui semiplan.

Sinteză - fundamente geometrice

- ▶ **Condiție necesară:** direcția de extragere \vec{d} trebuie să aibă componenta z pozitivă
- ▶ **În general:** o față standard \hat{f} a matriței (corespunzătoare unei fețe f a piesei) pentru care unghiul dintre normala exteroară $\vec{n}(f)$ la față f și \vec{d} este mai mic de 90° împiedică translația în direcția \vec{d}
- ▶ **Propoziție.** *Un poliedru \mathcal{P} poate fi extras din matrița sa $M_{\mathcal{P}}$ prin translație în direcția \vec{d} dacă și numai dacă \vec{d} face un unghi de cel puțin 90° cu normala exteroară a fiecărei fețe standard a lui \mathcal{P} .*
- ▶ **Reformulare.** Dat \mathcal{P} , trebuie găsită o direcție \vec{d} astfel încât, pentru fiecare față standard f , unghiul dintre \vec{d} și $\vec{n}(f)$ să fie cel puțin 90° .
- ▶ **Analitic - pentru o față:** fiecare față definește un semiplan, i.e. dată o față standard f a poliedrului / matriței, a găsi o direcție admisibilă revine la a rezolva o inecuație $(*_f)$, care corespunde unui semiplan.
- ▶ **Analitic - toate fețele:** Fie \mathcal{P} un poliedru; față superioară fixată, paralelă cu planul Oxy . Considerăm matrița asociată și toate fețele matriței (i.e. toate fețele standard ale poliedrului). A determina o direcție admisibilă revine la a determina o direcție care verifică toate inegalitățile de tip $(*)$, deci un sistem de inecuații.

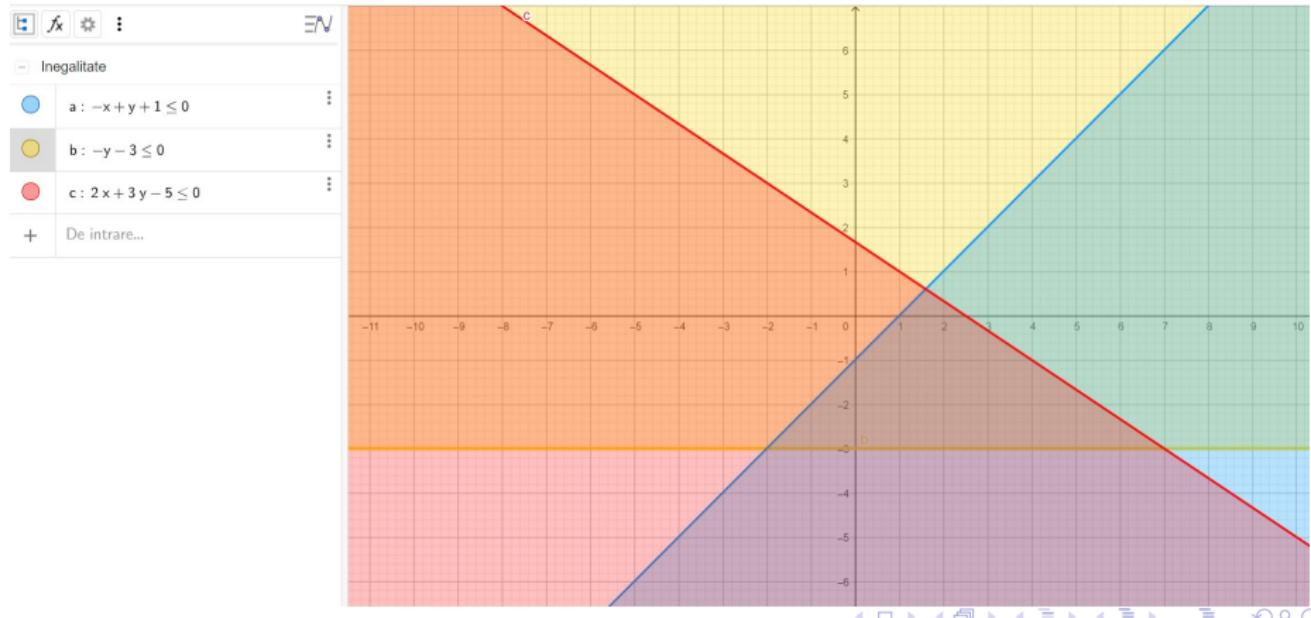
Sinteză - fundamente geometrice

- ▶ **Condiție necesară:** direcția de extragere \vec{d} trebuie să aibă componenta z pozitivă
- ▶ **În general:** o față standard \hat{f} a matriței (corespunzătoare unei fețe f a piesei) pentru care unghiul dintre normala exteroară $\vec{n}(f)$ la față f și \vec{d} este mai mic de 90° împiedică translația în direcția \vec{d}
- ▶ **Propoziție.** *Un poliedru \mathcal{P} poate fi extras din matrița sa $M_{\mathcal{P}}$ prin translație în direcția \vec{d} dacă și numai dacă \vec{d} face un unghi de cel puțin 90° cu normala exteroară a fiecărei fețe standard a lui \mathcal{P} .*
- ▶ **Reformulare.** Dat \mathcal{P} , trebuie găsită o direcție \vec{d} astfel încât, pentru fiecare față standard f , unghiul dintre \vec{d} și $\vec{n}(f)$ să fie cel puțin 90° .
- ▶ **Analitic - pentru o față:** fiecare față definește un semiplan, i.e. dată o față standard f a poliedrului / matriței, a găsi o direcție admisibilă revine la a rezolva o inecuație $(*_f)$, care corespunde unui semiplan.
- ▶ **Analitic - toate fețele:** Fie \mathcal{P} un poliedru; față superioară fixată, paralelă cu planul Oxy . Considerăm matrița asociată și toate fețele matriței (i.e. toate fețele standard ale poliedrului). A determina o direcție admisibilă revine la a determina o direcție care verifică toate inegalitățile de tip $(*)$, deci un sistem de inecuații.
- ▶ **Concluzie:** Pentru a stabili dacă există o direcție admisibilă, trebuie stabilit dacă o intersecție de semiplane este nevidă.

Exemple

1. Intersecția semiplanelor

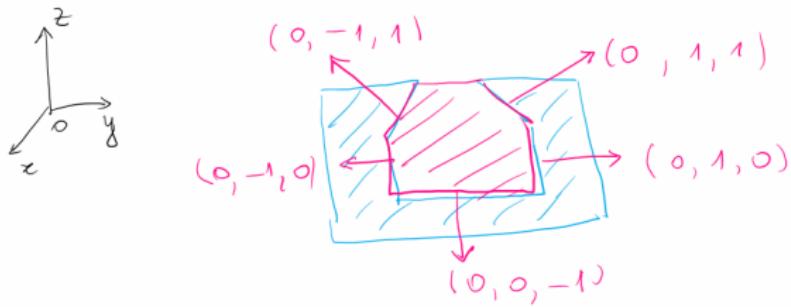
$$-x + y + 1 \leq 0; \quad -y - 3 \leq 0; \quad 2x + 3y - 5 \leq 0.$$



Exemple

2 (a). Normalele exterioare ale fețelor standard sunt coliniare cu vectorii

$$(0, -1, 1), (0, 1, 1), (0, 1, 0), (0, 0, -1), (0, -1, 0).$$



$$(0, -1, 1) \rightsquigarrow 0 \cdot x + (-1) \cdot y + 1 \leq 0$$

$$(0, 1, 1) \rightsquigarrow 0 \cdot x + 1 \cdot y + 1 \leq 0$$

$$(0, 1, 0) \rightsquigarrow 0 \cdot x + 1 \cdot y + 0 \leq 0$$

$$(0, 0, -1) \rightsquigarrow 0 \cdot x + 0 \cdot y + (-1) \leq 0$$

$$(0, -1, 0) \rightsquigarrow 0 \cdot x + (-1) \cdot y + 0 \leq 0$$

sistem incompatibil,
obiectul nu poate
fi înras

$$y \geq 1$$

$$y \leq -1$$

$$y \leq 0$$

$$-1 \leq 0$$

$$y \geq 0$$

Temă

2 (b). Normalele exterioare ale fețelor standard sunt coliniare cu vectorii

$$(0, 1, 0), (0, 1, -1), (0, 0, -1), (0, -1, -1), (0, -1, 0).$$

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

- (i) **Caracterizare explicită:** Să se determine care sunt elementele (vârfuri, muchii, etc.) care determină o intersecție de semiplane.

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

- (i) **Caracterizare explicită:** Să se determine care sunt elementele (vârfuri, muchii, etc.) care determină o intersecție de semiplane.
- (ii) **Calitativ:** Să se stabilească dacă o intersecție de semiplane este nevidă.

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

- (i) **Caracterizare explicită:** Să se determine care sunt elementele (vârfuri, muchii, etc.) care determină o intersecție de semiplane.
- (ii) **Calitativ:** Să se stabilească dacă o intersecție de semiplane este nevidă.

► Rezultate: (descrise în detaliu ulterior)

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

- (i) **Caracterizare explicită:** Să se determine care sunt elementele (vârfuri, muchii, etc.) care determină o intersecție de semiplane.
- (ii) **Calitativ:** Să se stabilească dacă o intersecție de semiplane este nevidă.

► Rezultate: (descrise în detaliu ulterior)

- (i) *Intersecția unei mulțimi de n semiplane poate fi determinată cu complexitate-temp $O(n \log n)$ și folosind $O(n)$ memorie.*

Intersecții de semiplane - probleme studiate, rezultate

► Probleme studiate:

- (i) **Caracterizare explicită:** Să se determine care sunt elementele (vârfuri, muchii, etc.) care determină o intersecție de semiplane.
- (ii) **Calitativ:** Să se stabilească dacă o intersecție de semiplane este nevidă.

► Rezultate: (descrise în detaliu ulterior)

- (i) *Intersecția unei mulțimi de n semiplane poate fi determinată cu complexitate-timp $O(n \log n)$ și folosind $O(n)$ memorie.*
- (ii) *Se poate stabili cu complexitate-timp medie $O(n)$ dacă o intersecție de semiplane este nevidă.*
- (ii)' *Fie \mathcal{P} un poliedru cu n fețe. Se poate decide dacă \mathcal{P} reprezintă un obiect care poate fi turnat cu complexitate-timp medie $O(n^2)$ și folosind $O(n)$ spațiu. În caz afirmativ, o matriță și o direcție admisibilă în care poate fi extras \mathcal{P} este determinată cu aceeași complexitate-timp.*

(i) Caracterizare explicită - Formularea problemei

- ▶ Fie $\mathcal{H} = \{H_1, H_2, \dots, H_n\}$ o mulțime de semiplane din \mathbb{R}^2 ; semiplanul H_i dat de o relație de forma

$$a_i x + b_i y + c_i \leq 0$$

(i) Caracterizare explicită - Formularea problemei

- ▶ Fie $\mathcal{H} = \{H_1, H_2, \dots, H_n\}$ o mulțime de semiplane din \mathbb{R}^2 ; semiplanul H_i dat de o relație de forma

$$a_i x + b_i y + c_i \leq 0$$

- ▶ Intersecția $H_1 \cap H_2 \cap \dots \cap H_n$ este dată de un sistem de inecuații; este o mulțime poligonală convexă, mărginită de cel mult n muchii (poate fi vidă, mărginită, nemărginită,...)

(i) Caracterizare explicită - Formularea problemei

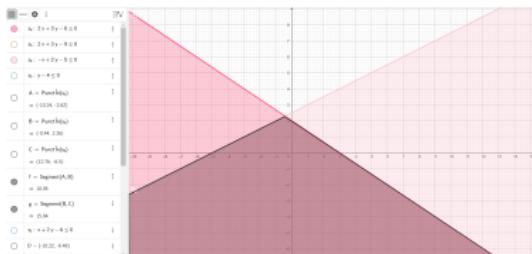
- ▶ Fie $\mathcal{H} = \{H_1, H_2, \dots, H_n\}$ o mulțime de semiplane din \mathbb{R}^2 ; semiplanul H_i dat de o relație de forma

$$a_i x + b_i y + c_i \leq 0$$

- ▶ Intersecția $H_1 \cap H_2 \cap \dots \cap H_n$ este dată de un sistem de inecuații; este o mulțime poligonală convexă, mărginită de cel mult n muchii (poate fi vidă, mărginită, nemărginită,...)
- ▶ De clarificat: date n semiplane, care sunt cele care contribuie **efectiv** la determinarea intersecției acestora? (v. slide-ul următor)

Semiplane - intersecții

Când determinăm o intersecție de semiplane, nu sunt neapărat relevante toate semiplanele. În figura de mai jos sunt considerate cinci semiplane s_1, s_2, s_3, s_4, s_5 dintre care relevante pentru intersecție sunt doar s_2 și s_4 .



Dualitate – motivație euristică

- De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2
- ▶ De câte informații (numerice) este nevoie pentru a indica **o dreaptă în plan?**

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2
- ▶ De câte informații (numerice) este nevoie pentru a indica **o dreaptă în plan?**
- ▶ 2

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2
- ▶ De câte informații (numerice) este nevoie pentru a indica **o dreaptă în plan?**
- ▶ 2
- ▶ Există o modalitate naturală de a stabili o corespondență între puncte și drepte?

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2
- ▶ De câte informații (numerice) este nevoie pentru a indica **o dreaptă în plan?**
- ▶ 2
- ▶ Există o modalitate naturală de a stabili o corespondență între puncte și drepte?
- ▶ **DA: dualitate**

Dualitate – motivație euristică

- ▶ De câte informații (numerice) este nevoie pentru a indica **un punct în plan?**
- ▶ 2
- ▶ De câte informații (numerice) este nevoie pentru a indica **o dreaptă în plan?**
- ▶ 2
- ▶ Există o modalitate naturală de a stabili o corespondență între puncte și drepte?
- ▶ **DA: dualitate**
- ▶ Cum se reflectă / respectă diferite proprietăți geometrice (de exemplu incidența) prin dualitate?

Dualitate – definiții

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

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

Dualitate – definiții

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

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

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

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

Dualitate – definiții

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

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

- ▶ 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 $y = \frac{x^2}{2}$.

Dualitate – proprietăți elementare

1) Păstrează incidenta

$$p \in d \Leftrightarrow d^* \in p^*$$

Exemplu

Pl. primal

$$d : (y = 2x + 1)$$

$$p = (1, 3)$$

Pl. dual

$$d^* = (2, -1)$$

$$p^* : (y = \infty - 3)$$

Dualitate – proprietăți elementare

2) Păstrează "ordinea"

p este situat deasupra dreptei d (nereverticală) \Leftrightarrow
 d^* ——— ——— p^*

Exemplu

Pl. primal

$$P = (1, 1)$$

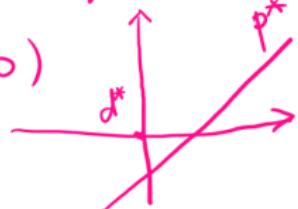
$$d: (y = 0)$$



Pl. dual

$$P^*: (y = x - 1)$$

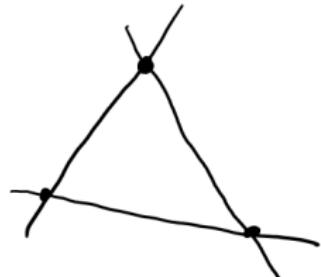
$$d^* = (0, 0)$$



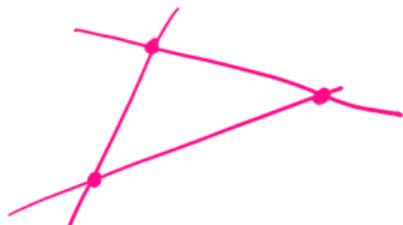
Dualitate – “dicționar” concepte și configurații

| 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 (<i>wedge</i>) |

Exemplu

Configurație primală

3 puncte necoliniare
și dreptele determinate
de ele

Configurație duală

3 drepte care nu
trec prin același punct
și punctele determinate
de ele

Semiplane inferioare și semiplane superioare

- Exemple.



Semiplane inferioare și semiplane superioare

- Exemple.



semiplan inferior



semiplan superior

Semiplane inferioare și semiplane superioare

- Exemple.



semiplan inferior



semiplan superior

- Dat un semiplan delimitat de o dreaptă neverticală

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

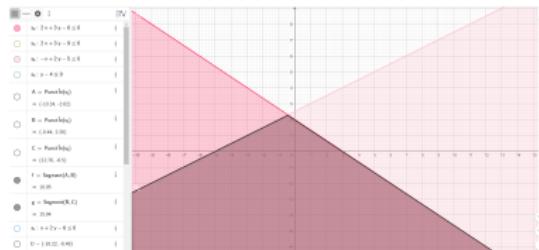
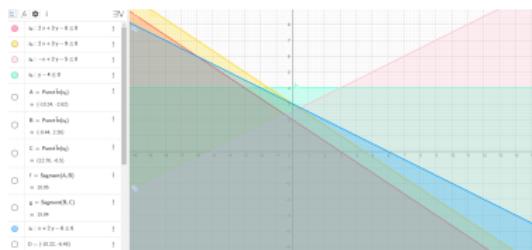
cum se decide dacă este semiplan inferior sau semiplan superior?
Exemple:

$$-x + y + 3 \leq 0 \quad \text{semiplan inferior}$$

$$x - y - 3 \leq 0 \quad \text{semiplan superior}$$

Semiplane inferioare și semiplane superioare

Când determinăm o intersecție de semiplane inferioare / superioare, nu sunt neapărat relevante toate semiplanele. În figura de mai jos sunt considerate cinci semiplane inferioare s_1, s_2, s_3, s_4, s_5 dintre care relevante pentru intersecție sunt doar s_2 și s_4 .

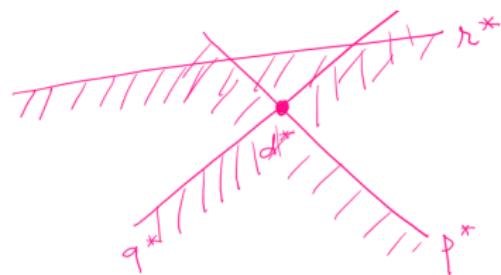


Observația fundamentală

Fie p, q cu $p \neq q$ și dreapta $d = pq$ neverticală. Fie r un punct situat dedesubtul dreptei $d = pq$. Care este configurația duală?



planul primal:
 r este dedesubtul dreptei $pq = d$



planul dual:
 d^* este dedesubtul dreptei r^*

Observația fundamentală

- ▶ Fie \mathcal{P} o mulțime de puncte.

Observația fundamentală

- ▶ Fie \mathcal{P} o mulțime de puncte.
- ▶ **Q:** Ce înseamnă că un segment $[pq]$ ($p, q \in \mathcal{P}$) participă la frontieră superioară a acoperirii convexe a lui \mathcal{P} ?

Observația fundamentală

- ▶ Fie \mathcal{P} o mulțime de puncte.
- ▶ **Q:** Ce înseamnă că un segment $[pq]$ ($p, q \in \mathcal{P}$) participă la frontieră superioară a acoperirii convexe a lui \mathcal{P} ?
- ▶ **A:** Toate celelalte puncte sunt dedesubtul dreptei $d = pq$.

Observația fundamentală

- ▶ Fie \mathcal{P} o mulțime de puncte.
- ▶ **Q:** Ce înseamnă că un segment $[pq]$ ($p, q \in \mathcal{P}$) participă la frontieră superioară a acoperirii convexe a lui \mathcal{P} ?
- ▶ **A:** Toate celelalte puncte sunt dedesubtul dreptei $d = pq$.
- ▶ Configurația duală: Punctul d^* este situat dedesubtul dreptelor corespunzătoare celorlalte puncte și, prin trecere la semiplane inferioare, “conțează” semiplanele inferioare determinate de p^* și q^* .

Concluzie pentru (i) - abordarea cantitativă

- ▶ Pentru a determina o intersecție de **semiplane inferioare** se consideră mulțimea de puncte din planul dual și se determină **frontiera superioară** a acoperirii convexe a mulțimii respective. Un rezultat analog are loc pentru intersecții de **semiplane superioare** și **frontiera inferioară** a acoperirii convexe a mulțimii de puncte duale.
În consecință:

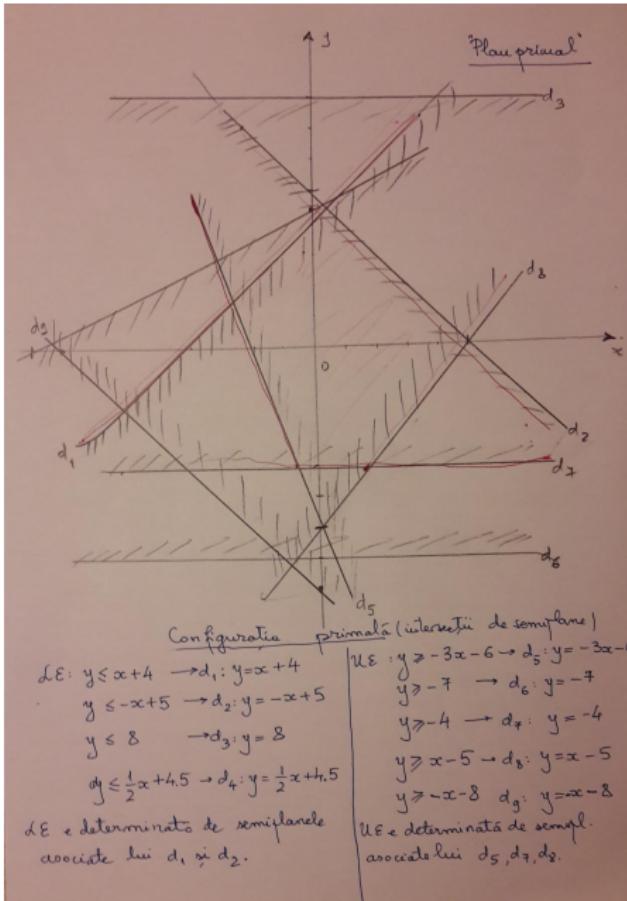
Concluzie pentru (i) - abordarea cantitativă

- ▶ Pentru a determina o intersecție de **semiplane inferioare** se consideră mulțimea de puncte din planul dual și se determină **frontiera superioară** a acoperirii convexe a mulțimii respective. Un rezultat analog are loc pentru intersecții de **semiplane superioare** și **frontiera inferioară** a acoperirii convexe a mulțimii de puncte duale.
În consecință:
- ▶ **Teoremă** *Intersecția a n semiplane poate fi descrisă cu un algoritm de complexitate $O(n \log n)$.*

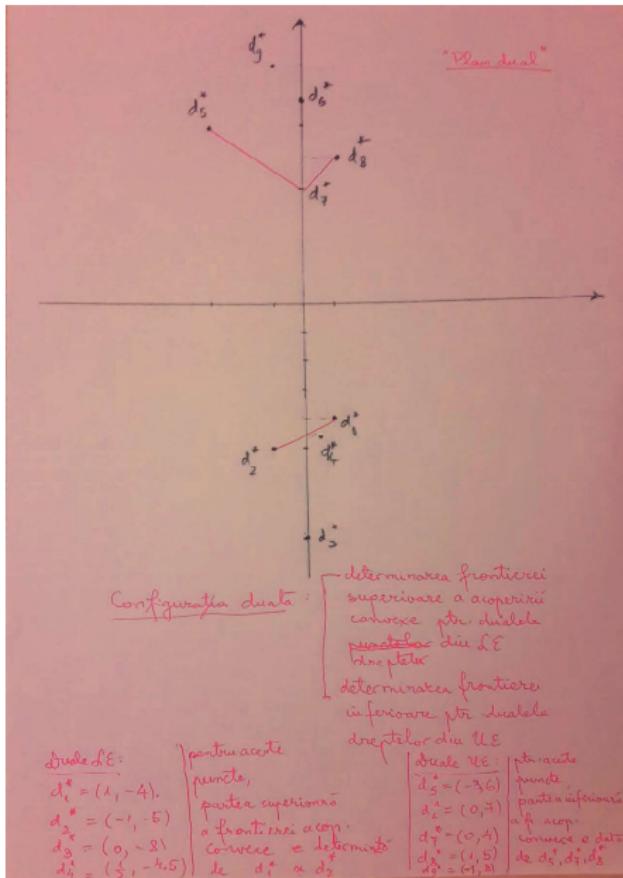
Concluzie pentru (i) - abordarea cantitativă

- ▶ Pentru a determina o intersecție de **semiplane inferioare** se consideră mulțimea de puncte din planul dual și se determină **frontiera superioară** a acoperirii convexe a mulțimii respective. Un rezultat analog are loc pentru intersecții de **semiplane superioare** și **frontiera inferioară** a acoperirii convexe a mulțimii de puncte duale. În consecință:
 - ▶ **Teoremă** *Intersecția a n semiplane poate fi descrisă cu un algoritm de complexitate $O(n \log n)$.*
 - ▶ De fapt, aplicarea dualității combinată cu sortarea lexicografică poate fi interpretată ca fiind ordonare a dreptelor după panta dreptei suport și apoi utilizarea unui mecanism incremental, cu eliminarea semiplanelor redundante. Această abordare directă este descrisă de [Z. Zhu](#) în 2006, în *New Algorithm for Half-plane Intersection and its Practical Value* (v. și [această referință](#)).

Exemplu



Exemplu



(ii) Abordarea calitativă. Motivație

- Sunt realizate 3 produse (note 1, 2 și 3) pe 2 aparate (noteate X și Y).

(ii) Abordarea calitativă. Motivație

- ▶ Sunt realizate 3 produse (noteate 1, 2 și 3) pe 2 aparete (noteate X și Y).
- ▶ Ciclul de producție este săptămânal (40h de lucru). Timpul de producție (în minute) pentru produs este indicat în tabel.

| | X | Y | Obs. | Nr. prod. | Spațiu | Profit |
|---|-----|-----|----------------------|-------------------------|-----------|--------|
| 1 | 10 | 27 | pe ambele | x_1 | $0.1m^2$ | 10 |
| 2 | 12 | 19 | în paralel, simultan | x_2 , respectiv y_2 | $0.2m^2$ | 13 |
| 3 | 8 | 24 | în paralel, simultan | x_3 , respectiv y_3 | $0.05m^2$ | 9 |

(ii) Abordarea calitativă. Motivație

- ▶ Sunt realizate 3 produse (noteate 1, 2 și 3) pe 2 aparate (noteate X și Y).
- ▶ Ciclul de producție este săptămânal (40h de lucru). Timpul de producție (în minute) pentru produs este indicat în tabel.

| | X | Y | Obs. | Nr. prod. | Spațiu | Profit |
|---|----|----|----------------------|-------------------------|-----------|--------|
| 1 | 10 | 27 | pe ambele | x_1 | $0.1m^2$ | 10 |
| 2 | 12 | 19 | în paralel, simultan | x_2 , respectiv y_2 | $0.2m^2$ | 13 |
| 3 | 8 | 24 | în paralel, simultan | x_3 , respectiv y_3 | $0.05m^2$ | 9 |

- ▶ Aparatele X și Y au un interval de menenanță de 5%, respectiv 7% din timpul de lucru. Spațiul total de depozitare este de $50m^2$.

(ii) Abordarea calitativă. Motivație

- Sunt realizate 3 produse (noteate 1, 2 și 3) pe 2 apарате (noteate X și Y).
- Ciclul de producție este săptămânal (40h de lucru). Timpul de producție (în minute) pentru produs este indicat în tabel.

| | X | Y | Obs. | Nr. prod. | Spațiu | Profit |
|---|-----|-----|----------------------|-------------------------|-----------|--------|
| 1 | 10 | 27 | pe ambele | x_1 | $0.1m^2$ | 10 |
| 2 | 12 | 19 | în paralel, simultan | x_2 , respectiv y_2 | $0.2m^2$ | 13 |
| 3 | 8 | 24 | în paralel, simultan | x_3 , respectiv y_3 | $0.05m^2$ | 9 |

- Aparatele X și Y au un interval de menenanță de 5%, respectiv 7% din timpul de lucru. Spațiul total de depozitare este de $50m^2$.
- Modelul matematic:

Constrângeri:

$$\begin{array}{ll} 0.1x_1 + 0.2(x_2 + y_2) + 0.05(x_3 + y_3) \leq 50 & \text{Spațiu de depozitare} \\ 10x_1 + 12x_2 + 8x_3 \leq 0.95 \cdot 40 \cdot 60 & \text{Timp aparatul } X \\ 27x_1 + 19y_2 + 24y_3 \leq 0.93 \cdot 40 \cdot 60 & \text{Timp aparatul } Y \end{array}$$

Cerință:

$$\text{maximizează}(10x_1 + 13(x_2 + y_2) + 9(x_3 + y_3))$$

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

maximizează $(c_1x_1 + c_2x_2 + \dots + c_dx_d)$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

- date de intrare: $(a_{ij})_{i=\overline{1,n}, j=\overline{1,d}}$, $(b_i)_{i=\overline{1,n}}$, $(c_j)_{j=\overline{1,d}}$

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

- date de intrare: $(a_{ij})_{i=\overline{1,n}, j=\overline{1,d}}$, $(b_i)_{i=\overline{1,n}}$, $(c_j)_{j=\overline{1,d}}$
- funcție obiectiv: $(c_1x_1 + c_2x_2 + \dots + c_dx_d)$

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

- date de intrare: $(a_{ij})_{i=\overline{1,n}, j=\overline{1,d}}$, $(b_i)_{i=\overline{1,n}}$, $(c_j)_{j=\overline{1,d}}$
- funcție obiectiv: $(c_1x_1 + c_2x_2 + \dots + c_dx_d)$
- constrângerile: inegalitățile (1)

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

- date de intrare: $(a_{ij})_{i=\overline{1,n}, j=\overline{1,d}}$, $(b_i)_{i=\overline{1,n}}$, $(c_j)_{j=\overline{1,d}}$
- funcție obiectiv: $(c_1x_1 + c_2x_2 + \dots + c_dx_d)$
- constrângerile: inegalitățile (1)
- regiune realizabilă (fezabilă): intersecția semispațiilor care definesc constrângerile problemei

Problematizare, terminologie

- Formulare generală (în spațiul d -dimensional):

$$\text{maximizează } (c_1x_1 + c_2x_2 + \dots + c_dx_d)$$

date constrângerile liniare (inegalități)

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1d}x_d \leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2d}x_d \leq b_2 \\ \dots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nd}x_d \leq b_n \end{array} \right. \quad (1)$$

- Denumiri:

- date de intrare: $(a_{ij})_{i=\overline{1,n}, j=\overline{1,d}}$, $(b_i)_{i=\overline{1,n}}$, $(c_j)_{j=\overline{1,d}}$
- funcție obiectiv: $(c_1x_1 + c_2x_2 + \dots + c_dx_d)$
- constrângerile: inegalitățile (1)
- regiune realizabilă (fezabilă): intersecția semispațiilor care definesc constrângerile problemei

- Obs. Interpretare a cerinței de maximizare: Maximizarea funcției obiectiv revine la a determina un punct al cărui vector de poziție are proiecția maximă de direcția dată de vectorul $\vec{c} = (c_1, c_2, \dots, c_d)$.

Exemplu - cazul 1D ($d = 1$)

Coordonata x :

$$\begin{cases} a_1x \leq b_1, \text{ interval} \\ a_2x \leq b_2 \\ \vdots \\ a_nx \leq b_n \end{cases}$$

funcție obiectiv
constrângeri

maximizează ($c x$)

Exemplu concret : maximizează ($2x$)

$$\begin{cases} 3x \leq 6 \\ -2x \leq 4 \\ 6x \leq 6 \end{cases} \quad \begin{cases} x \leq 2, & x \in (-\infty, 2] \\ x \geq -2, & x \in [-2, \infty) \\ x \leq 1, & x \in (-\infty, 1] \end{cases}$$

pe intervalul $[-2, 1]$. regimne fizabile



Maximul funcției obiectiv este egal cu 2 și se obține printr-o $x = 1$.

Exemplu - cazul 1D ($d = 1$)

Coordonata x :

funcție obiectiv
constraintă

maximizează ($c x$)

$$\left\{ \begin{array}{l} a_1 x \leq b_1, \text{ interval} \\ a_2 x \leq b_2 \\ \vdots \\ a_n x \leq b_n \end{array} \right.$$

Exemplu concret : maximizează ($2x$)

$$\left\{ \begin{array}{l} 3x \leq 6 \\ -2x \leq 4 \\ 6x \leq 6 \end{array} \right. \quad \left\{ \begin{array}{l} x \leq 2, \quad x \in (-\infty, 2] \\ x \geq -2, \quad x \in [-2, \infty) \\ x \leq 1, \quad x \in (-\infty, 1] \end{array} \right.$$

intervalul $[-2, 1]$. regimne fizabile



Maximul funcției obiectiv este egal cu 2 și se obține pr $x = 1$.

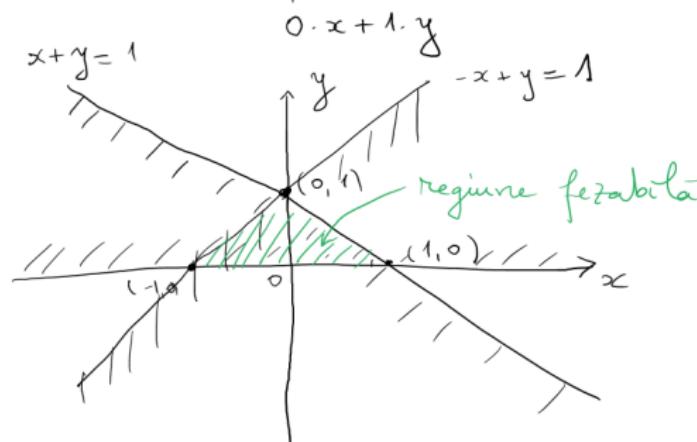
Lemă. (Pentru $d = 1$) Un program liniar 1-dimensional poate fi rezolvat în timp liniar.

Exemplu - cazul 2D ($d = 2$)

Notăm coordonatele cu x și y .

maximizează (y) ; $\vec{c} = (0, 1)$, date

$$\begin{cases} x + y \leq 1 \\ -x + y \leq 0 \\ -x + y \leq 1 \end{cases}$$



Funcția are valoarea maximă 1, atinsă în punctul $(0, 1)$.

Probleme de programare liniară în plan ($d = 2$)

- ▶ **Convenții și terminologie:**

Probleme de programare liniară în plan ($d = 2$)

► Convenții și terminologie:

- Coordonatele: x și y

Probleme de programare liniară în plan ($d = 2$)

► Convenții și terminologie:

- Coordonatele: x și y
- Funcția obiectiv: $f_{\vec{c}}(p) = c_x x + c_y y$, unde $\vec{c} = (c_x, c_y)$.

Probleme de programare liniară în plan ($d = 2$)

► Convenții și terminologie:

- Coordonatele: x și y
- Funcția obiectiv: $f_{\vec{c}}(p) = c_x x + c_y y$, unde $\vec{c} = (c_x, c_y)$.
- Constrângerile: h_1, h_2, \dots, h_n (semiplane); se notează
 $H = \{h_1, h_2, \dots, h_n\}$
- Regiunea fezabilă este $C = h_1 \cap h_2 \cap \dots \cap h_n$.

Probleme de programare liniară în plan ($d = 2$)

► Convenții și terminologie:

- Coordonatele: x și y
 - Funcția obiectiv: $f_{\vec{c}}(p) = c_x x + c_y y$, unde $\vec{c} = (c_x, c_y)$.
 - Constrângерile: h_1, h_2, \dots, h_n (semiplane); se notează
 $H = \{h_1, h_2, \dots, h_n\}$
 - Regiunea fezabilă este $C = h_1 \cap h_2 \cap \dots \cap h_n$.
- **Program liniar:** (H, \vec{c}) .
- **Scop:** Se caută $p \in C$ astfel ca $f_{\vec{c}}(p)$ să fie maximă.

Probleme de programare liniară în plan ($d = 2$)

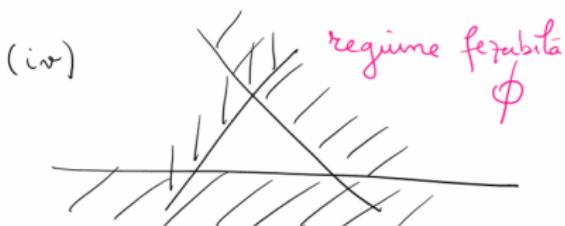
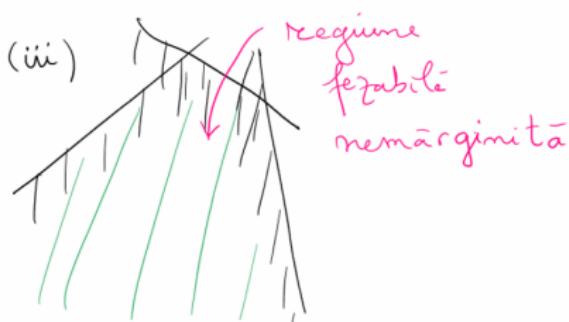
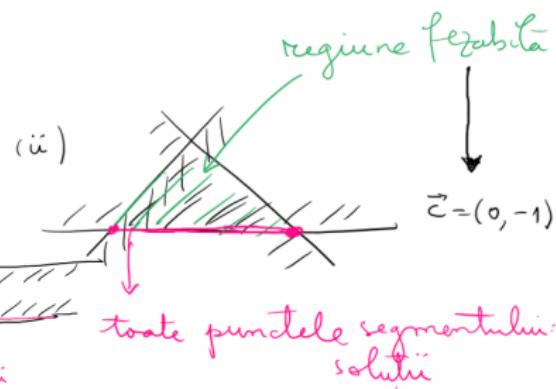
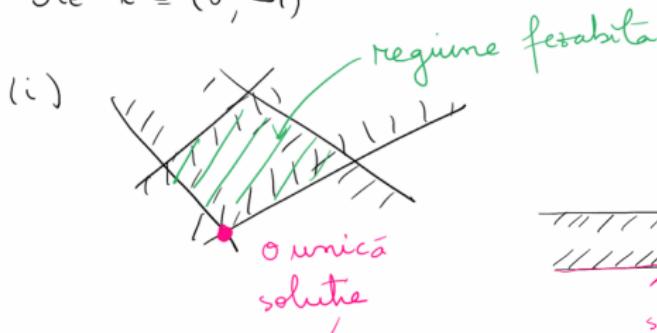
► Convenții și terminologie:

- Coordonatele: x și y
 - Funcția obiectiv: $f_{\vec{c}}(p) = c_x x + c_y y$, unde $\vec{c} = (c_x, c_y)$.
 - Constrângерile: h_1, h_2, \dots, h_n (semiplane); se notează
 $H = \{h_1, h_2, \dots, h_n\}$
 - Regiunea fezabilă este $C = h_1 \cap h_2 \cap \dots \cap h_n$.
- **Program liniar:** (H, \vec{c}) .
- **Scop:** Se caută $p \in C$ astfel ca $f_{\vec{c}}(p)$ să fie maximă.

- Pentru o problemă de programare liniară în plan pot fi distinse patru situații: (i) o soluție unică; (ii) toate punctele de pe o muchie sunt soluții; (iii) regiunea fezabilă este nemărginită și pot fi găsite soluții de-a lungul unei semidrepte; (iv) regiunea fezabilă este vidă.

Cazul 2D ($d = 2$) - exemple de regiuni fezabile

Fie $\vec{e} = (0, -1)$



Algoritm incremental pentru rezolvarea unei probleme de programare liniară 2D

- Principii:

Algoritm incremental pentru rezolvarea unei probleme de programare liniară 2D

- ▶ Principii:
 - ▶ constrângerile sunt adăugate una câte una;

Algoritm incremental pentru rezolvarea unei probleme de programare liniară 2D

► Principii:

- ▶ constrângerile sunt adăugate una câte una;
- ▶ presupunem că la fiecare pas soluția (punctul de maxim) există, apoi actualizează;
- ▶ sunt adăugate la început constrângeri care garantează mărginirea programului liniar, definite astfel: se alege $M \gg 0$ și se definesc noi constrângerile convenabile;

Algoritm incremental pentru rezolvarea unei probleme de programare liniară 2D

► Principii:

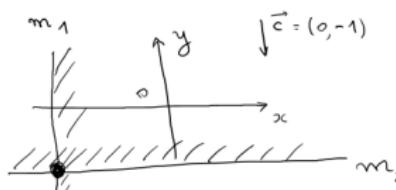
- ▶ constrângerile sunt adăugate una câte una;
- ▶ presupunem că la fiecare pas soluția (punctul de maxim) există, apoi actualizează;
- ▶ sunt adăugate la început constrângeri care garantează mărginirea programului liniar, definite astfel: se alege $M >> 0$ și se definesc noi constrângerile convenabile;
- ▶ se lucrează cu convenția de ordonare lexicografică, astfel încât există o **unică** soluție optimă.

Algoritm incremental pentru rezolvarea unei probleme de programare liniară 2D

► Principii:

- ▶ constrângerile sunt adăugate una câte una;
 - ▶ presupunem că la fiecare pas soluția (punctul de maxim) există, apoi actualizează;
 - ▶ sunt adăugate la început constrângerile care garantează mărginirea programului liniar, definite astfel: se alege $M >> 0$ și se definesc noi constrângerile convenabile;
 - ▶ se lucrează cu convenția de ordonare lexicografică, astfel încât există o **unică** soluție optimă.
- Vom considera în continuare $\vec{c} = (0, -1)$, iar noile constrângerile vor fi:

$$m_1 : x \geq -M, \quad m_2 : y \geq -M.$$



Notății

- Fie (H, \vec{c}) un program liniar cu constrângerile h_1, h_2, \dots, h_n . Se notează:

$H_i = \{m_1, m_2, h_1, h_2, \dots, h_i\}$, **mulțime de semiplane**

$C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \dots \cap h_i$, **regiune fezabilă**.

Notăția este pentru $i = 0, \dots, n$, în particular

$$H_0 = \{m_1, m_2\} \quad C_0 = m_1 \cap m_2.$$

Notării

- Fie (H, \vec{c}) un program liniar cu constrângерile h_1, h_2, \dots, h_n . Se notează:

$H_i = \{m_1, m_2, h_1, h_2, \dots, h_i\}$, mulțime de semiplane

$C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \dots \cap h_i$, regiune fezabilă.

Notăția este pentru $i = 0, \dots, n$, în particular

$$H_0 = \{m_1, m_2\} \quad C_0 = m_1 \cap m_2.$$

- Observații:

Notării

- Fie (H, \vec{c}) un program liniar cu constrângerile h_1, h_2, \dots, h_n . Se notează:

$H_i = \{m_1, m_2, h_1, h_2, \dots, h_i\}$, mulțime de semiplane

$C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \dots \cap h_i$, regiune fezabilă.

Notăția este pentru $i = 0, \dots, n$, în particular

$$H_0 = \{m_1, m_2\} \quad C_0 = m_1 \cap m_2.$$

- Observații:

(i) $C_0 \supseteq C_1 \supseteq C_2 \supseteq \dots \supseteq C_n = C$.

Notății

- Fie (H, \vec{c}) un program liniar cu constrângerile h_1, h_2, \dots, h_n . Se notează:

$H_i = \{m_1, m_2, h_1, h_2, \dots, h_i\}$, multime de semiplane

$C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \dots \cap h_i$, regiune fezabilă.

Notăția este pentru $i = 0, \dots, n$, în particular

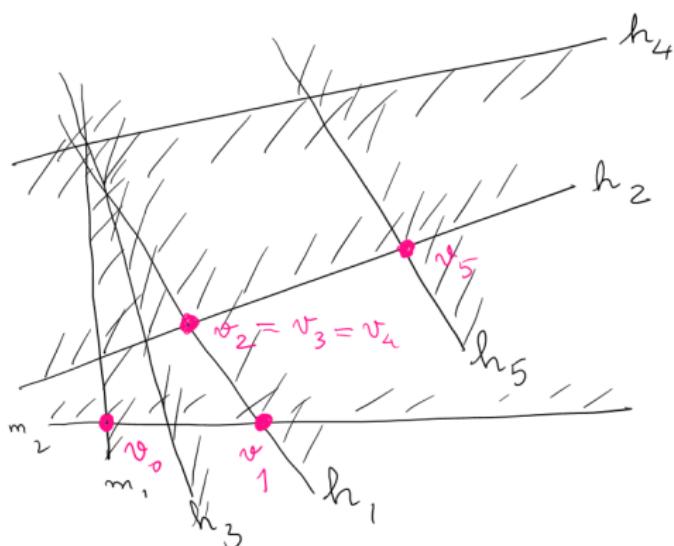
$$H_0 = \{m_1, m_2\} \quad C_0 = m_1 \cap m_2.$$

- Observații:

(i) $C_0 \supseteq C_1 \supseteq C_2 \supseteq \dots \supseteq C_n = C$.

(ii) Pentru fiecare i , regiunea fezabilă C_i , dacă este nevidă, are un vîrf care reprezintă o soluție optimă a problemei (H_i, \vec{c}) . Punctul este notat cu v_i (depinde de alegerea lui m_1 și m_2).

Exemplu



$$\downarrow$$

$$v_3 = v_2, \text{ pt. c\~a } v_2 \in h_3$$

$$v_4 = v_3 = v_2 \text{ pt. c\~a } v_3 \in h_4$$

$$v_5 \neq v_4 \text{ pt. c\~a } v_4 \notin h_5$$

Observații

- ▶ Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:

Observații

- Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:
 - (i) dacă $v_{i-1} \in h_i$, atunci $v_i = v_{i-1}$,

Observații

- Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:
 - (i) dacă $v_{i-1} \in h_i$, atunci $v_i = v_{i-1}$,
 - (ii) dacă $v_{i-1} \notin h_i$, atunci

Observații

- Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:

- (i) dacă $v_{i-1} \in h_i$, atunci $v_i = v_{i-1}$,
- (ii) dacă $v_{i-1} \notin h_i$, atunci
fie $C_i = \emptyset$

Observații

- Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:
 - (i) dacă $v_{i-1} \in h_i$, atunci $v_i = v_{i-1}$,
 - (ii) dacă $v_{i-1} \notin h_i$, atunci
 - fie $C_i = \emptyset$
 - fie $v_i \in d_i$, unde d_i este dreapta care mărginește h_i .

Observații

- Fie $1 \leq i \leq n$, presupunem că C_{i-1} și v_{i-1} sunt determinate. Considerăm h_i . Sunt două situații:

- (i) dacă $v_{i-1} \in h_i$, atunci $v_i = v_{i-1}$,
- (ii) dacă $v_{i-1} \notin h_i$, atunci

fie $C_i = \emptyset$

fie $v_i \in d_i$, unde d_i este dreapta care mărginește h_i .

În acest caz, găsirea lui v_i revine la găsirea lui $p \in d_i$ care maximizează $f_c(p)$, date constrângerile deja existente ($p \in h, \forall h \in H_i$). **De fapt, aceasta este o problemă pe programare liniară 1-dimensională, care are complexitatea-timp liniară, adică $O(i)$.**

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
 - ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.
1. $v_0 \leftarrow$ "colțul" lui c_0

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$ punctul p de pe d_i care
 maximizează $f_{\vec{c}}(p)$ date constrângerile din H_i

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$ punctul p de pe d_i care
 maximizează $f_{\vec{c}}(p)$ date constrângerile din H_i
7. **if** p nu există

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

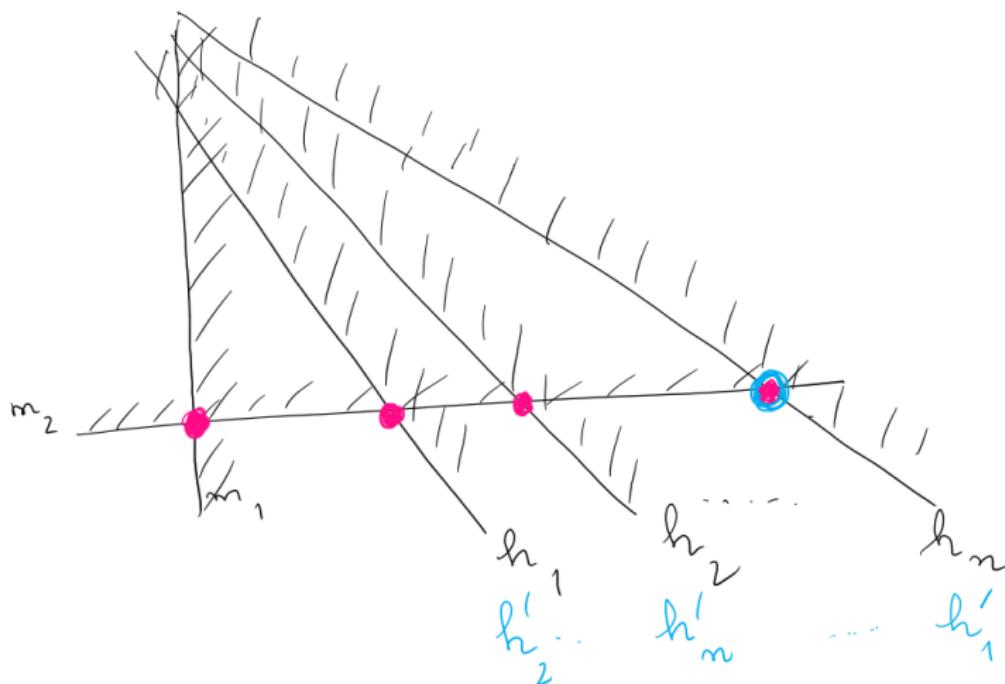
1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$ punctul p de pe d_i care
 maximizează $f_{\vec{c}}(p)$ date constrângerile din H_i
7. **if** p nu există
8. **then** raportează "nefezabil" **end**

Algoritm LPMARG2D (H, \vec{c}, m_1, m_2)

- ▶ **Input.** Un program liniar $(H \cup \{m_1, m_2\}, \vec{c})$ din \mathbb{R}^2
- ▶ **Output.** Dacă $(H \cup \{m_1, m_2\}, \vec{c})$ nu e realizabil (fezabil), raportează. În caz contrar, indică punctul cel mai mic lexicografic p care maximizează $f_{\vec{c}}(p)$.

1. $v_0 \leftarrow$ "colțul" lui c_0
2. fie h_1, h_2, \dots, h_n semiplanele din H
3. **for** $i \leftarrow 1$ **to** n
4. **do if** $v_{i-1} \in h_i$
5. **then** $v_i \leftarrow v_{i-1}$
6. **else** $v_i \leftarrow$ punctul p de pe d_i care
 maximizează $f_{\vec{c}}(p)$ date constrângerile din H_i
7. **if** p nu există
8. **then** raportează "nefezabil" **end**
9. **return** v_n

Comentariu - ordinea contează



Algoritm aleatoriu

- ▶ **Pasul 2.** este înlocuit cu:
2'. Calculează o permutare arbitrară a semiplanelor, folosind o procedură adecvată.

Algoritm aleatoriu

- ▶ Pasul 2. este înlocuit cu:
2'. Calculează o permutare arbitrară a semiplanelor, folosind o procedură adecvată.
- ▶ Algoritmul incremental LPMARG2D are complexitate-timp $O(n^2)$, iar varianta bazată pe alegerea aleatorie a semiplanelor are complexitate-timp medie $O(n)$ (n este numărul semiplanelor).

Analiza complexității-timp - varianta algoritmului probabilist (I)

Fie $(X_i)_{i=1,\dots,n}$ variabilă aleatoare definită astfel:

$$X_i = \begin{cases} 0, & \text{dacă } v_{i-1} \in h_i \text{ (adică este ales pasul 5 la iterarea } i\text{)} \\ 1, & \text{dacă } v_{i-1} \notin h_i \text{ (adică este ales pasul 6 la iterarea } i\text{).} \end{cases}$$

Analiza complexității-timp - varianta algoritmului probabilist (I)

Fie $(X_i)_{i=1,\dots,n}$ variabilă aleatoare definită astfel:

$$X_i = \begin{cases} 0, & \text{dacă } v_{i-1} \in h_i \text{ (adică este ales pasul 5 la iterarea } i\text{)} \\ 1, & \text{dacă } v_{i-1} \notin h_i \text{ (adică este ales pasul 6 la iterarea } i\text{).} \end{cases}$$

În concluzie, timpul total de rulare este $\sum_{i=1}^n X_i O(i) + O(n)$.

Analiza complexității-timp - varianta algoritmului probabilist (I)

Fie $(X_i)_{i=1,\dots,n}$ variabilă aleatoare definită astfel:

$$X_i = \begin{cases} 0, & \text{dacă } v_{i-1} \in h_i \text{ (adică este ales pasul 5 la iteratăia } i) \\ 1, & \text{dacă } v_{i-1} \notin h_i \text{ (adică este ales pasul 6 la iteratăia } i). \end{cases}$$

În concluzie, timpul total de rulare este $\sum_{i=1}^n X_i O(i) + O(n)$.

Valoarea așteptată (timpul mediu)

$$E\left[\sum_{i=1}^n X_i O(i)\right] = \mu\left(\sum_{i=1}^n X_i O(i)\right) = \sum_{i=1}^n O(i) \mu(X_i) \leq \sum_{i=1}^n O(i) \cdot \frac{2}{i} = O(n).$$

Analiza complexității-timp - varianta algoritmului probabilist (I)

Fie $(X_i)_{i=1,\dots,n}$ variabilă aleatoare definită astfel:

$$X_i = \begin{cases} 0, & \text{dacă } v_{i-1} \in h_i \text{ (adică este ales pasul 5 la iteratăia } i) \\ 1, & \text{dacă } v_{i-1} \notin h_i \text{ (adică este ales pasul 6 la iteratăia } i). \end{cases}$$

În concluzie, timpul total de rulare este $\sum_{i=1}^n X_i O(i) + O(n)$.

Valoarea așteptată (timpul mediu)

$$E\left[\sum_{i=1}^n X_i O(i)\right] = \mu\left(\sum_{i=1}^n X_i O(i)\right) = \sum_{i=1}^n O(i) \mu(X_i) \leq \sum_{i=1}^n O(i) \cdot \frac{2}{i} = O(n).$$

Vom arăta că $\mu(X_i) \leq \frac{2}{i}$, de unde va rezulta inegalitatea dorită.

Analiza complexității-timp - varianta algoritmului probabilist (II)

- Demonstrăm că $\mu(X_i) \leq \frac{2}{i}$, pentru orice $i = 1, \dots, n$, adică probabilitatea ca $v_{i-1} \notin h_i$ este $\leq \frac{2}{i}$.

Analiza complexității-timp - varianta algoritmului probabilist (II)

- ▶ Demonstrăm că $\mu(X_i) \leq \frac{2}{i}$, pentru orice $i = 1, \dots, n$, adică probabilitatea ca $v_{i-1} \notin h_i$ este $\leq \frac{2}{i}$.
- ▶ Arătăm inegalitatea pentru $i = n$ (cazul general, analog). Presupunem algoritmul terminat, v_n vârful optim.

Analiza complexității-timp - varianta algoritmului probabilist (II)

- ▶ Demonstrăm că $\mu(X_i) \leq \frac{2}{i}$, pentru orice $i = 1, \dots, n$, adică probabilitatea ca $v_{i-1} \notin h_i$ este $\leq \frac{2}{i}$.
- ▶ Arătăm inegalitatea pentru $i = n$ (cazul general, analog). Presupunem algoritmul terminat, v_n vârful optim.
 - Care este probabilitatea ca $v_{n-1} \notin h_n$, adică la adăugarea lui h_n , vârful v_{n-1} să fie modificat în v_n ? \Leftrightarrow

Analiza complexității-timp - varianta algoritmului probabilist (II)

- ▶ Demonstrăm că $\mu(X_i) \leq \frac{2}{i}$, pentru orice $i = 1, \dots, n$, adică probabilitatea ca $v_{i-1} \notin h_i$ este $\leq \frac{2}{i}$.
- ▶ Arătăm inegalitatea pentru $i = n$ (cazul general, analog). Presupunem algoritmul terminat, v_n vârful optim.
 - Care este probabilitatea ca $v_{n-1} \notin h_n$, adică la adăugarea lui h_n , vârful v_{n-1} să fie modificat în v_n ? \Leftrightarrow
 - Care este probabilitatea ca eliminând unul dintre semiplane să fie modificat vârful optim v_n ?

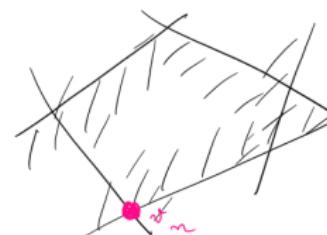
Analiza complexității-timp - varianta algoritmului probabilist (II)

- ▶ Demonstrăm că $\mu(X_i) \leq \frac{2}{i}$, pentru orice $i = 1, \dots, n$, adică probabilitatea ca $v_{i-1} \notin h_i$ este $\leq \frac{2}{i}$.
 - ▶ Arătăm inegalitatea pentru $i = n$ (cazul general, analog). Presupunem algoritmul terminat, v_n vârful optim.
 - Care este probabilitatea ca $v_{n-1} \notin h_n$, adică la adăugarea lui h_n , vârful v_{n-1} să fie modificat în v_n ? \Leftrightarrow
 - Care este probabilitatea ca eliminând unul dintre semiplane să fie modificat vârful optim v_n ?

- mr. cauzwi possible: m
- mr. cauzwi surviving
modification = 2

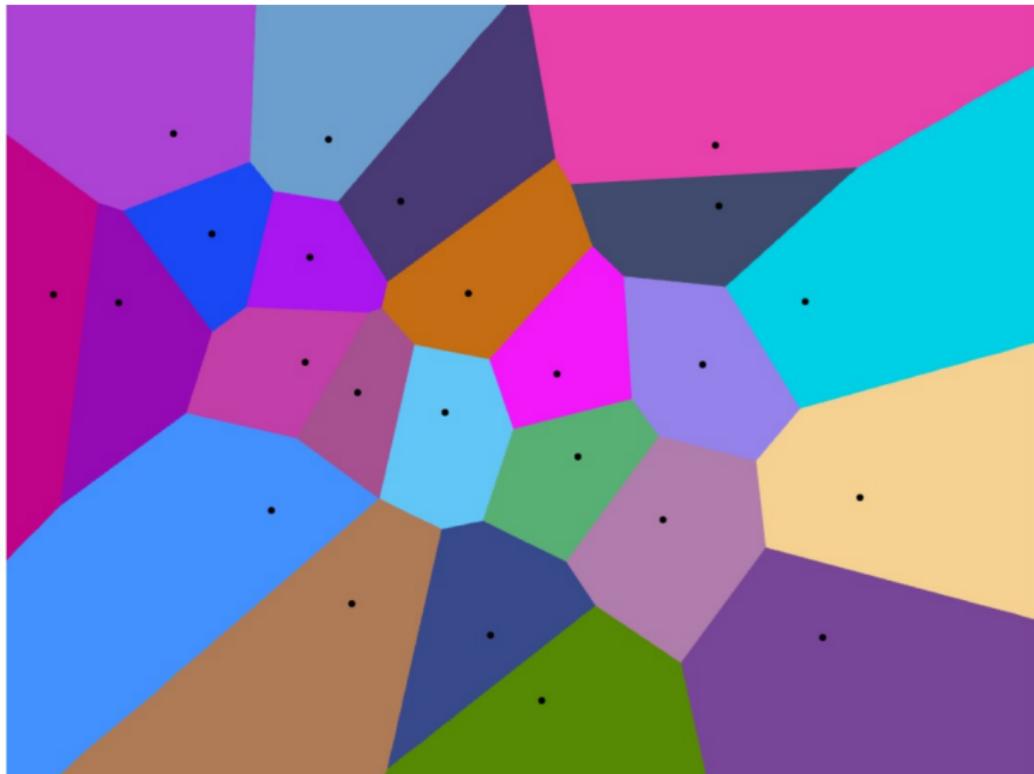
\Rightarrow probabilitatea de a modifica $\leq \frac{2}{n}$

analog ptr. i - q.e.d.

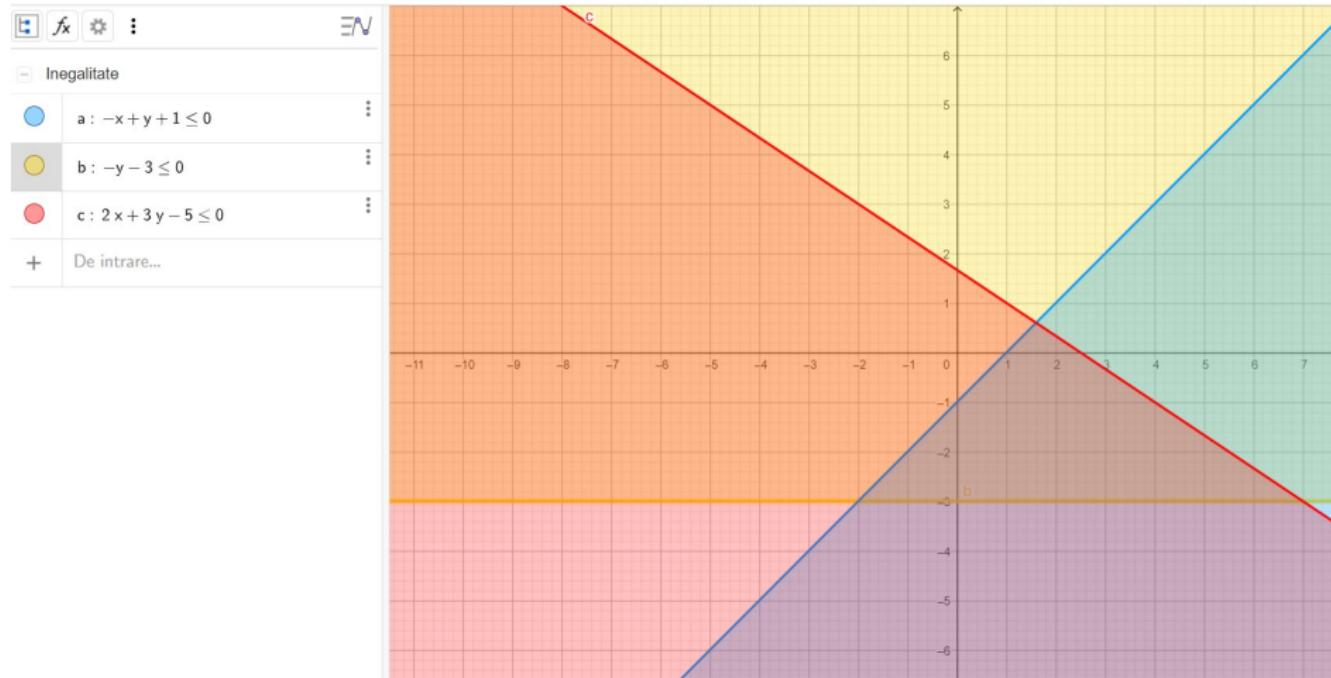


$$\downarrow \vec{c} = (0, -1)$$

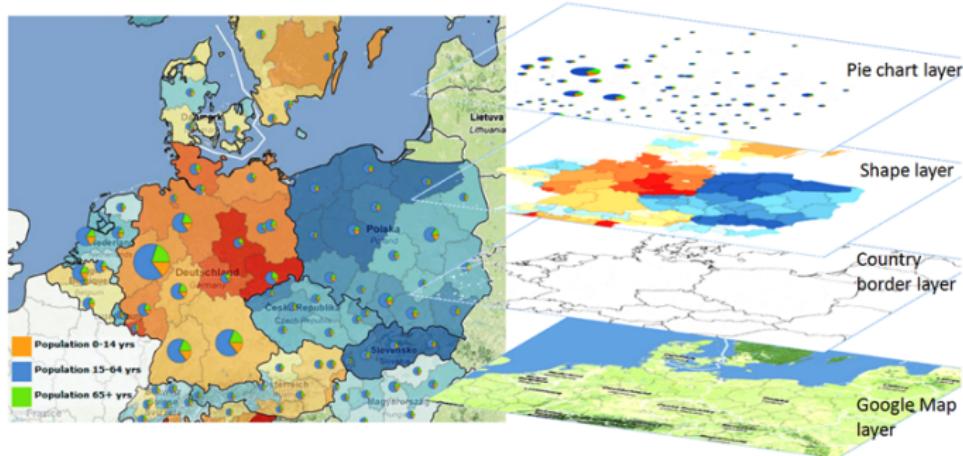
Motivație - reprezentarea diagramelor Voronoi



Motivație - reprezentarea intersecțiilor de semiplane



Motivație - reprezentarea datelor geo-spațiale



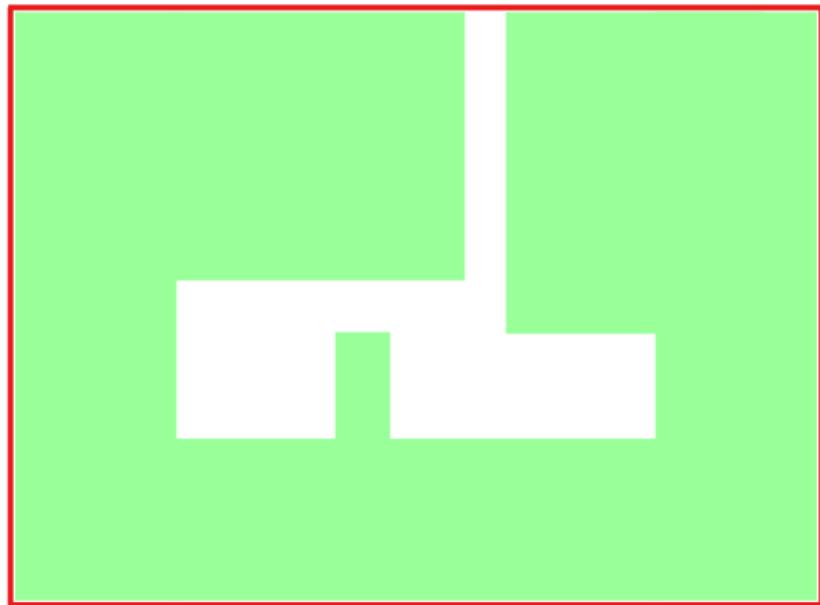
Sursa: <https://s-media-cache-ak0.pinimg.com/originals/37/90/86/37908600ab7db99c424c3bc6e1ddb740.jpg>

Ce structură de date este adecvată pentru a memora astfel de informații?

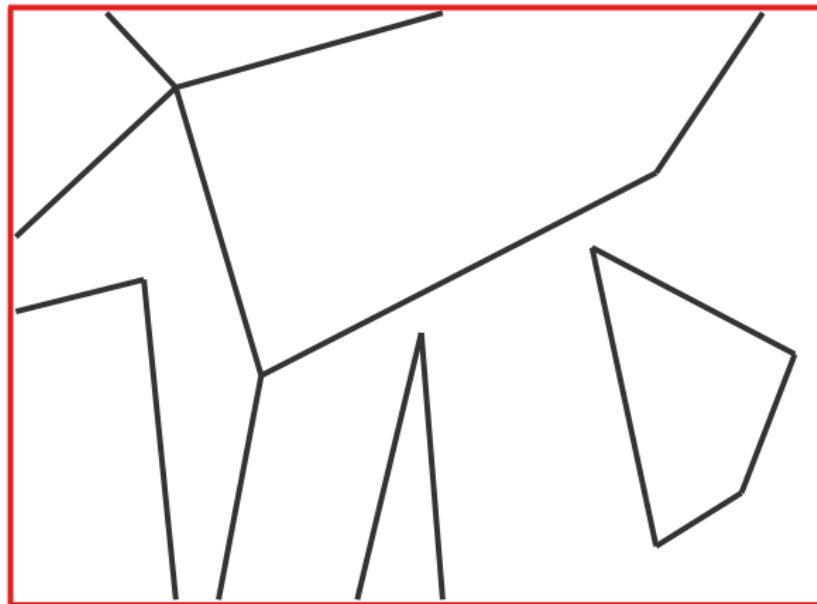
Problematizare



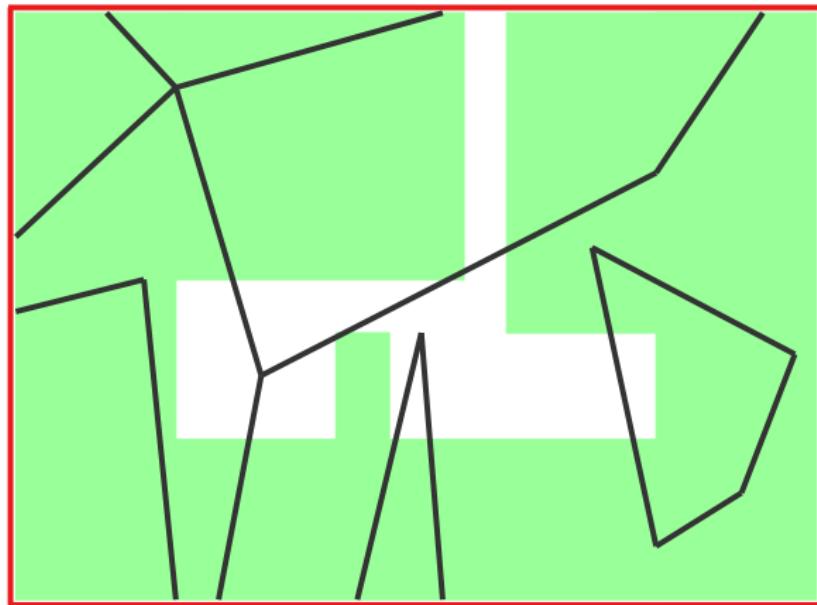
Problematizare



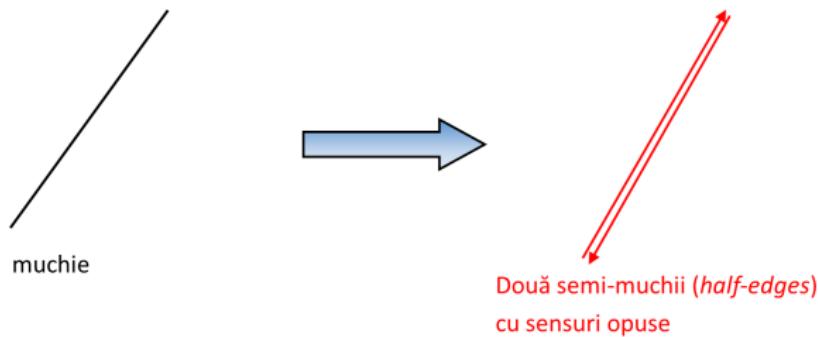
Problematizare



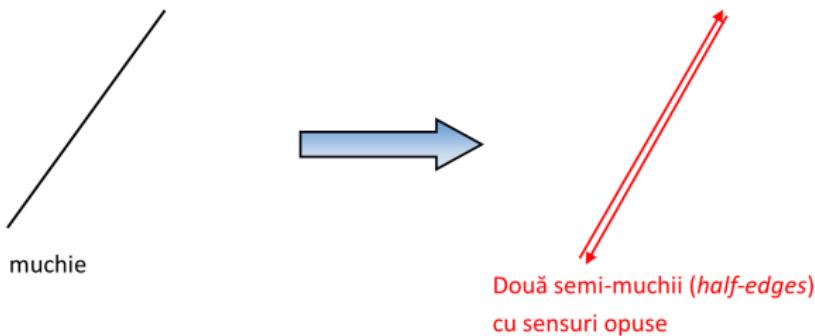
Problematizare



Ideea cheie

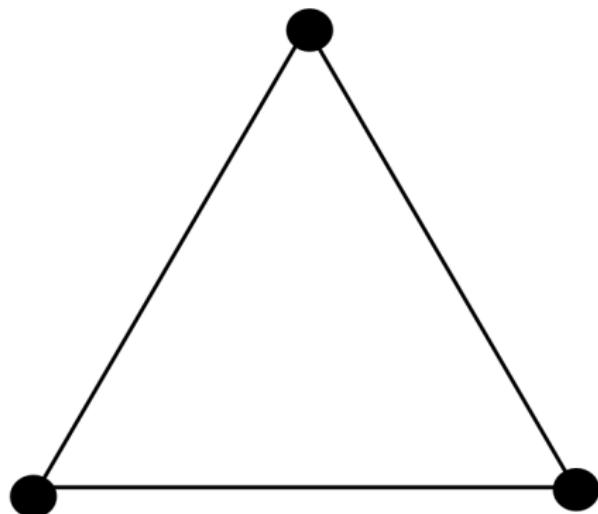


Ideea cheie

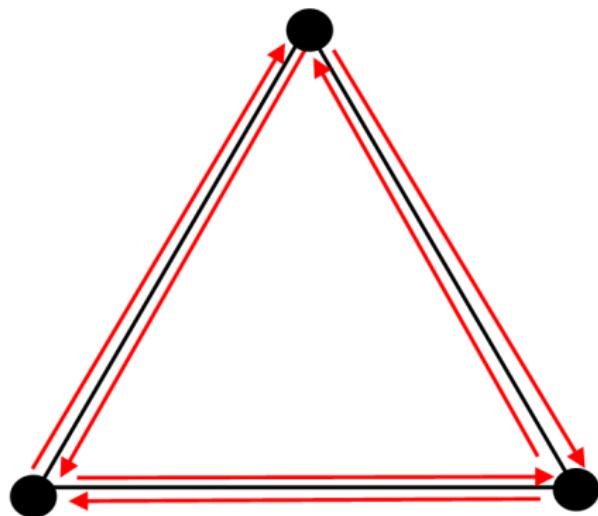


- ▶ Ideea fundamentală este de a introduce conceptul de semi-muchie (muchie orientată), cf. “*half-edge*”.

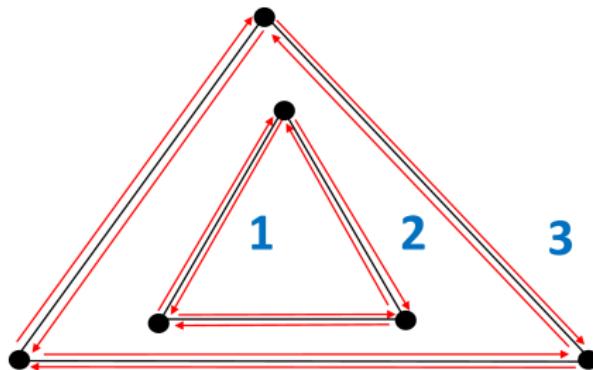
Exemplu



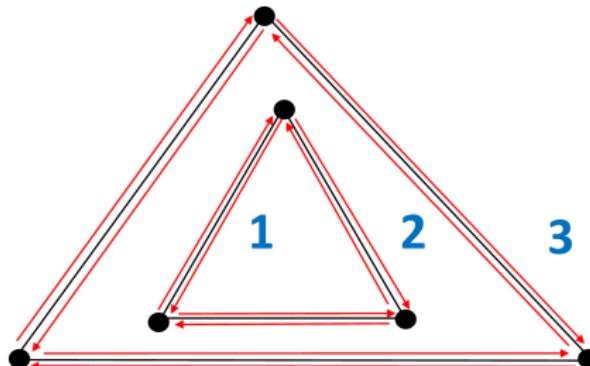
Exemplu



Exemplu

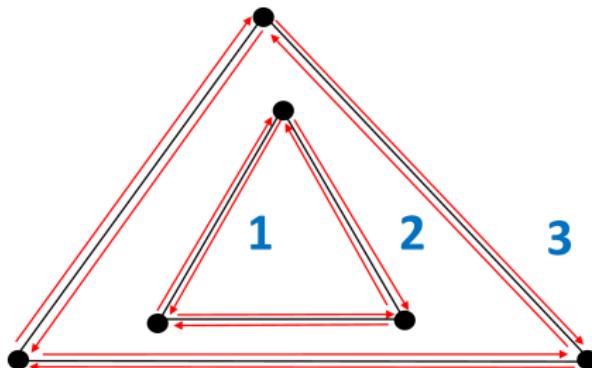


Exemplu



Subdiviziunea din figură are 6 vârfuri, 12 semi-muchii, 3 fețe.

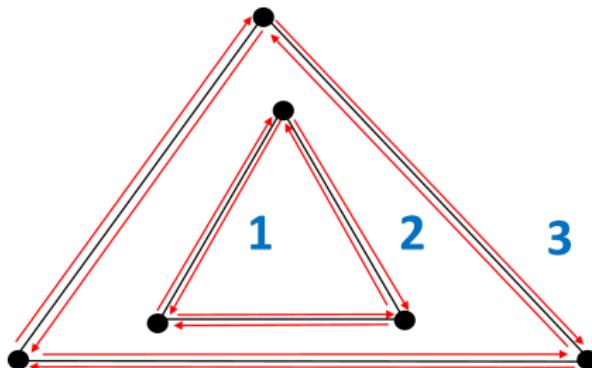
Exemplu



Subdiviziunea din figură are 6 vârfuri, 12 semi-muchii, 3 fețe.

Semi-muchiile delimită frontiere ale fețelor, față delimitată fiind la stânga.

Exemplu

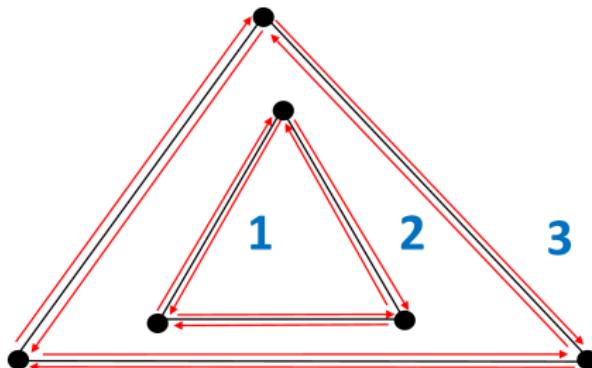


Subdiviziunea din figură are 6 vârfuri, 12 semi-muchii, 3 fețe.

Semi-muchiile delimită frontiere ale fețelor, față delimitată fiind la stânga.

Dat un poligon (eventual cu goluri):

Exemplu



Subdiviziunea din figură are 6 vârfuri, 12 semi-muchii, 3 fețe.

Semi-muchiile delimită frontiere ale fețelor, față delimitată fiind la stânga.

Dat un poligon (eventual cu goluri):

- ▶ **frontieră exterioară**, care poate fi parcursă cu ajutorul semi-muchiilor astfel încât poligonul să fie la stânga frontierei, iar virajele convexe să fie la stânga,
- ▶ **frontieră interioară** (dacă există goluri), caz în care poligonul este tot la stânga, dar virajele în vîrfurile convexe sunt la dreapta.

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăntuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăncuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
 - ▶ **Vârf** v : coordonatele lui v în $\text{Coordinates}(v)$, pointer $\text{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăntuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
 - ▶ **Vârf** v : coordonatele lui v în $\text{Coordinates}(v)$, pointer $\text{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine
 - ▶ **Față** f : pointer $\text{OuterComponent}(f)$ spre o muchie orientată corespunzătoare frontierei externe (pentru fața nemărginită este **nil**); listă $\text{InnerComponents}(f)$, care conține, pentru fiecare gol, un pointer către una dintre muchiile orientate de pe frontieră

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăntuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
 - ▶ **Vârf** v : coordonatele lui v în $\text{Coordinates}(v)$, pointer $\text{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine
 - ▶ **Față** f : pointer $\text{OuterComponent}(f)$ spre o muchie orientată corespunzătoare frontierei externe (pentru fața nemărginită este **nil**); listă $\text{InnerComponents}(f)$, care conține, pentru fiecare gol, un pointer către una dintre muchiile orientate de pe frontieră
 - ▶ **Muchie orientată** \vec{e} : pointer $\text{Origin}(\vec{e})$, pointer $\text{Twin}(\vec{e})$ pointer $\text{IncidentFace}(\vec{e})$, pointer $\text{Next}(\vec{e})$, pointer $\text{Prev}(\vec{e})$.

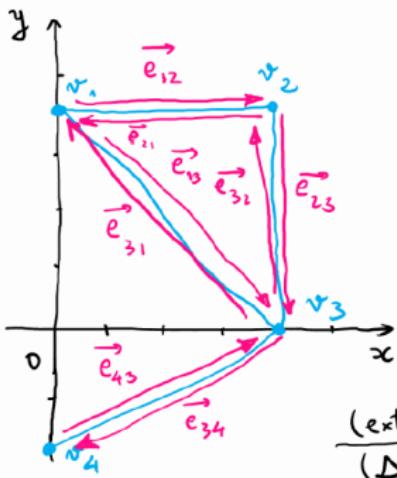
Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăntuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
 - ▶ **Vârf** v : coordonatele lui v în $\text{Coordinates}(v)$, pointer $\text{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine
 - ▶ **Față** f : pointer $\text{OuterComponent}(f)$ spre o muchie orientată corespunzătoare frontierei externe (pentru fața nemărginită este **nil**); listă $\text{InnerComponents}(f)$, care conține, pentru fiecare gol, un pointer către una dintre muchiile orientate de pe frontieră
 - ▶ **Muchie orientată** \vec{e} : pointer $\text{Origin}(\vec{e})$, pointer $\text{Twin}(\vec{e})$ pointer $\text{IncidentFace}(\vec{e})$, pointer $\text{Next}(\vec{e})$, pointer $\text{Prev}(\vec{e})$.
- ▶ Oricărei subdiviziuni planare \mathcal{S} i se asociază o listă de muchii dublu înlăntuite $\mathcal{D}_{\mathcal{S}}$.

Subdiviziuni planare

- ▶ Conceptul de subdiviziune planară: vârfuri, muchii, fețe.
- ▶ **Listă de muchii dublu înlăntuite / DCEL - Doubly Connected Edge List** [Müller și Preparata, 1978] (trei înregistrări: vârfuri, fețe, muchii orientate (semi-muchii)).
 - ▶ **Vârf** v : coordonatele lui v în $\text{Coordinates}(v)$, pointer $\text{IncidentEdge}(v)$ spre o muchie orientată care are v ca origine
 - ▶ **Față** f : pointer $\text{OuterComponent}(f)$ spre o muchie orientată corespunzătoare frontierei externe (pentru fața nemărginită este **nil**); listă $\text{InnerComponents}(f)$, care conține, pentru fiecare gol, un pointer către una dintre muchiile orientate de pe frontieră
 - ▶ **Muchie orientată** \vec{e} : pointer $\text{Origin}(\vec{e})$, pointer $\text{Twin}(\vec{e})$ pointer $\text{IncidentFace}(\vec{e})$, pointer $\text{Next}(\vec{e})$, pointer $\text{Prev}(\vec{e})$.
- ▶ Oricărei subdiviziuni planare \mathcal{S} i se asociază o listă de muchii dublu înlăntuite $\mathcal{D}_{\mathcal{S}}$.
- ▶ **Obs.** Explicați cum, folosind pointerii de mai sus: (i) poate fi parcursă frontiera exterioară / interioară a unei fețe (a unui poligon); (ii) pot fi găsite toate semi-muchiile incidente cu un vârf.

Exemplu



| Vârf | Coordinate | Incident Edge |
|-------|------------|----------------------------------|
| v_1 | (0, 4) | \vec{e}_{12} |
| v_2 | (4, 4) | \vec{e}_{23} |
| v_3 | (4, 0) | \vec{e}_{32} |
| v_4 | (0, -2) | \vec{e}_{24} \vec{e}_{43} |

| Fata | Outer Component | Inner Component(s) |
|--------------------|-----------------|--------------------|
| (exterior) f_1 | nil | \vec{e}_{12} |
| (Δ) f_2 | \vec{e}_{23} | nil |

| Semi-monchile | Origin | Twist | Incident Face | Next | Prev |
|----------------|--------|----------------|---------------|----------------|----------------|
| \vec{e}_{12} | v_1 | \vec{e}_{21} | f_1 | \vec{e}_{23} | \vec{e}_{31} |
| \vec{e}_{23} | v_2 | \vec{e}_{32} | f_1 | \vec{e}_{34} | \vec{e}_{12} |
| \vec{e}_{34} | v_3 | \vec{e}_{43} | f_2 | \vec{e}_{43} | \vec{e}_{23} |

Algoritmi Avansați 2023

c-7

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro





Index

- Desfășurare examen & predare
- Ce este un algoritm
- Complexitatea timp a unui algoritm
- P, NP, NPC
- Ce este o problema de optim?
- Idei alternative de rezolvare (prelude)

Desfășurare examen & predare

- Curs Modular;
- Evaluare pe parcurs 50% + examen final 50%
- Limbaj de programare: La alegere Python sau C++
- [7 săptamani] va fi o continuare a cursului de AF
- Prezenta nu este obligatorie dar probabil este necesara
- Promovarea unui mediu interactiv
- Feedback-ul este mereu de apreciat





Ce este un algoritm?

- informal: o succesiuni de pasi elementari/simpli dupa a căror execuție pe un input dat, obținem un output care este soluție pentru problema noastră
- Formal: Echivalent cu **Mașina Turing** (*to be continued*)



Complexitatea timp

Informal spus, complexitatea timp a unui algoritm este dat de *numărul de operații* efectuate până ce se ajunge la rezultat. Evident numărul de operații va depinde și de input, mai exact lungimea inputului.

Fie un algoritm care pentru o intrare (de lungime) n efectuează $f(n)$ operații.

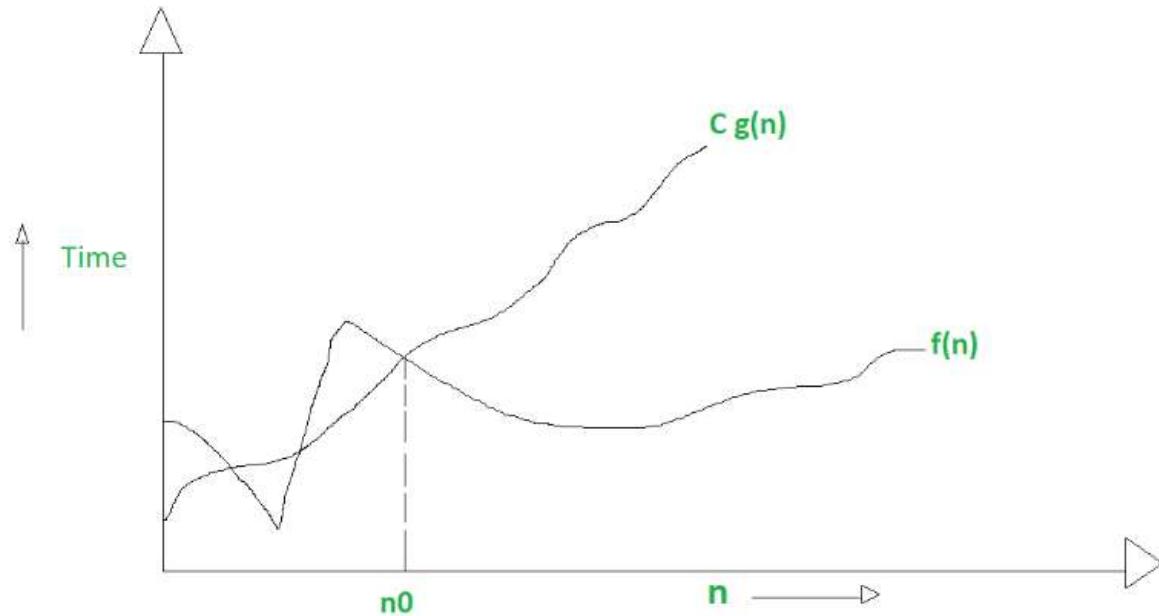
Definim clasele de complexitate \mathcal{O} , Ω , Θ după cum urmează

Clasa O (Big Oh)



- Descrie o **limită superioară** pentru numărul de operații efectuate de algoritm pentru orice intrare de la o lungime n_0 încolo.
- Spunem ca un algoritm rulează în timp $O(g(n))$ dacă există o funcție g și o valoare n_0 , astfel încât să avem relația: $C \times g(n) \geq f(n) \geq 0 \mid \forall n > n_0$ (unde C este o constantă pozitivă).
- f este asimptotic mărginită superior de către g (multiplicată cu un factor constant C)
- Observăm, spre exemplu, că $O(n)$ este inclusă în $O(n^2)$

Clasa O (Big Oh)



Clasa O (Big Oh)



Definiția riguroasă este "un pic mai complicată":

Fie un algoritm Alg și o funcție $f:N \rightarrow N$, astfel încât Alg se termină exact în $f(n)$ pași pentru o intrare de lungime "n". Spunem că Alg rulează în $O(g(n))$ dacă avem relația:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty$$

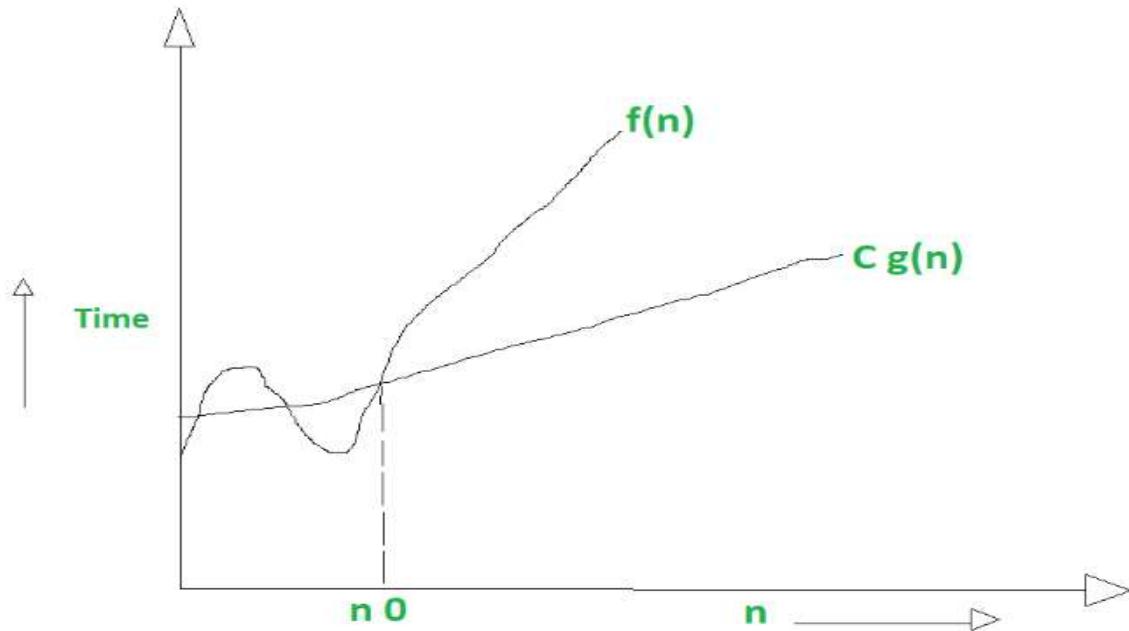
Această definiție ne arată ca în clasa de complexitate O, factorul dominant este cel care ne dă complexitatea. Ex: $O(n^2+2n) \equiv O(n^2)$

Clasa Ω (Big Omega)



- Descrie o limită inferioară pentru numărul de operații efectuate de algoritm pentru orice intrare de la o lungime n_0 încolo.
- Spunem ca un algoritm rulează în timp $\Omega(g(n))$ dacă există o funcție g și o valoare n_0 , astfel încât să avem relația: $f(n) \geq C \times g(n) \geq 0 \mid \forall n > n_0$ (unde C este o constantă pozitivă).
- f este asimptotic mărginită inferior de către g (multiplicată cu un factor constant C)
- Observăm, spre exemplu, că $\Omega(n)$ este inclusă în $\Omega(n^2)$

Clasa Ω (Big Omega)



Clasa Ω (Big Omega)



Definiția riguroasă:

Fie un algoritm Alg și o funcție $f:N \rightarrow N$, astfel încât Alg se termină exact în $f(n)$ pași pentru o intrare de lungime "n". Spunem că Alg rulează în $\Omega(g(n))$ dacă avem relația:

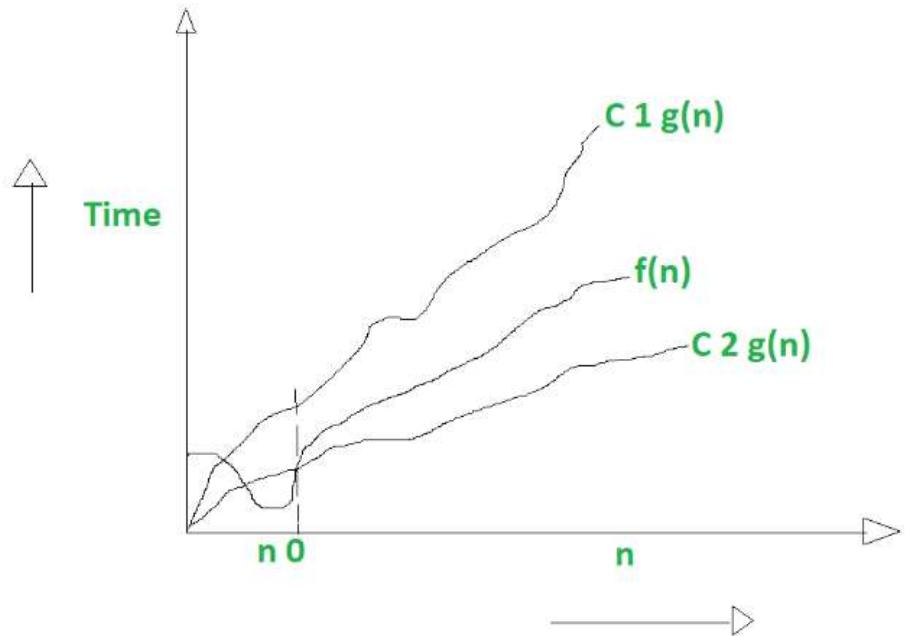
$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0$$

Clasa Θ (Big Theta)



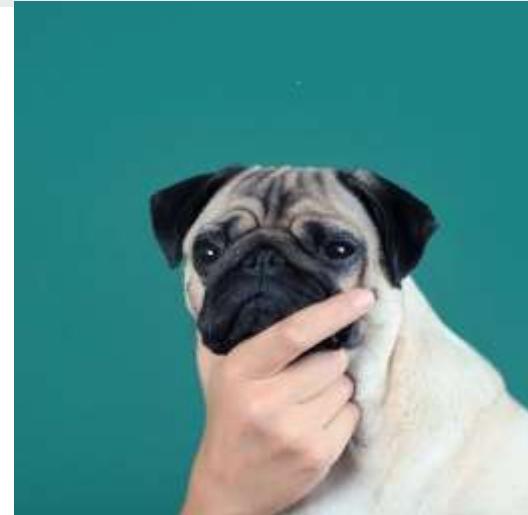
- O combinație între cele două clase anterioare. Presupune o mărginire atât superioară cât și inferioară.
- Spunem ca un algoritm rulează în timp $\Theta(g(n))$ dacă există o funcție g și o valoare n_0 , astfel încât să avem relația: $C_1 \times g(n) \geq f(n) \geq C_2 \times g(n) \geq 0 \mid \forall n > n_0$ (unde C_1 și C_2 sunt două constante pozitive).
- f este asymptotic mărginită inferior de către g (multiplicată cu un factor constant C_2) respectiv superior tot de g (multiplicată cu un factor constant C_1)
- **Nu mai este valabilă observația că $\Theta(n)$ ar fi inclusă în $\Theta(n^2)$**
- Nu toți algoritmii au o complexitate Theta

Clasa Θ (Big Theta)



Clasa Θ (Big Theta)

Cum ar arăta definiția folosind limite?



Clasa Θ (Big Theta)



Fie un algoritm Alg și o funcție $f:N \rightarrow N$, astfel încât Alg se termină exact în $f(n)$ pași pentru o intrare de lungime "n". Spunem că Alg rulează în $\Theta(g(n))$ dacă avem relația:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} \in \mathbb{R}_+$$

Discuții libere

Cel mai adesea folosim clasa O.

Evident ne interesează numitele "tight bounds".

Ce se întâmplă când pe o intrare de aceeași lungime ai număr de pași semnificativ diferiți?

- Complexitate "worst case" vs complexitate medie

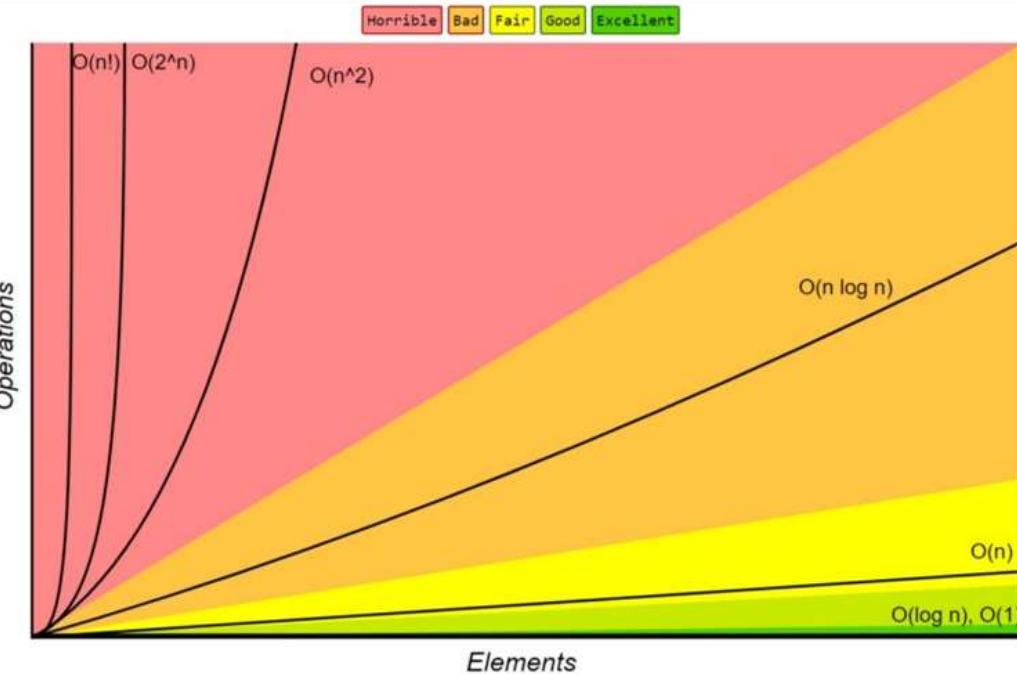
Care este complexitatea următorilor algoritmi?

- Căutare binară; Bubble sort; quicksort, merge-sort;

Paradoxul în care Big-O nu redă realitatea...

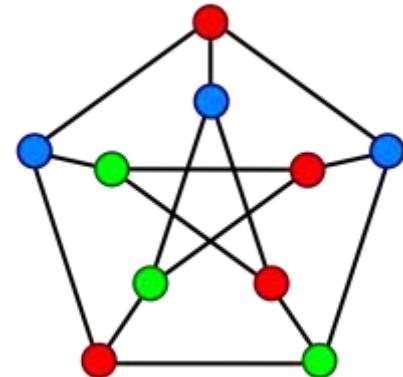
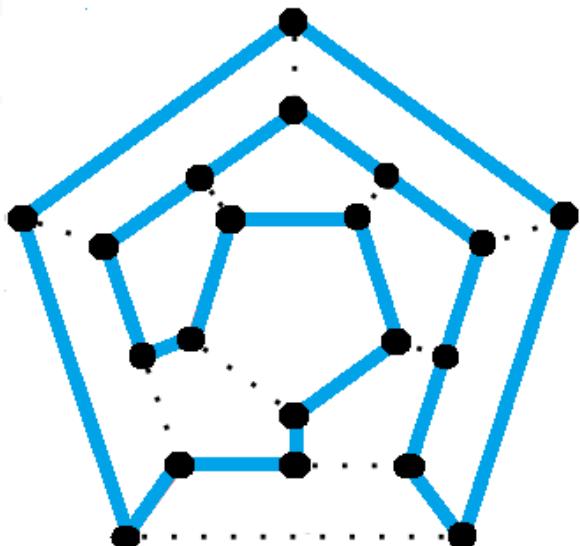


Ce inseamnă "algoritm eficient"?

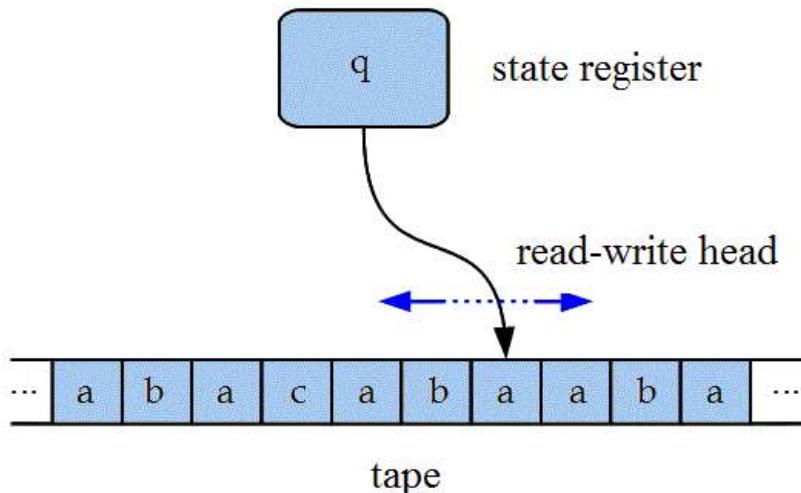


Timp polinomial: Determinism vs nedeterminims

| | | | | |
|---|---|---|-----|-------|
| 5 | 9 | | | 4 |
| 7 | 8 | 3 | 4 | 9 |
| 6 | 1 | | | 7 3 |
| 4 | 6 | 2 | 5 | |
| 3 | 8 | 5 | 7 2 | 6 4 9 |
| 1 | 7 | 4 | 8 | 2 |
| 2 | | 1 | | 4 |
| | 3 | | 4 | 8 7 |
| 7 | | 5 | 3 | 6 |



Scurt prezentare: "Turing Machine"



O mașină Turing $M=(Q, \Gamma, b, \Sigma, \rho, q_0, F)$ unde:

- Q - mulțimea stărilor
- Γ - alfabetul de lucru al mașinii
- $b \in \Gamma$ - un simbol special, numit "blank"
- $\Sigma \subset \Gamma \setminus \{b\}$ - alfabetul de intrare (alfabetul pt input)
- q_0, F - starea inițială, respectiv mulțimea stărilor finale

ρ - funcția de tranziție:

cazul determinist: $\rho: \Gamma \times Q \rightarrow \Gamma \times Q \times \{\text{left, right}\}$

cazul nedeterminist: $\rho: \Gamma \times Q \rightarrow 2^{\Gamma \times Q \times \{\text{left, right}\}}$



Clasele de Complexitate P și NP

Formal spus, în clasa problemelor din P sunt acele probleme care pot fi rezolvate în timp polinomial, $O(n^c)$, de către un sistem determinist. (P=polynomial)

Iar cele din clasa NP sunt problemele care pot rezolvațe tot în timp polinomial (!) dar de către o mașina Turing nedeterminista. (NP=nondeterministic Polynomial)

Evident ca $P \subset NP$.

Se presupune ca $P \not\subseteq NP$, totuși încă nu există o demonstrație a acestui rezultat.

Clasele de Complexitate P și NP



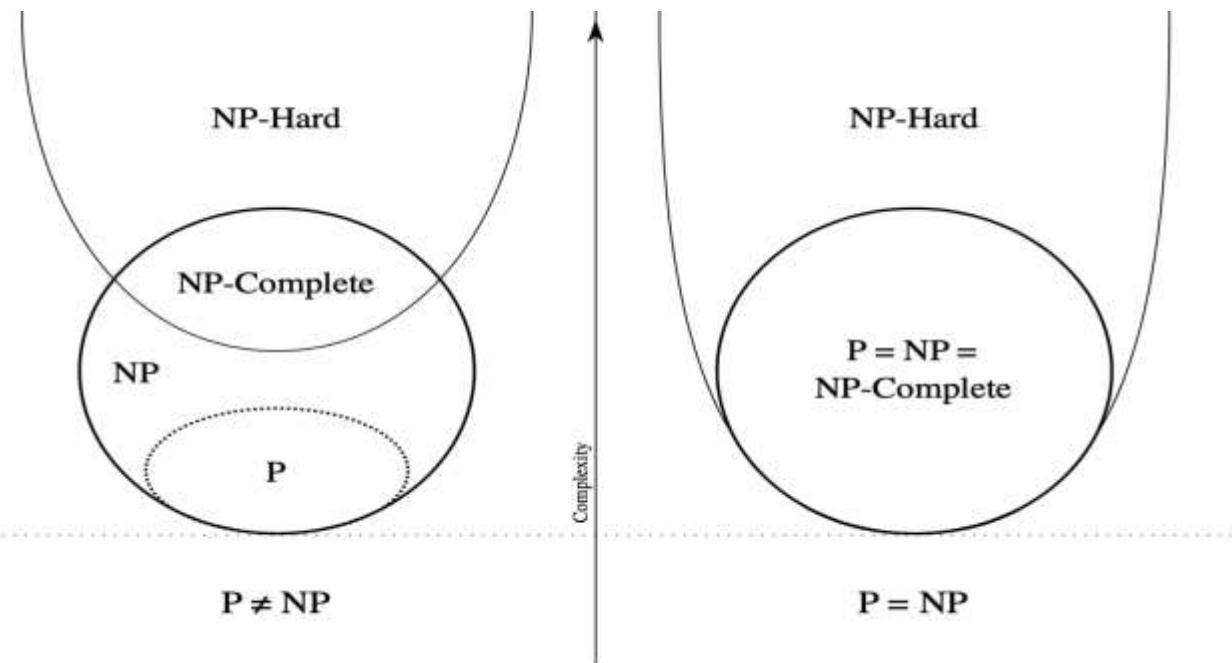
Mai ușor de înțeles:

P - clasa de probleme pentru care le putem afla soluția în timp polinomial

NP - clasa de probleme pentru care **putem verifica** în timp polinomial dacă un rezultat este soluție corectă pentru problema noastră.

| | | | | |
|---|---|---|-----|-------|
| 5 | 9 | | | 4 |
| 7 | 8 | 3 | 4 | 9 |
| 6 | 1 | | | 7 3 |
| 4 | 6 | 2 | 5 | |
| 3 | 8 | 5 | 7 2 | 6 4 9 |
| 1 | 7 | 4 | 8 | 2 |
| 2 | | 1 | | 4 |
| | 3 | | 4 | 8 7 |
| | 7 | | 5 3 | 6 |

Clasele de Complexitate P, NP, NP-C

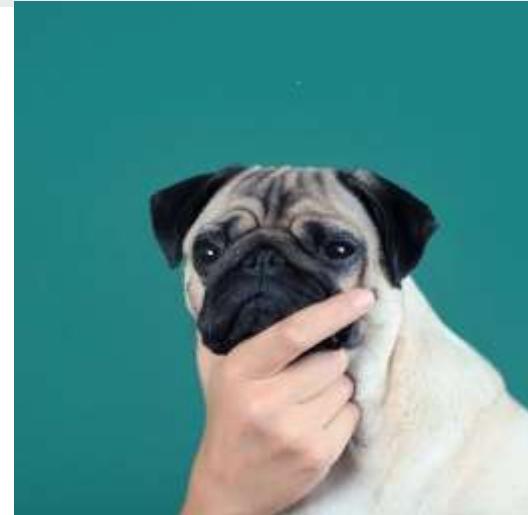


[Source for further reading](#)

Ce ne facem cu problemele din NP

Avem la dispoziție mașini (calculatoare) nedeterministe?

În cazul problemelor de optim există două soluții:
Plătim costul în timp (de multe ori nu se poate) sau ne mulțumim cu o soluție apropiată de optim, dacă nu optimă, dar ce se poate obține în timp fezabil.



Probleme de optim

O problemă de optim este de forma următoare:
Fie o mulțime de restricții. Să se construiască o soluție care nu doar îndeplinește toate restricțiile, ci minimizează/maximizeze o funcție de cost/profit.

Ex: Problema rucsacului (varianta discretă) sau probleme de acoperire minimală pentru grafuri.



Probleme de optim

Fie OPT soluția optim a problemei. Ea poate fi obținută foarte greu (practic imposibil) Două dintre căile de atac pentru astfel de probleme ar fi:

- avem un algoritm care construiește pe rand soluții la problemă, din ce în ce "mai optime", care converg către OPT. Lăsăm acest algoritm să ruleze un timp rezonabil, sau până când rezultatul nu se mai poate îmbunătăți și ne multumim cu ce avem.

(algoritmi evoluționisti)



Probleme de optim

Fie OPT soluția optimă a problemei. Ea poate fi obținută foarte greu (practic imposibil) Două dintre căile de atac pentru astfel de probleme ar fi:

- fie cazul în care OPT trebuie să minimizeze un cost.

Să reușim să construim o soluție ALG, cu

$OPT \leq ALG \leq \rho \times OPT$

(algoritmi ρ -aproximativi)





Aplicatie:

Algoritm aproximativ pentru 1/0 Knapsack Problem

[Whiteboard]

S24

S23

S25

YT channel

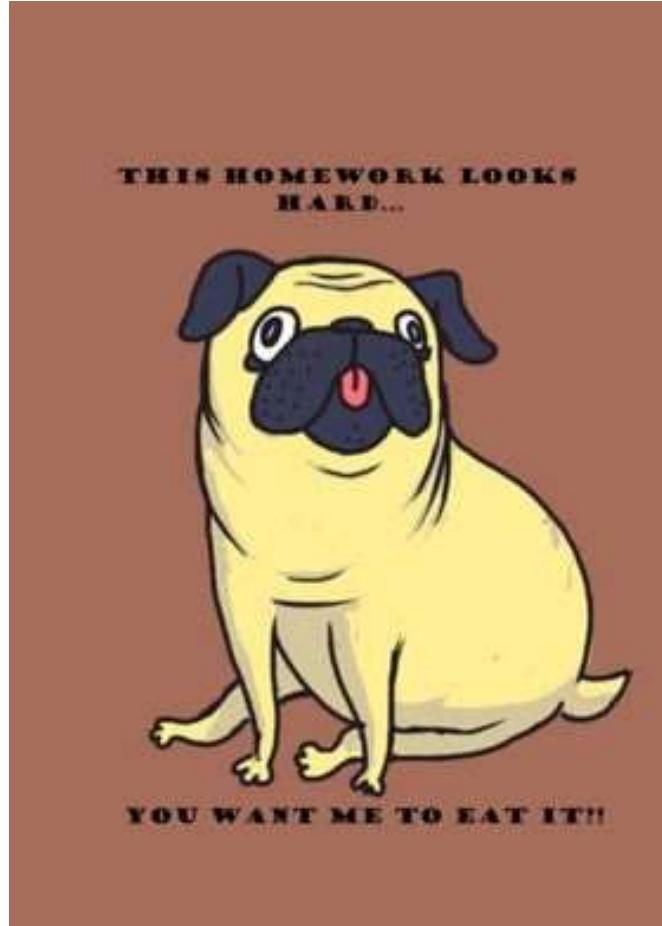
Next time:

Seminar & Lab - Recapitulare Fundamentele Algoritmilor

Curs: introducere în algoritmi aproximativi



Homework:



Algoritmi Avansați 2023

c-8

Algoritmi ρ -aproximativi

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



Din cursul anterior

Recapitulare:

reamintit ce este acela un algoritm

complexitatea unui algoritm

temp determinist vs nedeterminist

crash-course in ce inseamna P, NP, NPC



Pug 1880

Cursul prezent

- Motivație
- Terminologie de baza
- Un prim exemplu de algoritm aproximativ
- Un exemplu mai detaliat
- Un început pt Tema 1

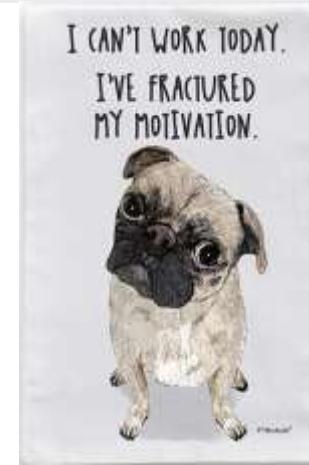


Motivăție

Q: Daca avem nevoie să aflăm răspunsul la o problemă NP-hard?

A: Nu prea sunt șanse să găsim un algoritm care să ruleze în timp polinomial

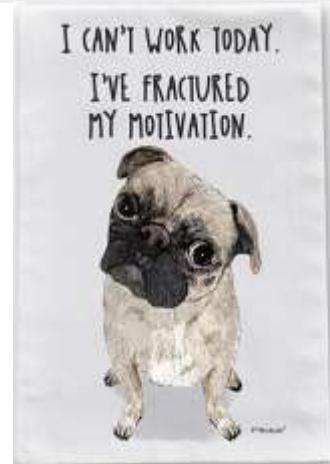
Așa că....



Motivăție

Trebuie să renunțăm măcar la unul dintre următoarele 3 elemente:

1. Găsirea unui algoritm polinomial pentru problemă
2. Găsirea unui algoritm general (pentru o instanță oarecare) a problemei
3. Găsirea soluției exacte (optime) pentru problema



Basic Terminology & Notations



Problema de Optim:

Informal spus este problema in care trebuie sa gasesti o “cea mai buna” solutie/constructie fezabila.

“Cea mai buna” - poate avea doua sensuri:

Fie avem o problema de **minimizare** precum Problema Comis-voiajorului.

Fie o problema de **maximizare** precum cea de a găsi o acoperire de cardinal maxim pentru multimea varfurilor unui graf

Basic Terminology & Notations



Problema de Optim:

Fie P - o problema de optim, și I o intrare pe aceasta problema. Vom nota cu $OPT(I)$ "valoarea" soluției optime.

În mod analog, atunci când propunem un algoritm care să ofere o soluție fezabilă pentru problema noastră, vom nota "valoarea" acelei soluții cu $ALG(I)$.

De cele mai multe ori, atunci când nu se crează confuzie, vom simplifica notațiile folosind termenii "OPT", respectiv "ALG".

Pe parcursul prezentării vom presupune că atât OPT , cât și ALG sunt ≥ 0 .

Basic Terminology & Notations



Problema de Optim:

Pentru a justifica ca un algoritm este *util*, acesta trebuie însotit de o justificare că soluția oferită este fezabilă pentru problema, precum și o relație între ALG și OPT. Aceast tip de relație este descrisă astfel:

Definiție 1

- Un algoritm ALG pentru o problema de **minimizare** 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

Basic Terminology & Notations



OBSERVAȚIE

(pt probleme de minim) Orice algoritm ρ -aproximativ este la rândul lui ρ' -aproximativ pentru orice $\rho' > \rho$. De aceea, în cazul unui algoritm ALG pentru o problemă de minimizare, spre exemplu, trebuie ca justificarea ce însotește pe ALG să ofere cea mai mică valoare ρ pentru care ALG este ρ -aproximativ.

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

Basic Terminology & Notations



Definiție 2

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

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 $ALG(\mathcal{I}) \leq \rho \times OPT(\mathcal{I})$ pentru orice intrare \mathcal{I}
2. Pentru orice $\rho' < \rho$ există un \mathcal{I} pentru care $ALG(\mathcal{I}) > \rho' \times OPT(\mathcal{I})$. Adesea totuși ne este mai la îndemână să arătăm ca există un \mathcal{I} pentru care $ALG(\mathcal{I}) = \rho \times OPT(\mathcal{I})$

O primă provocare: 1/o Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!



O primă provocare: 1/o Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!

Presupunere: Fiecare obiect are o greutate mai mică sau egală cu capacitatea rucsacului!



O primă provocare: 1/o Knapsack problem

Enunț pe scurt: Trebuie să găsim o submulțime de obiecte de valoare totală maximă, fără ca greutatea lor totală să depășească o capacitate dată a rucsacului. Obiectele sunt puse integral în rucsac sau sunt date deoparte. Nu pot fi fracționate!



Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, $G=\text{capacitatea rucsacului}$;

Pentru fiecare $O:L$

Dacă $\text{greutate}(O) \leq G$, atunci $S+ = \text{val}(O)$, $G- = \text{greutate}(O)$

$$\text{ALG}(I) = \max(S, O_p)$$

O primă provocare: 1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului!

Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, $G=\text{capacitatea rucsacului};$

Pentru fiecare $O:L$

Dacă $\text{greutate}(O) \leq G$, atunci $S+ = \text{val}(O)$, $G- = \text{greutate}(O)$

$$\text{ALG}(I) = \max(S, O_p)$$



O primă provocare: 1/0 Knapsack problem

Demonstrați că algoritmul de mai jos este un algoritm 1/2-aproximativ pentru problema 1/0 a Rucsacului! [Justificare](#)

Rezolvare propusă:

Fie L – lista obiectelor sortate după raportul valoare/greutate

Fie O_p – obiectul cu profitul cel mai mare din lista de obiecte.

$S=0$, $G=\text{capacitatea rucsacului};$

Pentru fiecare $O:L$

Dacă $\text{greutate}(O) \leq G$, atunci $S+=\text{val}(O)$, $G-=\text{greutate}(O)$

$$\text{ALG}(I) = \max(S, O_p)$$



Load Balancing Problem

Input:

- m calculatoare identice; n activitați ce trebuie 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.



Load Balancing Problem

Notății:

- $J(i)$ - 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 .
- $L_i = \sum_{j \in J(i)} t_j$

Scop: ???



Load Balancing Problem

Notății:

- $J(i)$ - 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 .
- $L_i = \sum_{j \in J(i)} t_j$

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



Load Balancing Problem

Pseudocodul:

Load-Balance(m, t_1, t_2, \dots, t_n)

for i=1 to m :

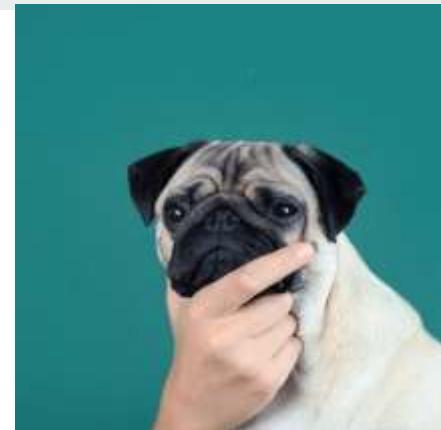
$L_i = 0; J(i) = \emptyset$ # initializare: Fiecare Load este 0 iar multimea joburilor este nula pt fiecare masina

for j=1 to n:

$i = \arg(\min\{L_k | k \in \{1, \dots, m\}\})$ # i – masina cu incarcatura cea mai mica in acest moment

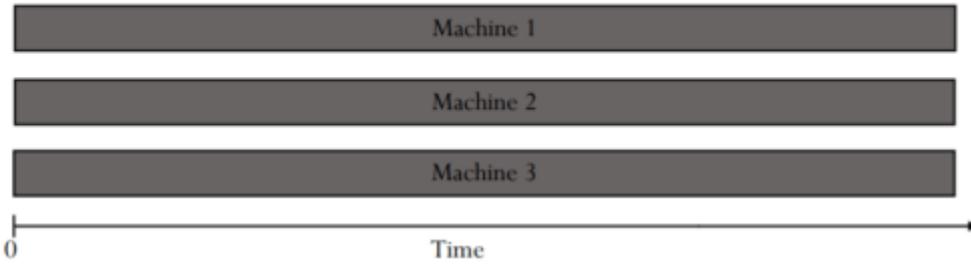
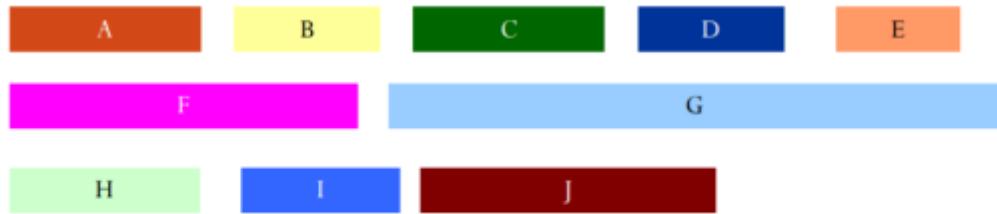
$J(i) = J(i) \cup \{j\}$

$L_i += t_j$



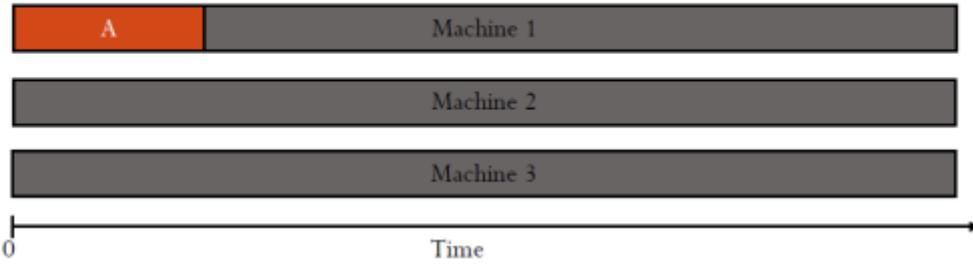
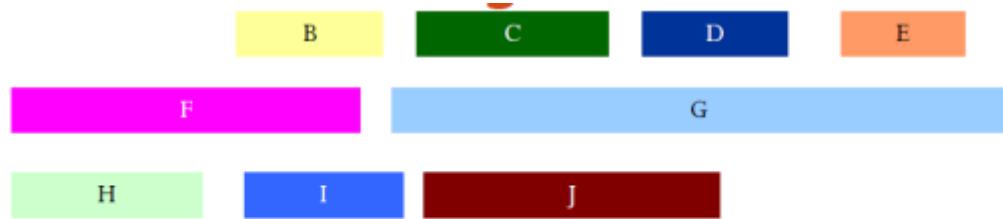


Step-by-step example



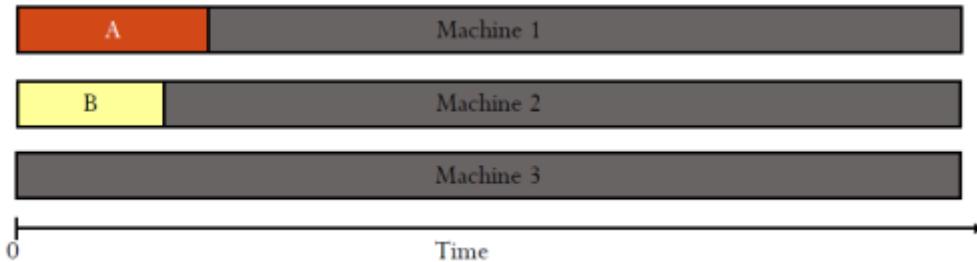
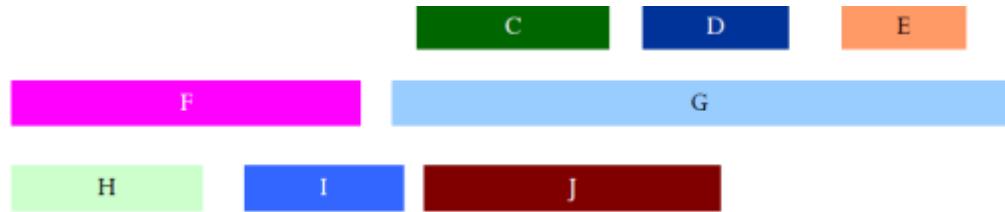


Step-by-step example



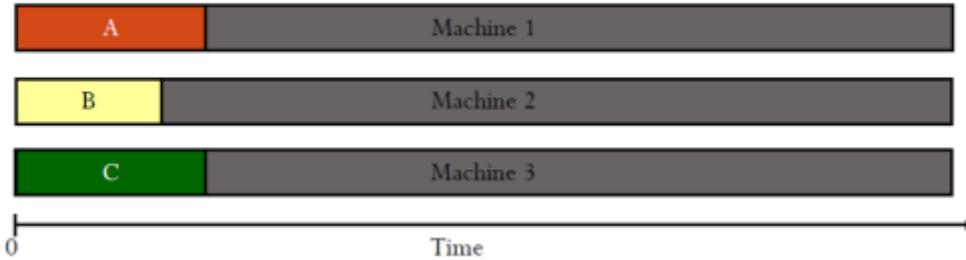


Step-by-step example



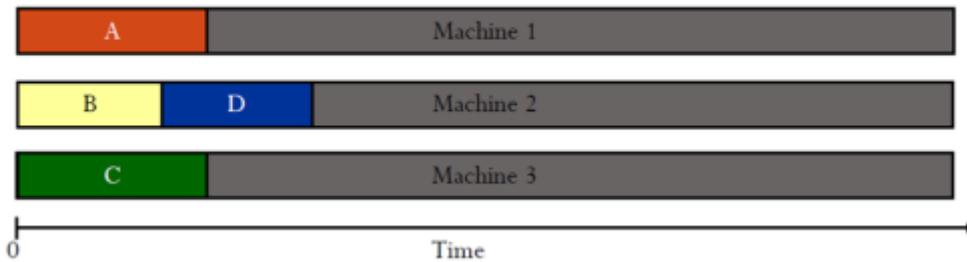


Step-by-step example



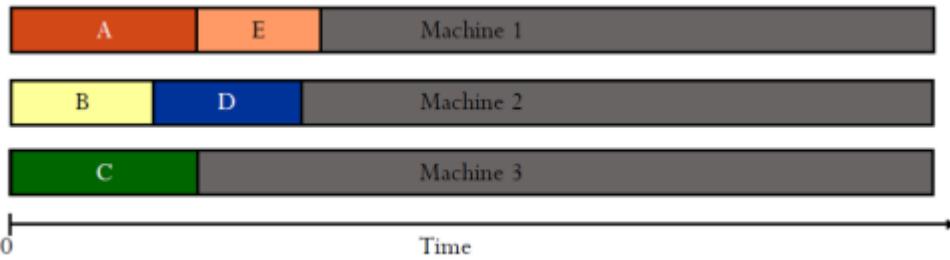


Step-by-step example



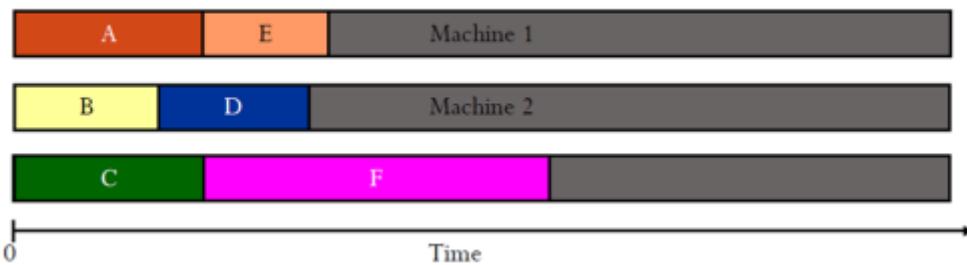


Step-by-step example



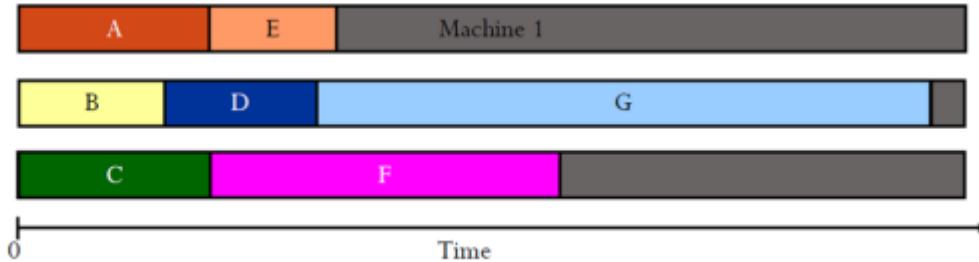


Step-by-step example



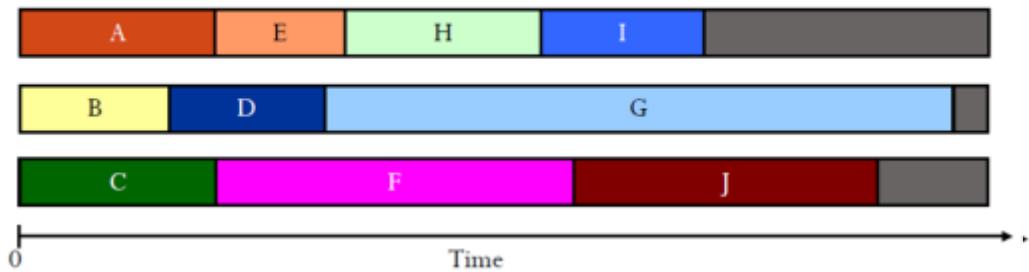


Step-by-step example





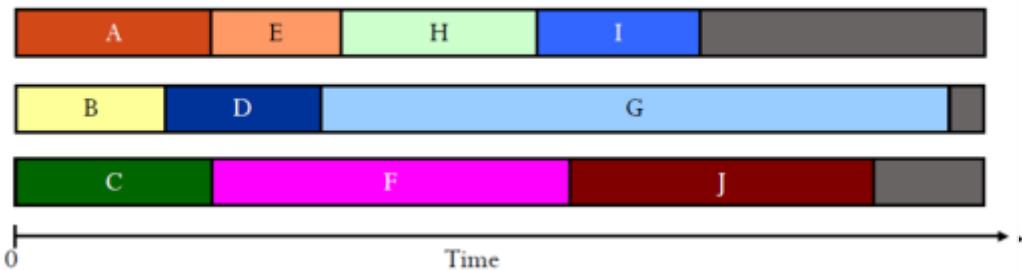
Step-by-step example (3 steps)

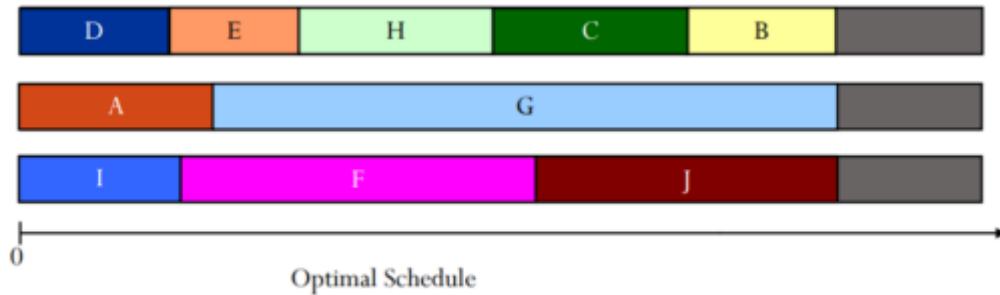


Step-by-step example (3 steps)

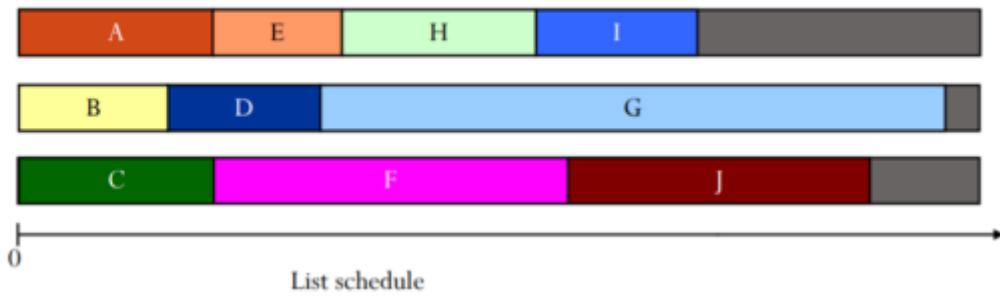


Este Optim?





NU



Care este Factorul de Aproximare?



Care este Factorul de Aproximare?

Lema 1.

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

Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $\text{ALG} = \max(L_i \mid i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $\text{ALG} \leq 2 \times \text{OPT}$



Care este Factorul de Aproximare?

Lema 1.

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

Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $ALG = \max(L_i \mid i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $ALG \leq 2 \times OPT$

Justificari



Care este Factorul de Aproximare?

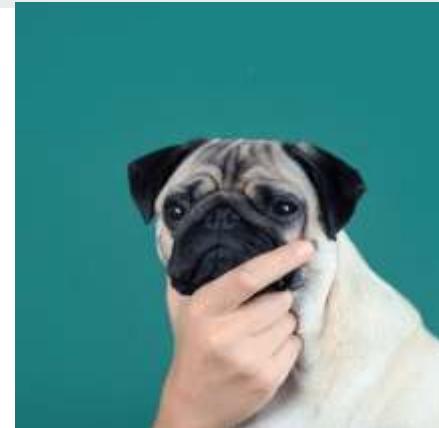
Lema 2.

Algoritmul descris anterior este un algoritm 2-Aproximativ.

Altfel spus, fie $ALG = \max(L_i | i \in \{1, \dots, m\})$ masina "cea mai incarcata". Avem de arătat că $ALG \leq 2 \times OPT$

Justificari

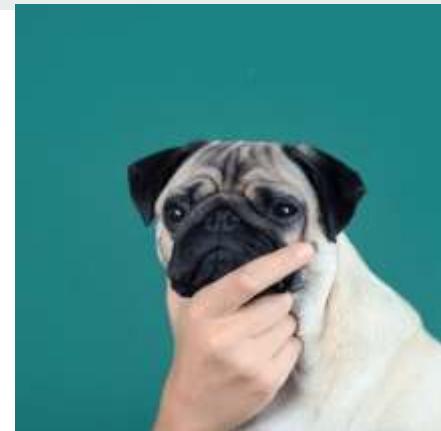
Este "tight bound"? Ce ar mai putea fi de facut?



Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) Un cu totul alt Algoritm care poate da un total alt LB.



Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) Un cu totul alt Algoritm care poate da un total alt LB.



Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit

Teorema: Algoritmul Greedy descris anterior este un algoritm $2 - 1/m$ aproximativ

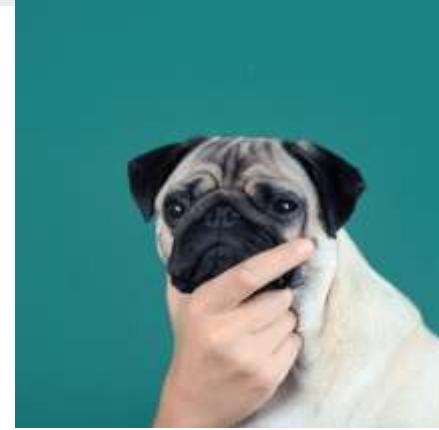


Justificari

Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) Un cu totul alt Algoritm care poate da un total alt LB.



Se poate imbunătății LB-ul?

3 abordari:

Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități

Nu se poate!

m mașini, $m(m-1)$ activități de cost 1 și o activitate de cost m



Se poate imbunătății LB-ul?

3 abordari:

- a) Acelasi Algoritm, o analiză mai buna asupra lower-bound-ului folosit
- b) Acelasi Algoritm, gasirea unui alt lower bound folosind alte inegalități
- c) Un cu totul alt Algoritm care poate da un total alt LB.



Se poate imbunătății LB-ul?

Un cu totul alt Algoritm care poate da un total alt LB.



Ordered-Scheduling Algorithm

Fie algoritmul precedent la care adaugam următoarea preprocesare:

Înainte de a fi programate, activitățile sunt sortate descrescător după timpul de lucru.



Tema (preludiu)



Lema 3.

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

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

TEOREMA 2

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

Justificari

TEMA

- profesorul de seminar poate pune întrebări pe loc legate de rezolvările din tema
- La examenul final (în funcție de cum se va face examinarea) profesorul poate pune întrebări și din tema.



Next time:

Saptamana 3:

Curs 3: TSP & Christofides

Vom discuta a doua parte din Tema 1



Algoritmi Avansați 2023

c-9

Hamiltonian Cycle Problem, TSP, bonus: Christofides' algorithm

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



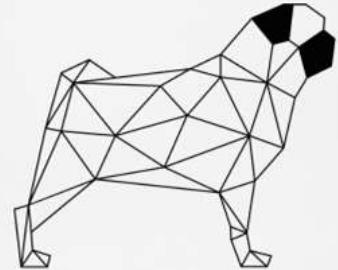
Ciclu Hamiltonian (HC-Problem)

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.

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"

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 necesara!

Traveling Salesman Problem (TSP)

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!

După cum vom vedea, nu disponem de un astfel de algoritm.



Traveling Salesman Problem (TSP)

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"

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.

Justificare



Traveling Salesman Problem (TSP)

În ciuda pesimismului oferit de rezultatul anterior, putem fi optimiști. :-)

Pug-ul nostru comis-voiajor se deplasează într-un spațiu euclidian. Deci se respectă întotdeauna regula triunghiului!



Traveling Salesman Problem (TSP)

În ciuda pesimismului oferit de rezultatul anterior, putem fi optimiști. :-)

Pug-ul nostru comis-voiajor se deplasează într-un spațiu euclidian. Deci se respectă întotdeauna regula triunghiului!

Regula triunghiului (recap): Pentru orice triunghi cu lungimea laturilor $L_1 \geq L_2 \geq L_3$, avem $L_3 + L_2 \geq L_1$



Traveling Salesman Problem (TSP)

În ciuda pesimismului oferit de rezultatul anterior, putem fi optimiști. :-)



Pug-ul nostru comis-voiajor se deplasează într-un spațiu euclidian. Deci se respectă întotdeauna regula triunghiului!

Regula triunghiului (recap): Pentru orice triunghi cu lungimea laturilor $L_1 \geq L_2 \geq L_3$, avem $L_3 + L_2 \geq L_1$

Pentru un graf complet, ponderat, care respectă regula triunghiului, există algoritmi aproximativi pentru rezolvarea TSP!

It's important to have a
twinkle
in your
wrinkle.

—Author Unknown



Traveling Salesman Problem (TSP)

În ciuda pesimismului oferit de rezultatul anterior, putem fi optimiști. :-)

Pug-ul nostru comis-voiajor se deplasează într-un spațiu euclidian. Deci se respectă întotdeauna regula triunghiului!

Regula triunghiului (recap): Pentru orice triunghi cu lungimea laturilor $L_1 \geq L_2 \geq L_3$, avem $L_3 + L_2 \geq L_1$

Pentru un graf complet, ponderat, care respectă regula triunghiului, există algoritmi aproximativi pentru rezolvarea TSP!!!



Traveling Salesman Problem (TSP)



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 .

Observație 2:

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

Justificare

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:



Arbore parțial de cost minim - algoritmi și timpi de lucru

Asemănare dintre MST și TSP?

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:



Arbore parțial de cost minim - algoritmi și timpi de lucru

Asemănare dintre MST și TSP?

**Ambele caută un traseu de cost total minim care să cuprindă
toate nodurile**

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:



Arbore parțial de cost minim - algoritmi și timpi de lucru

Diferențe dintre MST și TSP?

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:



**Arbore parțial de cost minim - algoritmi și timpi de lucru
Diferențe dintre MST și TSP?**

- unul este un arbore, altul este un ciclu
- una este P iar alta este NP hard!

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:

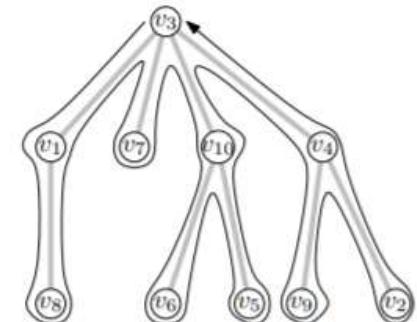
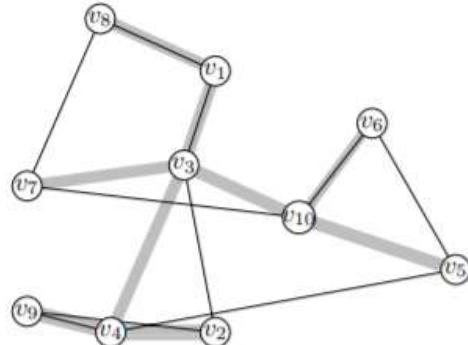


Lema 3:

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

$\text{OPT} \geq \text{MST}$

Justificare



Traveling Salesman Problem (TSP) algoritm 2-aproximativ:

ApproxTSP(G)

1: Calculam arborele parțial de cost minim T pentru graful G .

2: Alegem un nod $u \in T$ pe post de radacina.

3: $\Gamma = \emptyset$.

4: Parcursere (u, Γ)

5: concatenam nodul u la finalul lui Γ pentru a închide un ciclu.

6: return Γ



Traveling Salesman Problem (TSP) algoritm 2-aproximativ:



Parcurgere(u, Γ)

1: Concatenam pe u la Γ .

2: pentru fiecare v , fiu al lui u :

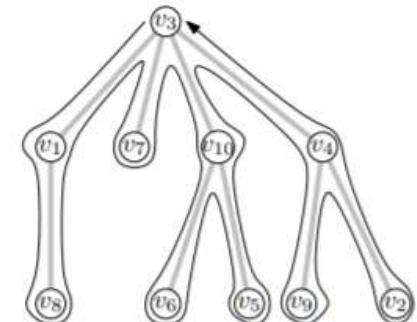
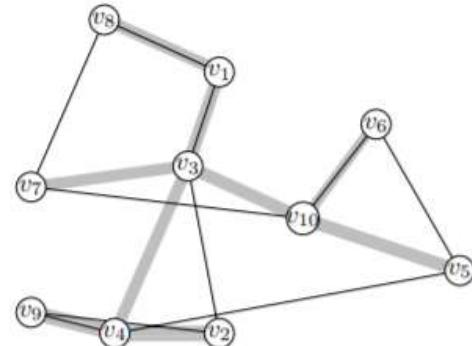
3: Parcurgere(v, Γ)

Traveling Salesman Problem (TSP) algoritm 2-aproximativ:

Teorema 4:

Algoritmul descris anterior este un algoritm 2-aproximativ
pentru TSP

Justificare



Traveling Salesman Problem (TSP)

Se poate oare mai bine?



Traveling Salesman Problem (TSP)

Se poate oare mai bine?

DA!





Traveling Salesman Problem (TSP) BONUS!

Se poate oare mai bine?

Algoritmul lui Christofides!

Un algoritm $3/2$ aproximativ

Traveling Salesman Problem (TSP) BONUS!



ChristofidesTSP(G)

1: Calculam T, un APCM in G

2: Fie $V^* \subset V$ multimea de varfuri de grad impar din T . (va exista mereu un numar par de varfuri de grad impar)

3: Fie graful $G^* = (V^*, E^*)$ - graful complet induș de V^* .

4: Calculam M - cuplajul perfect de pondere totala minima pentru G^*

5: reunim multimile M si T ,

6: deoarece toate nodurile au grad par, putem evidenția un ciclu Eulerian Γ in multigraful induș de $M \cup T$

7: Pentru fiecare varf din Γ , eliminam toate "dublurile" sale, reducand costul total.

8: return Γ

Next time:

Vertex Cover Problem
Linear Programming



Algoritmi Avansați 2023

C-10

Vertex Cover Problem, Linear Programming

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



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.



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.



Vertex cover problem

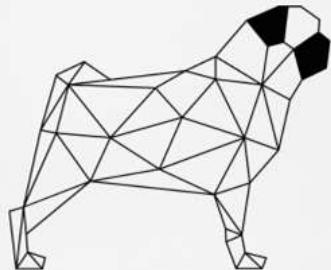
Problema formală:

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

Numim "acoperire" o submulțime $S \subset V$ cu proprietatea 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!



Vertex cover problem

Problema formală:

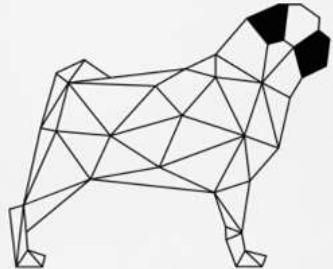
Fie un graf neorientat $G=(V,E)$.

Numim "acoperire" o submulțime $S \subset V$ cu proprietatea 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.



Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E$; $S=\emptyset$;

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S



Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S



Q1. Mulțimea de noduri S este o acoperire
pentru graful G ?
DA/NU

Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S



Q1 Multimea de noduri S este o acoperire
pentru graful G ?
DA!

Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S



Q2. Algoritmul de alături:

- a) Este un algoritm care generează mereu soluția optimă
- b) Este un algoritm 3-aproximativ pentru VCP
- c) poate furniza și un răspuns de 100 de ori mai slab decât soluția optimă

Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

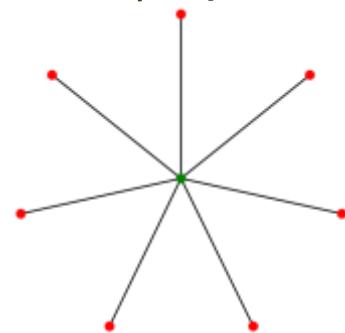
$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S

Q2. Algoritmul de alături:

poate furniza și un răspuns de 100
de ori mai slab decât soluția optimă



Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{x\}$

stergem din E' toate muchiile
incidente lui x

return S



Q3. Cum putem modifica algoritmul
alăturat astfel încât să îmbunătățim
rezultatul?

Vertex cover problem

Fie următorul algoritm:

INPUT: $G=(V,E)$

$E'=E; S=\emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S=S \cup \{$  $\}$

ștergem din E' toate muchiile incidente [lui x și lui y](#)

return S



Q3. Cum putem modifica algoritmul alăturat astfel încât să îmbunătățim rezultatul?

Vertex cover problem

Fie următorul algoritm:

ApproxVertexCover (V,E)

$E' = E; S = \emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S = S \cup \{x, y\}$

ștergem din E' toate muchiile incidente lui x și lui y

return S



Deși pare o abordare cel puțin ciudată, algoritmul alăturat este un algoritm 2-aproximativ pentru vertex cover problem!

Vertex cover problem

Fie următorul algoritm:

ApproxVertexCover (V,E)

$E' = E; S = \emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S = S \cup \{x,y\}$

ștergem din E' toate muchiile incidente lui x și lui y

return S



Deși pare o abordare cel puțin ciudată,
algoritmul alăturat

- 1) generează o acoperire validă
- 2) este un algoritm 2-aproximativ

Vertex cover problem

Fie următorul algoritm:

ApproxVertexCover (V, E)

$E' = E; S = \emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x, y) \in E'$;

$S = S \cup \{x, y\}$

ștergem din E' toate muchiile incidente lui x și lui y

return S



Lema 1. Fie $G = (V, E)$ un graf neorientat și OPT cardinalul unei acoperiri de grad minim a lui G . Fie $E^* \subset E$ o mulțime de muchii nod disjuncte.

Atunci avem că $\text{OPT} \geq |E^*|$

Demonstratie

Vertex cover problem

Fie următorul algoritm:

ApproxVertexCover (V,E)

$E' = E; S = \emptyset;$

cât timp $E' \neq \emptyset$:

aleg $(x,y) \in E'$;

$S = S \cup \{x,y\}$

ștergem din E' toate muchiile incidente lui x și lui y

return S



Teorema 2. Algoritmul alăturat este un algoritm 2 aproximativ pentru VCP.

Demonstratie

Complicam Problema! Weighted Vertex Problem.

Fie un graf $G=(V,E)$ - un graf simplu, si $f:V \rightarrow R_+$, care asociază fiecărui vârf, un cost

Trebuie să găsim o acoperire de varfuri S astfel încât să minimizăm: $\sum_{v \in S} f(v)$

Este dificil să găsim un algoritm aproximativ pt aceasta problemă prin metodele "tradiționale"

Tb sa gasim o abordare noua!



Programare Liniara

O problemă de programare liniară arată în felul următor:

- o funcție de "cost" cu d variabile x_1, x_2, \dots, x_d
- un set de n constrângerile liniare peste variabilele x_1, x_2, \dots, x_d



Scopul este asignarea de valori pentru variabilele de tip x_i , astfel încât să minimizăm (sau, după caz, să maximizăm) funcția de cost, respectând totodată toate cele n constrângeri

Programare Liniara

O problemă de programare liniară arată în felul următor:

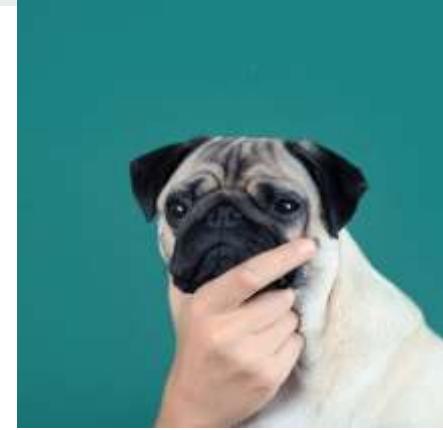
Ex:

Tb minimizat $c_1x_1 + \dots + c_dx_d$

astfel încât

$$a_{1,1}x_1 + \dots + a_{1,d}x_d \leq b_1$$
$$a_{2,1}x_1 + \dots + a_{2,d}x_d \leq b_2$$

...

$$a_{n,1}x_1 + \dots + a_{n,d}x_d \leq b_n$$


Programare Liniara

O constrângere poate conține adunări de variabile,
poate folosi inegalități de orice tip ($<$, $>$, $>=$, $<=$, $=$)



O constrângere nu poate fi optională! Toate constrângerile sunt
"binding"

În constrangeri nu pot apărea elemente de forma " $x_i * x_j$ " sau " x^2 " -
trebuie să fie liniare!

Programare Liniara

O problemă de programare liniară arată în felul următor:

Ex:

Tb minimizat $c_1x_1 + \dots + c_dx_d$
astfel încât

$$a_{1,1}x_1 + \dots + a_{1,d}x_d \leq b_1$$

$$a_{2,1}x_1 + \dots + a_{2,d}x_d \leq b_2$$

...

$$a_{n,1}x_1 + \dots + a_{n,d}x_d \leq b_n$$

Astfel de sisteme pot fi rezolvate în timp polinomial prin algoritmi *simplex* (vezi cursul de Tehnici de Optimizare).



Programare Liniara

O problemă de programare liniară arată în felul următor:

Ex:

Tb minimizat $c_1x_1 + \dots + c_dx_d$
astfel încât

$$a_{1,1}x_1 + \dots + a_{1,d}x_d \leq b_1$$

$$a_{2,1}x_1 + \dots + a_{2,d}x_d \leq b_2$$

...

$$a_{n,1}x_1 + \dots + a_{n,d}x_d \leq b_n$$

Astfel de sisteme pot fi rezolvate în timp polinomial prin algoritmi *simplex* (vezi cursul de Tehnici de Optimizare).

OBSERVAȚIE:

Algoritmii simplex rezolvă inegalitatea pentru **x_i - numere reale!**



Revenim la WVCP (slide 16)

Putem formula această problemă ca o problemă de programare liniară:

Demonstratie

Astfel de sisteme pot fi rezolvate în timp polinomial prin algoritmi *simplex* (vezi cursul de Tehnici de Optimizare).

OBSERVAȚIE:

Algoritmii simplex rezolvă inegalitatea pentru x_i - **numere reale!**



Further reading:

[Suport de curs saptamana 4 \(engl\)](#)



Algoritmi Avansați 2023

C-11

Genetic Algorithms

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



What are they used for?

Sunt utilizati în probleme de optim, pentru care

- spațiul de căutare a soluțiilor posibile este mare
- nu se cunosc algoritmi exacți mai rapizi Furnizează o soluție care nu este neapărat optimă.
- Căutarea în spațiul soluțiilor candidat – euristică, bazată pe principii ale evoluției în genetică



Denumirea lor se datorează preluării unor mecanisme din biologie: moștenirea genetică și evoluția naturală pentru populații de indivizi

What are they used for?

Aplicații

- Robotică, bioinformatică, inginerie
- Probleme de trafic, rutare, proiectare
- Criptare, code-breaking
- Teoria jocurilor
- Clustering etc



Informal talk



Informal talk



Exemplu ilustrativ (scop didactic)



Maximul unei funcții pozitive

Fie $f:D \rightarrow \mathbb{R}$. Să se calculeze

$$\max\{ f(x) \mid x \in D \}, \text{ unde } D = [a, b].$$

- Presupunem $f(x) > 0, \forall x \in D$.

Algoritmi Genetici: Notiuni

Cromozom = mulțime ordonată de elemente (gene) ale căror valoare (alele) determină caracteristicile unui individ

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|



Algoritmi Genetici: Notiuni

Cromozom = mulțime ordonată de elemente (gene) ale căror valoare (alele) determină caracteristicile unui individ

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|

Populație = mulțime de indivizi care trăiesc într-un mediu la care trebuie să se adapteze



Algoritmi Genetici: Notiuni

Cromozom = mulțime ordonată de elemente (gene) ale căror valoare (alele) determină caracteristicile unui individ

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|



Populație = mulțime de indivizi care trăiesc într-un mediu la care trebuie să se adapteze

Fitness (adecvare) = măsură a gradului de adaptare la mediu pentru fiecare individ (funcție de fitness)

Algoritmi Genetici: Noțiuni

Generație = etapă în evoluția populației



Algoritmi Genetici: Noțiuni

Generație = etapă în evoluția populației

Selectie = proces prin care sunt promovați indivizi cu grad ridicat de adaptare la mediu



Algoritmi Genetici: Noțiuni

Generație = etapă în evoluția populației

Selectie = proces prin care sunt promovați indivizi cu grad ridicat de adaptare la mediu

Operatori genetici:

- **Încrucișare** (combinare, crossover) - indivizi din noua generație moștenesc caracteristicile părinților



Algoritmi Genetici: Noțiuni

Generație = etapă în evoluția populației

Selectie = proces prin care sunt promovați indivizi cu grad ridicat de adaptare la mediu

Operatori genetici:

- **Încrucișare** (combinare, crossover) - indivizi din noua generație moștenesc caracteristicile părinților
- **mutație** - indivizi din noua generație pot dobândi și caracteristici noi



Structura pas-cu-pas a unui Algoritm Genetic



Algoritm

- $t=0$
- Consideră, o populație inițială $P(0)$: alegem aleator indivizi din intervalul D
- $t=t+1$



Algoritm

- $t=0$
 - Consideră, o populație inițială $P(0)$: alegem aleator indivizi din intervalul D
 - Cât timp nu există condiția de terminare:
-
- $t=t+1$



Condiții de terminare

- număr maxim de iterații / durată de execuție
- stabilizarea performanței medii/maxime
- am obținut o soluție suficient de bună



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
- $t=t+1$



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 $P^2(t)$
- $t=t+1$



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 $P(t+1)$
- $t=t+1$



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 $P(t+1)$
 - **optional: la $P(t+1)$ se adaugă elementul/elementele elitiste din $P(t)$**
 - $t=t+1$



Algoritm

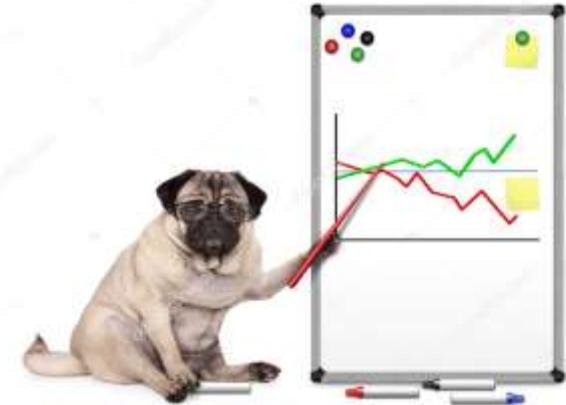
- $t=0$
- Consideră, **o populație inițială $P(0)$: alegem aleator indivizi** din intervalul D
- Cât timp nu există **condiția de terminare**:
 - construim o populație nouă $P(t+1)$ pe baza indivizilor din $P(t)$ astfel:
 - **selecție**: generează o populație intermediară $P^1(t)$ selectând indivizi 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 $P(t+1)$
 - **optional: la $P(t+1)$ se adaugă elementul/elementele elitiste din $P(t)$**
 - $t=t+1$



Exemplu: maximizarea unei funcții pozitive

Date de intrare + parametri de control

- intervalul $[a, b]$
- precizia p (numărul de zecimale)
- dimensiunea populației n
- numărul de generații
- probabilitatea de încrucișare pc
- probabilitatea de mutație pm



Populația

Dimensiune (număr de cromozomi) :

n - fixă, dată
constantă pe parcursul algoritmului



Populația

Dimensiune (număr de cromozomi) :

n - fixă, dată
constantă pe parcursul algoritmului



Codificare = cum asociem unei configurații din spațiul de căutare un cromozom

În general: codificare binară, lungime fixă

Populația

Dimensiune (număr de cromozomi) :

n - fixă, dată
constantă pe parcursul algoritmului



Codificare = cum asociem unei configurații din spațiul de căutare un cromozom

În general: codificare binară, lungime fixă

Cum calculăm lungimea pentru puncte din $D = [a,b]$?

Populația

Dimensiune (număr de cromozomi) :

n - fixă, dată
constantă pe parcursul algoritmului



Codificare = cum asociem unei configurații din spațiul de căutare un cromozom

În general: codificare binară, lungime fixă

Cum calculăm lungimea pentru puncte din $D = [a,b]$?

Depinde de nivelul de discretizare al intervalului $[a,b]$

Codificare

Codificare = cum asociem unei configurații din spațiul de căutare un cromozom
În general: codificare binară, lungime fixă



Pentru $D = [a,b]$ și o precizie p dată (ca număr)



male):

- discretizarea intervalului $\Rightarrow (l, u)$ subintervale (elemente)

Codificare

Codificare = cum asociem unei configurații din spațiul de căutare un cromozom
În general: codificare binară, lungime fixă



Pentru $D = [a,b]$ și o precizie p dată (ca număr de zecimale):

- discretizarea intervalului $\Rightarrow (b-a) \times 10^p$ subintervale (elemente)

Codificare

Codificare = cum asociem unei configurații din spațiul de căutare un cromozom
În general: codificare binară, lungime fixă



Pentru $D = [a,b]$ și o precizie p dată (ca număr de zecimale):

- discretizarea intervalului $\Rightarrow (b-a) \times 10^p$ subintervale (elemente)
- lungimea cromozomă



Codificare

Codificare = cum asociem unei configurații din spațiul de căutare un cromozom
În general: codificare binară, lungime fixă



Pentru $D = [a,b]$ și o precizie p dată (ca număr de zecimale):

- discretizarea intervalului $\Rightarrow (b-a) \times 10^p$ subintervale (elemente)
- $2^{l-1} < (b-a)10^p \leq 2^l \Rightarrow l = \lceil \log_2((b-a)10^p) \rceil$

Codificare

Codificare = cum asociem unei configurații din spațiul de căutare un cromozom
În general: codificare binară, lungime fixă



Pentru $D = [a,b]$ și o precizie p dată (ca număr de zecimale):

- discretizarea intervalului $\Rightarrow (b-a) \times 10^p$ subintervale (elemente)
- $2^{l-1} < (b-a)10^p \leq 2^l \Rightarrow l = \lceil \log_2((b-a)10^p) \rceil$
- $X_{(2)} \rightarrow X_{(10)} \rightarrow \frac{b-a}{2^l - 1} X_{(10)} + a$: translație liniară

Populație

Populația inițială se generează aleator



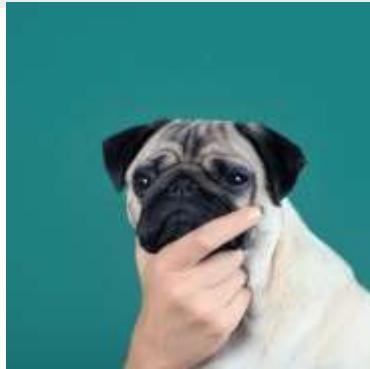
Funcția de fitness

- se pot folosi distanțe cunoscute (euclidiană, Hamming)
- pentru problema de maxim funcția este chiar f



Selectia

determinarea unei populații intermediare, ce conține indivizi care vor fi supuși operatorilor genetici



- Selectie proporțională
- Selectie elitistă
- Selectie turneu
- Selectie bazată pe ordonare

Selectia proporcională

Presupunem $P(t)=\{X_1, \dots, X_n\}$

asociem fiecarui individ X_i o probabilitate p_i de a fi selectat, în funcție de performanța acestuia (dată de funcția de fitness \hat{v})



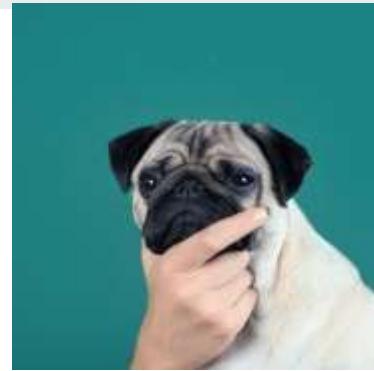
Selectia proporcională

Presupunem $P(t) = \{X_1, \dots, X_n\}$

asociem fiecarui individ X_i o probabilitate p_i de a fi selectat, în funcție de performanța acestuia (dată de f(x))

$$p_i = \frac{f(X_i)}{F}$$

$$F = \sum_{j=1}^n f(X_j) = \text{performanța totală a populației}$$



Selectia proporcională

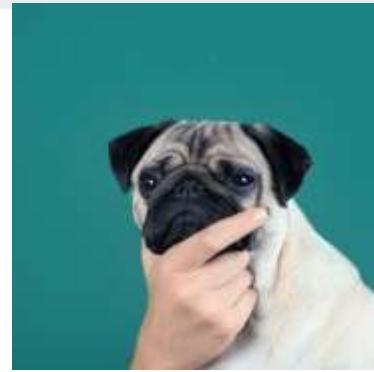
Presupunem $P(t) = \{X_1, \dots, X_n\}$

asociem fiecarui individ X_i o probabilitate p_i de a fi selectat, în funcție de performanța acestuia (dată de f(x))

$$p_i = \frac{f(X_i)}{F}$$

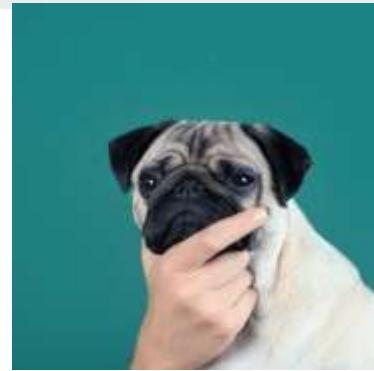
$$F = \sum_{j=1}^n f(X_j) = \text{performanța totală a populației}$$

folosind **metoda ruletei** selectăm n indivizi (!copii), cu distribuția de probabilitate (p_1, p_2, \dots, p_n)



Selectia proporcională - metoda ruletei

Folosind metoda ruletei selectăm n indivizi (!copii), cu distribuția de probabilitate (p_1, p_2, \dots, p_n)

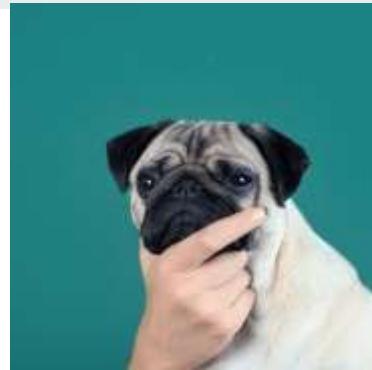


Etapa de Selecție:

- $P^1(t)=\emptyset$
- repetă de n ori:
 - generează j cu probabilitatea (p_1, p_2, \dots, p_n) folosind **metoda ruletei**
 - genereaza u variabila uniformă pe $[0,1]$
 - determină indicele j astfel încât u este între $q_{j-1} = p_1 + \dots + p_{j-1}$ și $q_j = p_1 + \dots + p_j$ (cu convenția $q_0 = 0$)
 - adaugă la populația selectată $P^1(t)$ o copie a lui X_j

Selectia

Selectie elitistă = trecerea explicită a celui mai bun individ în generația următoare



Selectie turneu = se aleg aleatoriu k indivizi din populație și se selectează cel mai performant dintre ei

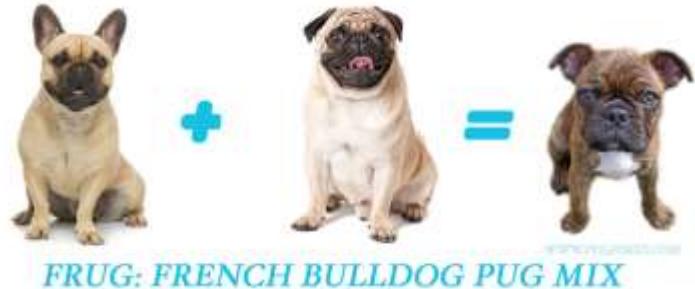
Selectie bazată pe ordonare = se ordonează indivizii după performanță și li se asociază câte o probabilitate de selecție în funcție de locul lor după ordonare

Încrucișarea

Permite combinarea informațiilor de la părinți

Doi părinți dau naștere la doi descendenți

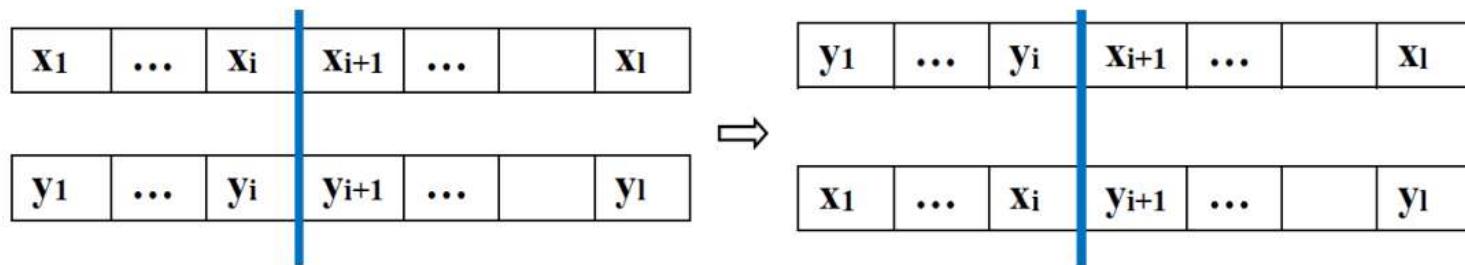
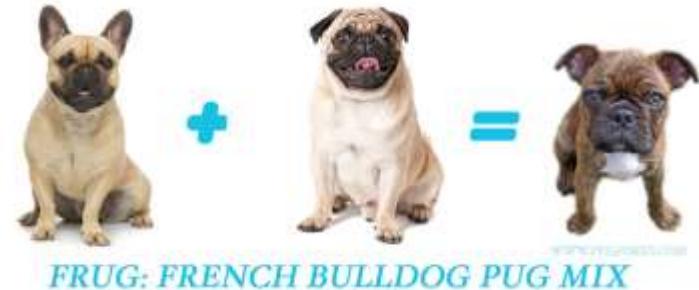
- cu un punct de tăietură (de rupere)
- cu mai multe puncte de rupere
- uniformă
- etc



Încrucișarea

Cu un punct de tăietură (de rupere)

2 părinți => **2 indivizi noi** care iau locul părinților în populație



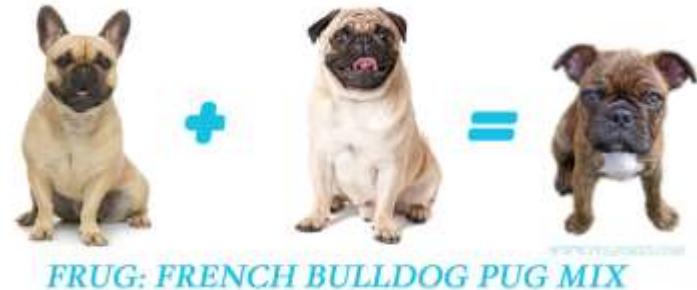
i – punct de rupere generat aleator

Încrucișarea

Cu un punct de tăietură (de rupere)

Nu toți cromozomii din $P^1(t)$ participă la încrucișare.

Un cromozom participă la încrucișare cu o probabilitate fixată p_c (probabilitate de încrucișare – dată de intrare)



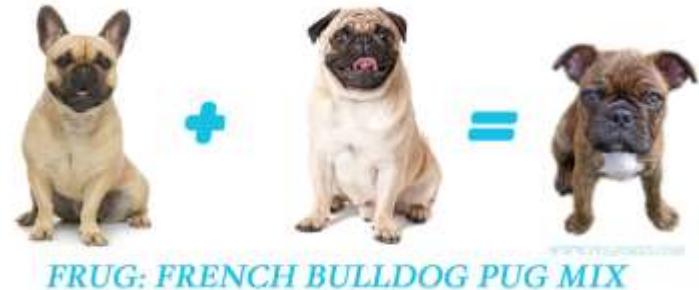
Încrucișarea

Cu un punct de tăietură (de rupere)

Un cromozom participă la încrucișare cu o probabilitate fixată p_c (probabilitate de încrucișare – dată de intrare)

Etapa de încrucișare:

- Notăm $P^1(t) = \{X_1, X_2, \dots, X_n\}$



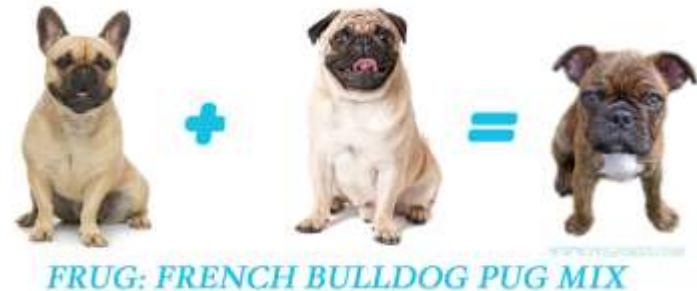
Încrucișarea

Cu un punct de tăietură (de rupere)

Un cromozom participă la încrucișare cu o probabilitate fixată p_c (probabilitate de încrucișare – dată de intrare)

Etapa de încrucișare:

- Notăm $P^1(t) = \{X_1, X_2, \dots, X_n\}$
- for $i = 1, n$
 - generează **u** variabilă uniformă pe $[0,1]$



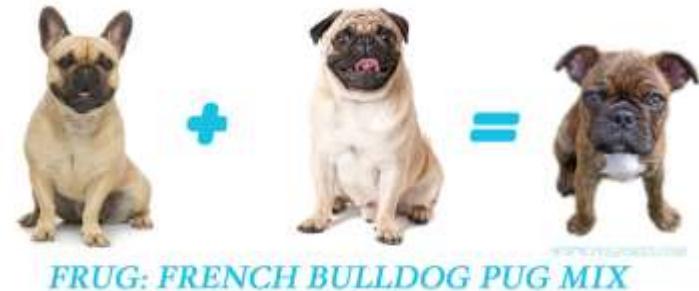
Încrucișarea

Cu un punct de tăietură (de rupere)

Un cromozom participă la încrucișare cu o probabilitate fixată pc (probabilitate de încrucișare – dată de intrare)

Etapa de încrucișare:

- Notăm $P^1(t) = \{X_1, X_2, \dots, X_n\}$
- for $i = 1, n$
 - genereaza u variabila uniformă pe $[0, 1]$
 - daca $u < pc$ atunci marcheaza (va participa la incruisare)



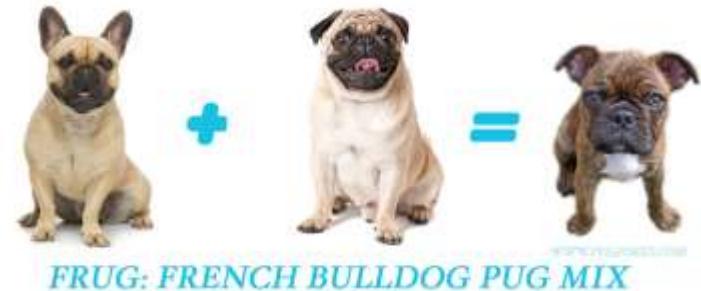
Încrucișarea

Cu un punct de tăietură (de rupere)

Un cromozom participă la încrucișare cu o probabilitate fixată pc (probabilitate de încrucișare – dată de intrare)

Etapa de încrucișare:

- Notăm $P^1(t) = \{X_1, X_2, \dots, X_n\}$
- for $i = 1, n$
 - genereaza u variabila uniformă pe $[0, 1]$
 - daca $u < pc$ atunci marcheaza (va participa la încrucișare)
- formează perechi disjuncte de cromozomi marcați și
- aplică pentru fiecare pereche operatorul de încrucișare;



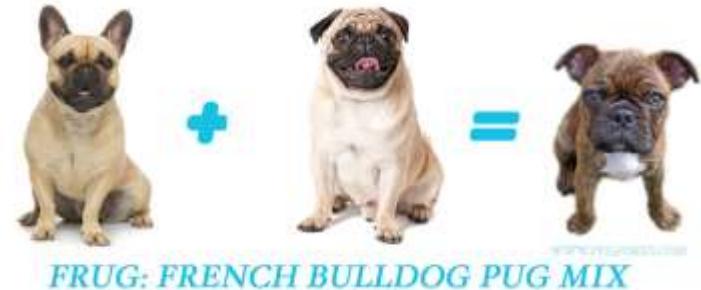
Încrucișarea

Cu un punct de tăietură (de rupere)

Un cromozom participă la încrucișare cu o probabilitate fixată pc (probabilitate de încrucișare – dată de intrare)

Etapa de încrucișare:

- Notăm $P^1(t) = \{X_1, X_2, \dots, X_n\}$
- for $i = 1, n$
 - genereaza u variabila uniformă pe $[0, 1]$
 - daca $u < pc$ atunci marcheaza (va participa la încrucișare)
- formează perechi disjuncte de cromozomi marcați și
- aplică pentru fiecare pereche operatorul de încrucișare;
- **descendenții rezultați înlocuiesc părinții în populație**



Mutăția

schimbarea valorilor unor gene din cromozom

asigură diversitatea populației

probabilitatea de mutație pm – dată de intrare



Mutăția

Etapa de mutație - **Varianta 1** (mutație rară):

- Notăm $P^2(t) = \{X_1, \dots, X_n\}$ populația obținută după încrucișare
- for $i = 1, n$
 - genereaza u variabila uniformă pe $[0,1]$
 - daca $u < pm$ atunci generează o poziție aleatoare p și
 - trece gena p din cromozomul X_i la complement $0 \leftrightarrow 1$



Mutăția

Etapa de mutație - Varianta 2 :

- Notăm $P^2(t) = \{X_1, \dots, X_n\}$ populația obținută după încrucișare
- for $i = 1, n$
 - for $j = 1, \text{len}(X_i)$
 - genereaza u variabila uniformă pe $[0,1]$
 - daca $u < pm$ atunci
 - trece gena j din cromozomul X_i la complement $0 \leftrightarrow 1$



Alte exemple



Next Time



Algoritmi Avansați 2023

C-12

Randomized Algorithms

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



Cuprins

Descriere

Probleme:

- Check Matrix multiplication
- Quicksort



Algoritmi probabilisti

- Ce sunt?



Algoritmi probabilisti

- Ce sunt algoritmii probabilisti?

Orice algoritm care generează aleator un element $r \in \{1, 2, \dots, R\}$ și efectuează decizii în funcție de valoarea acestuia



Algoritmi probabilisti

- Ce sunt algoritmii probabilisti?

Orice algoritm care generează aleator un element $r \in \{1, 2, \dots, R\}$ și efectuează decizii în funcție de valoarea acestuia

Un astfel de algoritm poate rula un număr diferit de pași și poate oferi output-uri diferite pe aceeași intrare. Astfel devine relevant să avem mai multe iterări ale algoritmului pe un același input!



Algoritmi probabilisti

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



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 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 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.



Algoritmi Monte Carlo

Exemplu de problemă



Algoritmi Monte Carlo

Matrix Multiplication:



Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.



Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.

Alternative:



Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.

Alternative:

- Implementare naivă. Complexitate ?



Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.

Alternative:

- Implementare naivă. Complexitate: $O(n^3)$



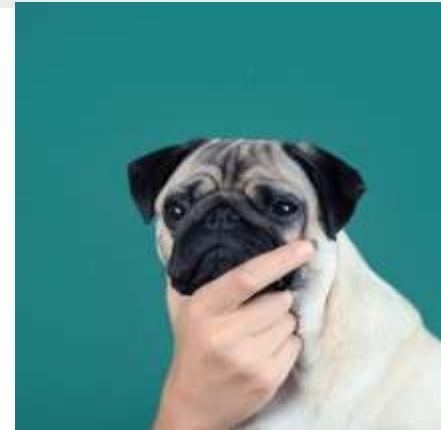
Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.

Alternative:

- Implementare naivă. Complexitate: $O(n^3)$
- Strassen (1969). $O(n^{\log 7}) = O(n^{2.81})$



Algoritmi Monte Carlo

Matrix Multiplication:

Fie A, B - două matrici pătratice de dimensiune $n \times n$. Dorim să efectuăm calculul $A \times B$.

Alternative:

- Implementare naivă. Complexitate: $O(n^3)$
- Strassen (1969). $O(n^{\log 7}) = O(n^{2.81})$
- Coppersmith-Winograd (1990). $O(n^{2.376})$



Algoritmi Monte Carlo

Matrix Multiplication Check

Fie A, B, C - trei matrici pătratice de dimensiune $n \times n$. Dorim să verificăm dacă $A \cdot B = C$



Algoritmi Monte Carlo

Matrix Multiplication Check

Fie A, B, C - trei matrici pătratice de dimensiune $n \times n$. Dorim să verificăm dacă $A \cdot B = C$

Se poate mai bine decât "calea directă"?



Algoritmi Monte Carlo

Matrix Multiplication Check

Fie A, B, C - trei matrici pătratice de dimensiune $n \times n$. Dorim să verificăm dacă $A \cdot B = C$

Se poate mai bine decât "calea directă"?

DA!



Monte Carlo: Frievald's Algorithm

Algoritm probabilist cu următoarele proprietăți:



Fie A, B, C - matricile din problemă.

- Dacă $AxB=C$, atunci algoritmul va returna întotdeauna "DA"
- Dacă $AxB \neq C$, atunci algoritmul va returna "NU" cu o probabilitate $\geq 1/2$

Monte Carlo: Frievald's Algorithm

Problemă: A,B,C - 3 matrici pătrate de dimensiune $n \times n$; Trebuie să verificăm dacă $A \times B = C$.



Monte Carlo: Frievald's Algorithm

Problemă: A,B,C - 3 matrici pătrate de dimensiune $n \times n$; Trebuie să verificăm dacă $AxB=C$.



Soluție:

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

Monte Carlo: Frievald's Algorithm

Soluție:

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

Complexitate?



Monte Carlo: Frievald's Algorithm

Soluție:

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

Complexitate: $O(n^2)$



Monte Carlo: Frievald's Algorithm

Soluție:

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

Complexitate: $O(n^2)$

Observatie: Dacă $AxB \neq C$, atunci $\Pr[Ax(Br) \neq Cr] \geq 1/2$

Justificare Pt simplitate vom presupune ca matricile sunt binare (doar elemente de 0 si 1)



Algoritmi Las Vegas

Quicksort. (C.A.R. Hoare, Moscova, 1959)



Algoritmi Las Vegas

Quicksort. (C.A.R. Hoare, Moscova, 1959)

Algoritm Bazat pe strategia Divide-et-Impera

Primește ca input un sir A de elemente comparabile, returnează sirul A sortat.

Sortează oarecum asemănător ca sortarea prin inserție: la fiecare pas se fixează un element pe poziția sa.



Algoritmi Las Vegas

Quicksort. (C.A.R. Hoare, Moscova, 1959)

Pașii:

- Divide: se alege un element x din sirul A pe post de pivot. Se partitioanează A în L (elementele $< x$), G (elementele $> x$) și E (elementele $= x$).
- Conquer: aplicăm recursiv sortarea pe sirurile L, respectiv G
- Combinare: ...



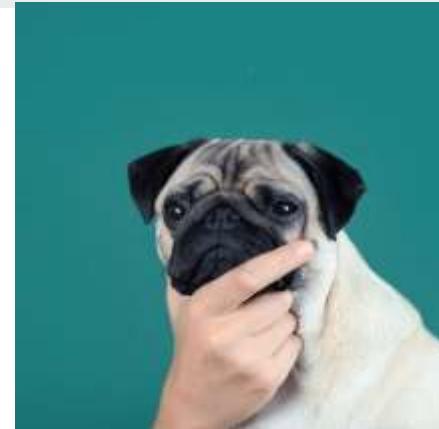
Algoritmi Las Vegas

Basic Quicksort.

1. Alegem pivotul x ca fiind fie $A[1]$, fie $A[n]$
2. În mod repetat eliminăm fiecare element y din A
 - a. inserăm y fie în L , G , sau E , în funcție de relația față de x

Fiecare inserție și ștergere durează $O(1)$

Partiționarea durează $O(n)$



Algoritmi Las Vegas

Basic Quicksort.

1. Alegem pivotul x ca fiind fie $A[1]$, fie $A[n]$
2. În mod repetat eliminăm fiecare element y din A
 - a. inserăm y fie în L , G , sau E , în funcție de relația față de x

Fiecare inserție și ștergere durează $O(1)$

Partiționarea durează $O(n)$

detalii în [CLRS](#) pag 171; Analiza algoritmului:Justificare - $O(n^2)$



Algoritmi Las Vegas

Quicksort.

Q: Cum ne asigurăm ca găsim un pivot bun?



Algoritmi Las Vegas

Quicksort.

Q: Cum să asigurăm ca găsim un pivot bun?

A: Găsirea medianei!

Q: Timp?



Algoritmi Las Vegas

Quicksort.

Q: Cum să asigurăm ca găsim un pivot bun?

A: Găsirea medianei!

Q: Timp?

A: Găsirea medianei se face în timp asimptotic liniar!



Algoritmi Las Vegas

Quicksort: Median selected as Pivot

- Ne asigură faptul că L și G sunt mereu echilibrate ca mărime
- Analiză complexitate: -prima $\Theta(n)$ este din cauza selectiei medianei, iar a doua pentru pasul de partiție.
- Avem: $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) + \theta(n)$
$$T(n) = \theta(n \cdot \log_2 n)$$



Algoritmi Las Vegas

Quicksort: Median selected as Pivot

- Ne asigură faptul că L și G sunt mereu echilibrate ca mărime
- Analiză complexitate: -prima $\Theta(n)$ este din cauza selectiei medianei, iar a doua pentru pasul de partiție.
- Avem: $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) + \theta(n)$
$$T(n) = \theta(n \cdot \log_2 n)$$



În practică acest algoritm perfomează mai prost decât varianta Basic.

Algoritmi Las Vegas

Randomized Quicksort:

- la fiecare pas al recursiei, pivotul este ales aleator.
- Este echivalent cu varianta Basic.
- Detalii în [CLRS](#) pag 181-184



Algoritmi Las Vegas

Paranoid Quicksort:

1. Repetă:
 - a. Alegem un pivot x aleator din A
 - b. Partitionam A în L, G, E , în funcție de x
2. Până când partițiile rezultate sunt de forma:
 - a. $|L| \leq 3/4|A|$ și $|G| \leq 3/4|A|$
3. Apelăm recursiv algoritmul pe L și G

Analiza algoritmului: [Justificare](#)





[Course notes in English](#)

Algoritmi Avansați 2023

C-13

Randomized Data Structures: Skip Lists; Bloom Filters

Lect. Dr. Ștefan Popescu

Email: stefan.popescu@fmi.unibuc.ro

Grup Teams:



Randomized Data Structures

- O privire pe scurt, “din avion” asupra Skip lists.



Randomized Data Structures

- O privire pe scurt, “din avion” asupra Skip lists.

Introduse in 1989 de către W. Purgh, sunt structuri dinamice bazate pe factor aleator (randomized)



Randomized Data Structures

- O privire pe scurt, “din avion” asupra Skip lists.

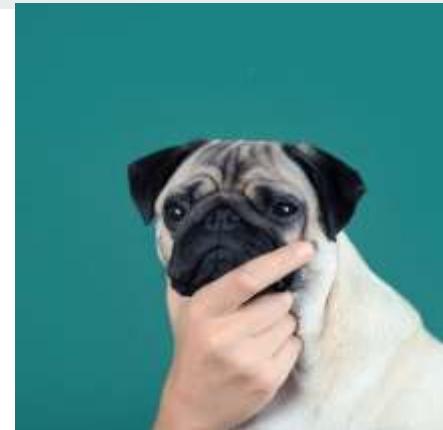
Introduse în 1989 de către W. Pugh, sunt structuri dinamice bazate pe factor aleator (randomized)

Skip lists are a probabilistic data structure that seem likely to supplant balanced trees as the implementation method of choice for many applications. Skip list algorithms have the same asymptotic expected time bounds as balanced trees and are simpler, faster and use less space.

— William Pugh, Concurrent Maintenance of Skip Lists (1989)



Listele “standard”



Listele “standard”



Q: Complexitatea cautarii unui element intr-o lista sortata?



Listele “standard”



Q: Complexitatea cautarii unui element intr-o lista sortata?

A: $O(n)$

Listele “standard”



Q: Complexitatea cautarii unui element intr-o lista sortata?

A: $O(n)$

Q: Suntem multumiti?

Listele “standard”



Q: Complexitatea cautarii unui element intr-o lista sortata?

A: $O(n)$

Q: Suntem multumiti?

A: NU!



Listele “standard”



Q: Complexitatea cautării unui element într-o listă sortată?

A: $O(n)$

Q: Complexitate întă?

Listele “standard”



Q: Complexitatea cautării unui element într-o listă sortată?

A: $O(n)$

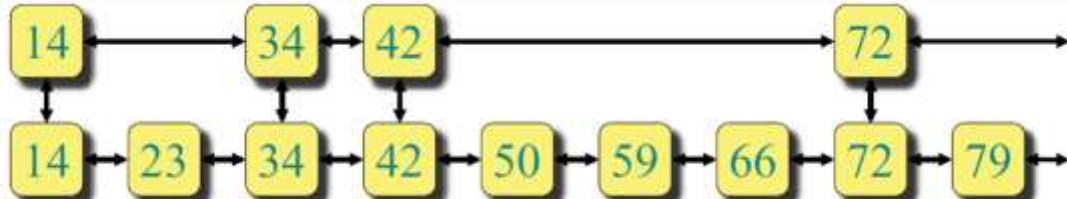
Q: Complexitate întă?

A: $O(\log n)$

Skip Lists



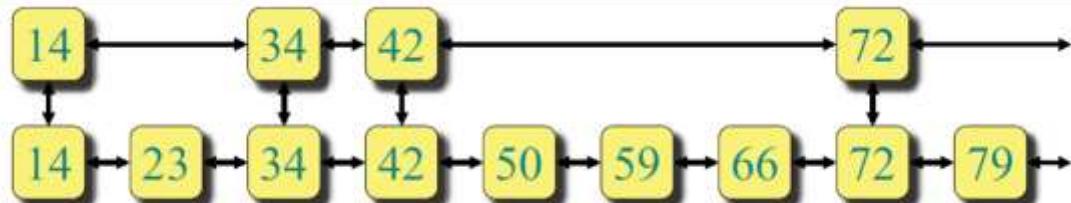
Pentru o cautare mai eficienta, vom retine 2 liste, dupa modelul urmator:



Skip Lists

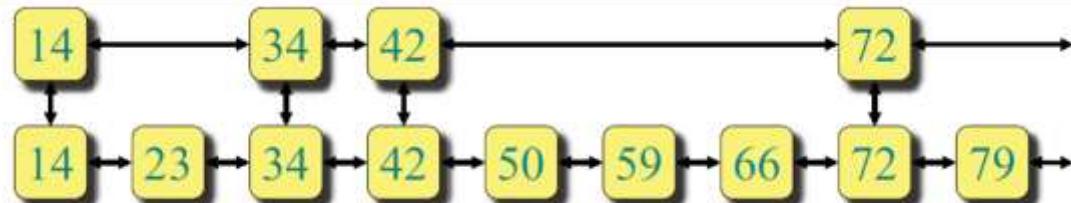
SEARCH(x):

- Walk right in top linked list (L1) until going right would go too far
- Walk down to bottom linked list (L2)
- Walk right in L2 until element found (or not)



Skip Lists

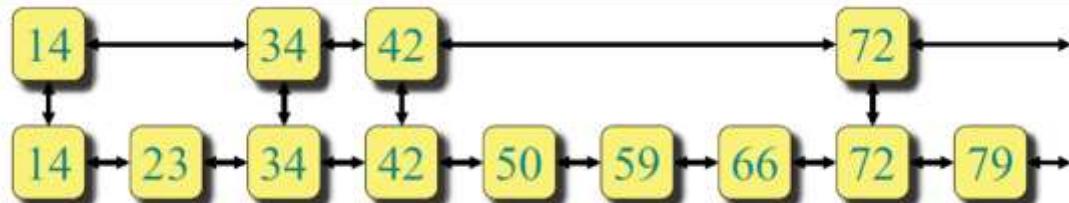
Q: Cum ar trebui distribuite nodurile din L1 (nivelul de mai sus) astfel incat cautarea sa fie cat mai eficienta?



Skip Lists

Q: Cum ar trebui distribuite nodurile din L1 (nivelul de mai sus) astfel incat cautarea sa fie cat mai eficienta?

A: Uniform distribuit!

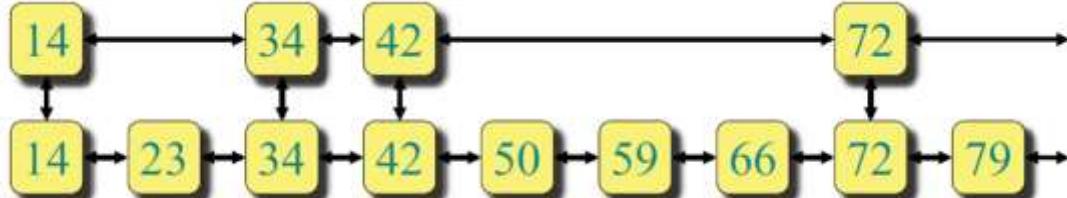


Skip Lists

Q: Cum ar trebui distribuite nodurile din L1 (nivelul de mai sus) astfel incat cautarea sa fie cat mai eficienta?

A: Uniform distribuit!

Q: cate noduri ar trebui sa existe in L1?



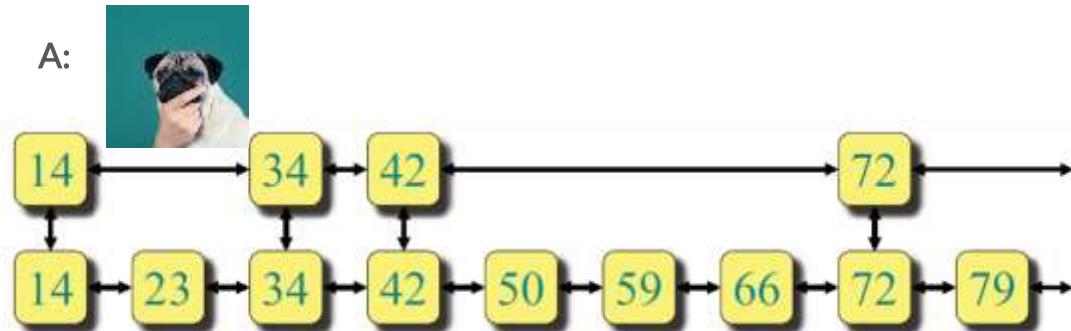
Skip Lists

Q: Cum ar trebui distribuite nodurile din L1 (nivelul de mai sus) astfel incat cautarea sa fie cat mai eficienta?

A: Uniform distribuit!

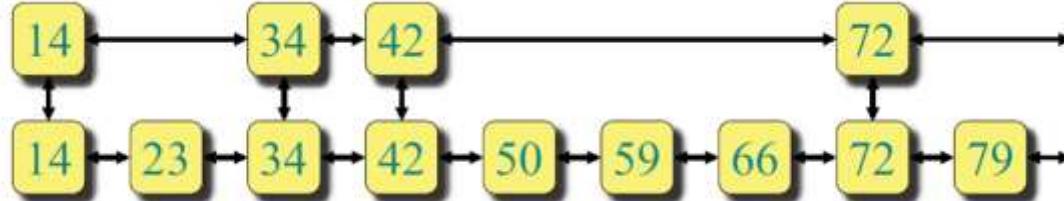
Q: cate noduri ar trebui sa existe in L1?

A:



Skip Lists: Numarul de elemente per nivel

Costul unei căutări în listă este aprox $|L1| + \frac{|L2|}{|L1|}$



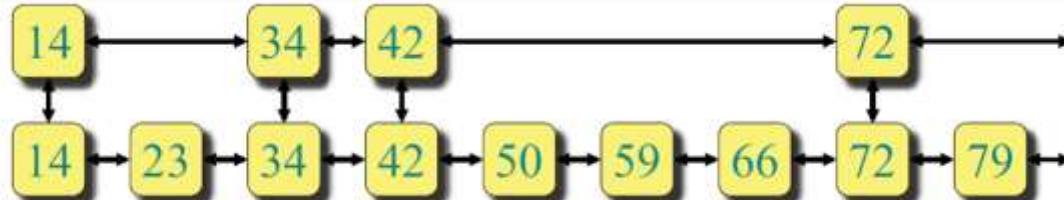
Skip Lists: Numarul de elemente per nivel

Costul unei căutări în listă este aprox $|L1| + \frac{|L2|}{|L1|}$



Relatia de mai sus este minimizata atunci cand $|L1|^2 = |L2| = n$; deci

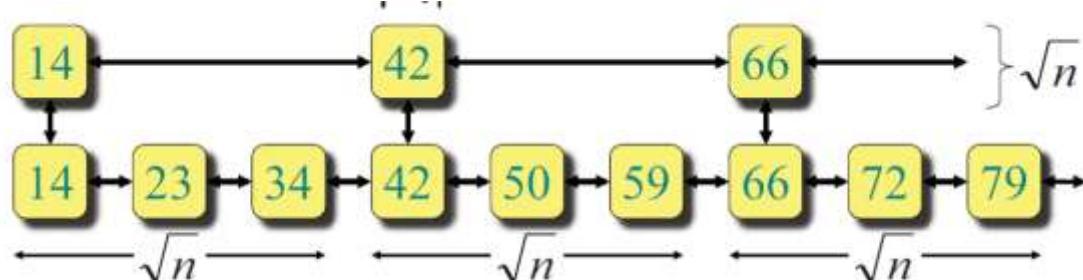
$$|L1| = \sqrt{n}$$



Skip Lists: Numarul de elemente per nivel

$|L1| = \sqrt{n}$; $|L2| = n$; Avem costul total de cautare pe 2 nivele:

$$|L1| + \frac{|L2|}{|L1|} = 2\sqrt{n}$$



Skip Lists: Numarul de elemente per nivel

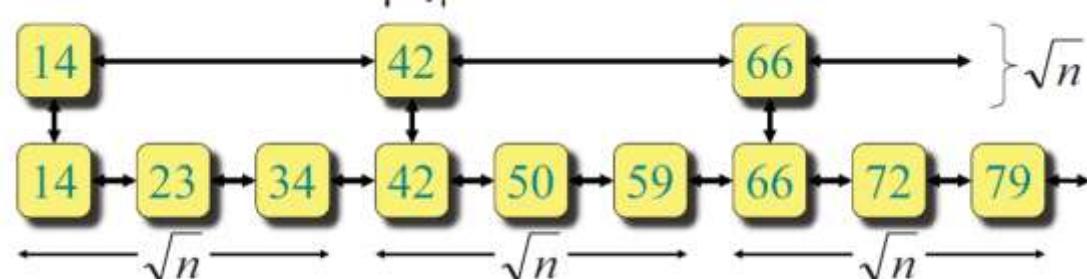
$|L1| = \sqrt{n}$; $|L2| = n$; Avem costul total de cautare pe 2 nivele:

$$|L1| + \frac{|L2|}{|L1|} = 2\sqrt{n}$$

Dar pentru 3 nivele?

Dar 4 nivele?

Dar k nivele?



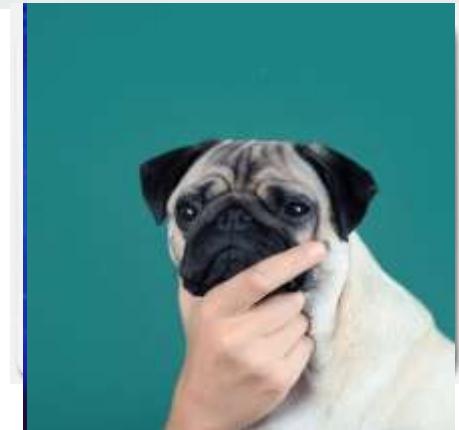
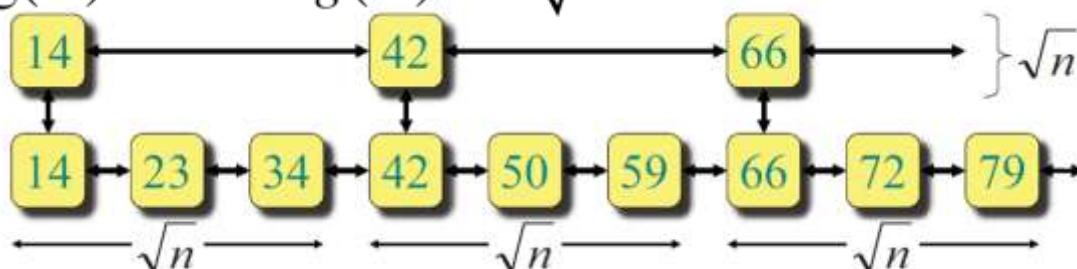
Skip Lists: Numarul de elemente per nivel

2 nivele: $2\sqrt{n}$

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

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

$\lg(k)$ nivele: $\lg(\lg(k))\sqrt{\lg(k)}\sqrt{n}$



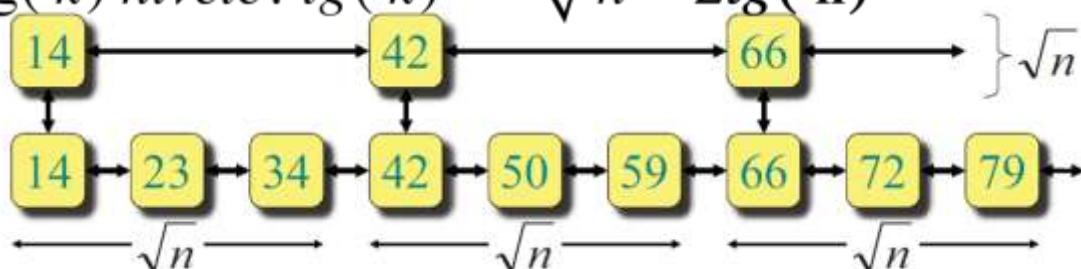
Skip Lists: Numarul de elemente per nivel

2 nivele: $2\sqrt{n}$

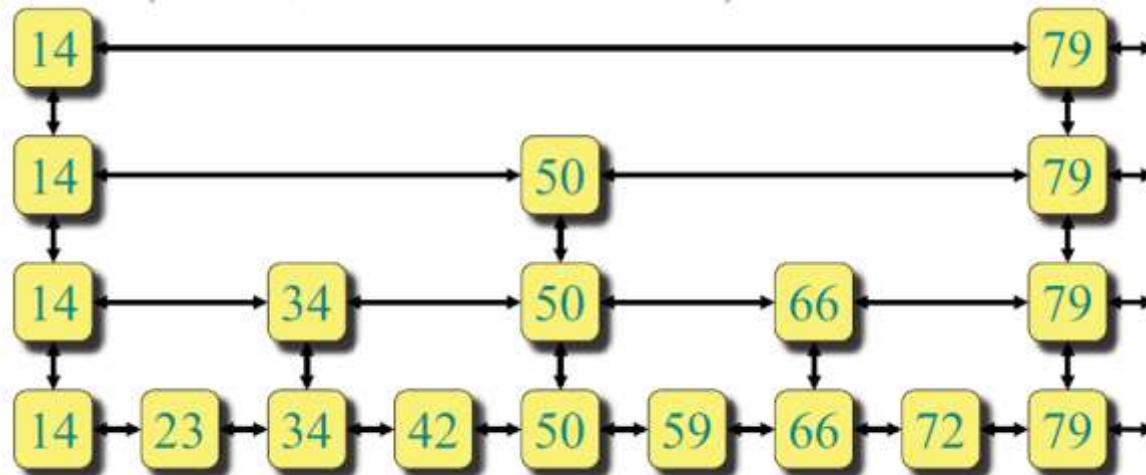
3 nivele: $3\sqrt[3]{n}$

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

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

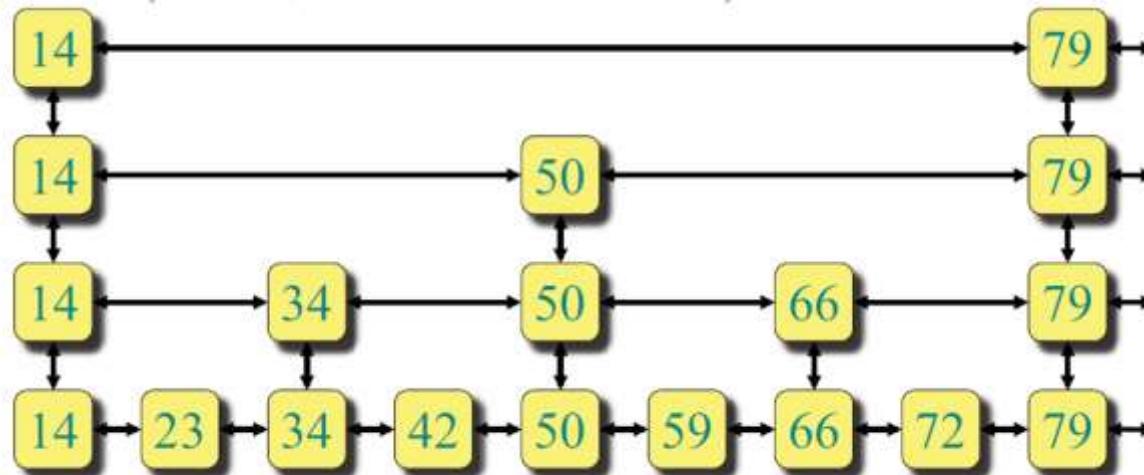


Skip Lists: Numarul de elemente per nivel



Cu ce seamănă oare?

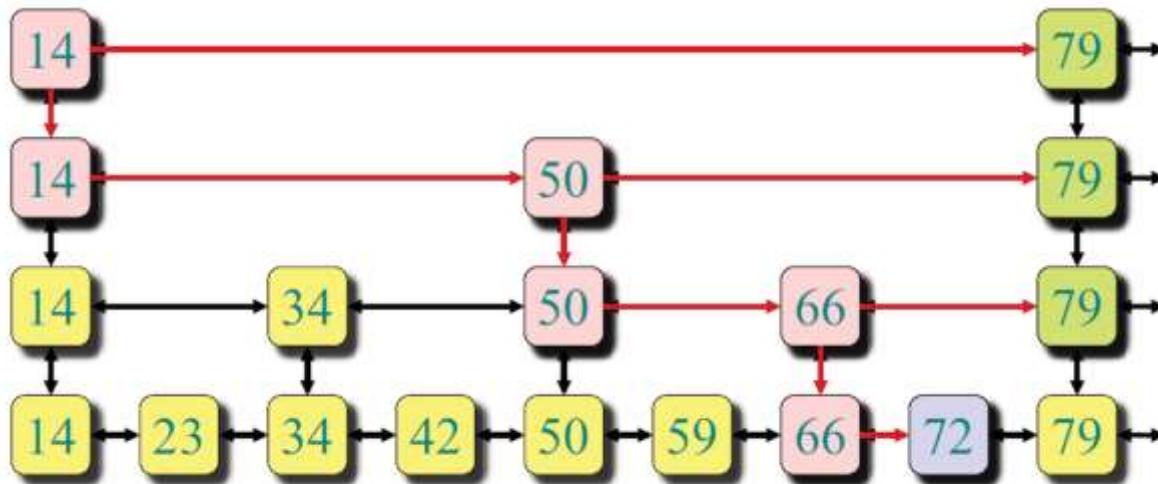
Skip Lists: Numarul de elemente per nivel



Cu ce seamana oare?

Un arbore!

Skip Lists: Search (X)



Search(72)

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



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 intotdeauna toate elementele

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 intotdeauna toate elementele

Q: Pe cate alte nivele inserez X?

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

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 intotdeauna toate elementele

Q: Pe cate alte nivele inserez X?

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

Consecinta: $\frac{1}{2}$ dintre elemente vor fi doar pe nivelul 0. $\frac{1}{4}$ dintre elemente vor fi doar pe nivelele 0 si 1. $\frac{1}{8}$ vor fi doar pe nivelele 0, 1 si 2, etc...

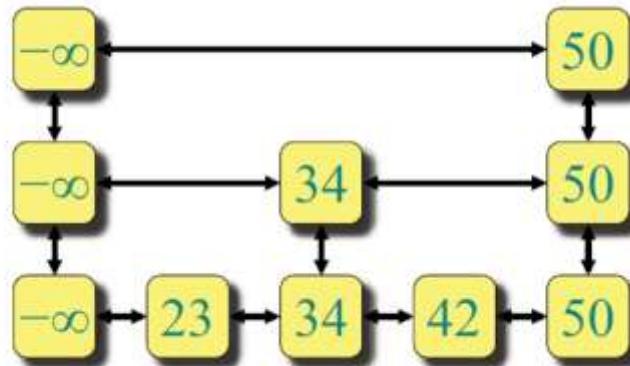
Skip Lists: Exercitiu

EXERCISE: Try building a skip list from scratch by repeated insertion using a real coin



Small change:

- Add special $-\infty$ value to *every* list
 \Rightarrow can search with the same algorithm



Skip Lists: Delete (x)

Se cauta elementul x in lista (se gaseste pe cel mai de sus nivel).

Se sterge elemntul de pe toate nivelele!



Skip Lists: How good are they?



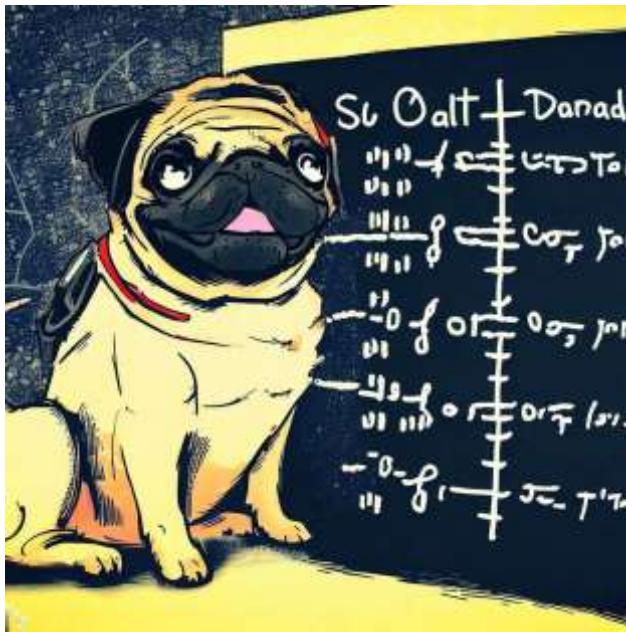
[Whiteboard S25](#)

[Mit Course on Skip Lists](#)

[Lecture Notes](#)

[Lecture Slides](#)

Bonus: Bloom Filters



Algoritmi avansați

C7 - Probleme de localizare

Mihai-Sorin Stupariu

Sem. al II-lea, 2022-2023

Căutare ortogonală – motivație

Exemplu.

Baza de date a unei bănci: informații numerice referitoare la clienți: data nașterii, număr de copii, venitul lunar, valoarea depozitelor, valoarea ratelor de plată, valoarea comisioanelor plătite anual, etc. → stocarea se realizează folosind puncte dintr-un spațiu numeric d -dimensional \mathbb{R}^d .

Căutare ortogonală – motivație

Exemplu.

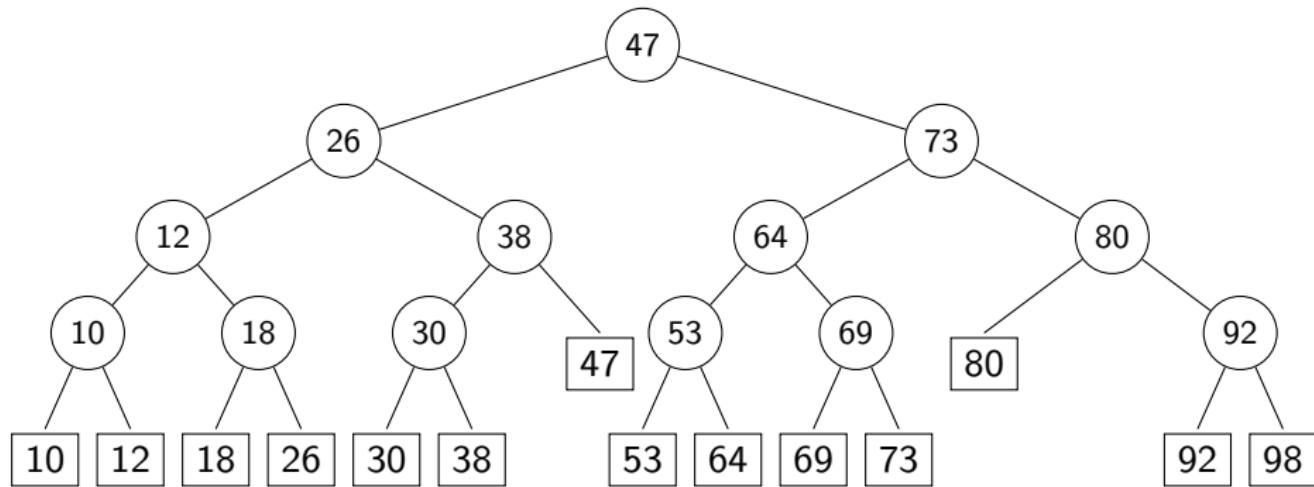
Baza de date a unei bănci: informații numerice referitoare la clienți: data nașterii, număr de copii, venitul lunar, valoarea depozitelor, valoarea ratelor de plată, valoarea comisioanelor plătite anual, etc. → stocarea se realizează folosind puncte dintr-un spațiu numeric d -dimensional \mathbb{R}^d .

A identifica un “grup-țintă” de clienți (de exemplu pentru lansarea unui produs), având anumite caracteristici – e.g. vârstă între 30-40 ani, 2-4 copii, un venit lunar între 3000-5000 lei, etc. revine la efectuarea căutări prin care să fie determinate punctele situate într-un “paralelipiped” d -dimensional.

Căutare 1-dimensională: formularea problemei

Cadru. Fie $M = \{a_1, a_2, \dots, a_n\}$ o mulțime de numere reale. Fie $I = [x, x'] \subset \mathbb{R}$ un interval real. Se dorește determinarea elementelor lui M situate în intervalul I .

Structura de date utilizată: Arbore binar de căutare echilibrat.

Exemplu de arbore \mathcal{T} 

Rezultatul principal - căutare 1D

Teoremă. Fie M o mulțime de n puncte din \mathbb{R} . Mulțimea M poate fi memorată într-un arbore binar de căutare echilibrat, folosind $O(n)$ memorie și cu timp de construcție $O(n \log n)$. Determinarea unor puncte dintr-un interval I poate fi realizată cu complexitate-timp $O(k + \log n)$, unde k este numărul de puncte din $M \cap I$.

Rezultatul principal - căutare 2D

Teoremă. Fie M o mulțime de n puncte din planul \mathbb{R}^2 . Un arbore de intervale (range tree) pentru M necesită $O(n \log n)$ memorie și poate fi construit în timp $O(n \log n)$. Determinarea unor puncte dintr-un dreptunghi D poate fi realizată cu complexitate-timp $O(k + \log^2 n)$, unde k este numărul de puncte din $M \cap D$.

Localizarea punctelor – problematizare

- ▶ Căutare cu Google Maps

Localizarea punctelor – problematizare

- ▶ Căutare cu Google Maps
- ▶ *Interrogare pentru localizarea unui punct:* dată o hartă și un punct p , indicat prin cordonatele sale, să se determine regiunea hărții în care este situat p .

Localizarea punctelor – problematizare

- ▶ Căutare cu Google Maps
- ▶ *Interrogare pentru localizarea unui punct:* dată o hartă și un punct p , indicat prin cordonatele sale, să se determine regiunea hărții în care este situat p .
- ▶ *Harta:* subdiviziune planară, formată din vârfuri, (semi)muchii, fețe.

Localizarea punctelor – problematizare

- ▶ Căutare cu Google Maps
- ▶ *Interrogare pentru localizarea unui punct:* dată o hartă și un punct p , indicat prin cordonatele sale, să se determine regiunea hărții în care este situat p .
- ▶ *Harta:* subdiviziune planară, formată din vârfuri, (semi)muchii, fețe.
- ▶ Necessități: pre-procesare a informației; interogare rapidă.

Formalizare

- ▶ Fie \mathcal{S} o subdiviziune planară cu n muchii. *Problema localizării unui punct* revine la

Formalizare

- ▶ Fie \mathcal{S} o subdiviziune planară cu n muchii. *Problema localizării unui punct* revine la
 - ▶ a reține informațiile referitoare la \mathcal{S} pentru a putea răspunde la interogări de tipul:

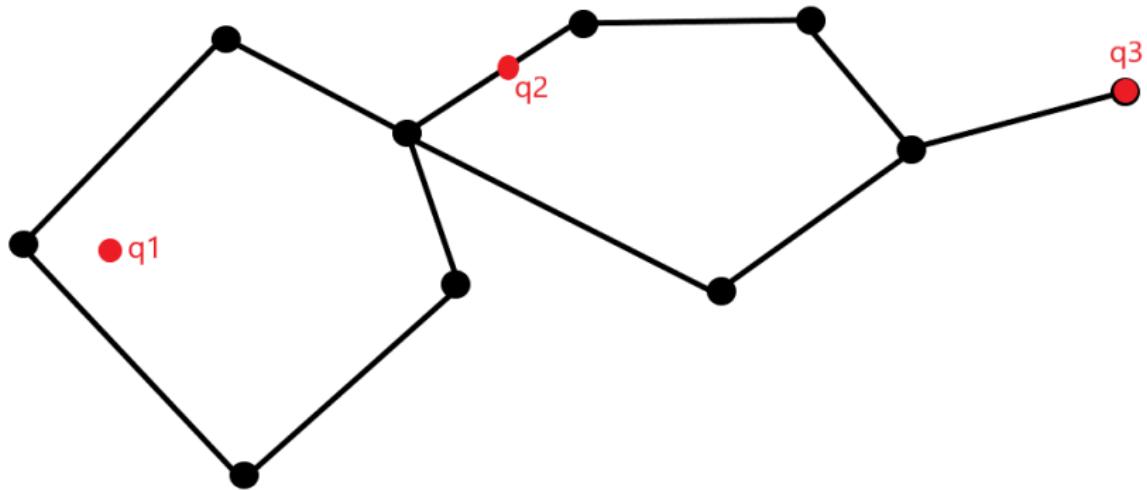
Formalizare

- ▶ Fie \mathcal{S} o subdiviziune planară cu n muchii. *Problema localizării unui punct* revine la
 - ▶ a reține informațiile referitoare la \mathcal{S} pentru a putea răspunde la interogări de tipul:
 - ▶ dat un punct p , se raportează fața f care îl conține pe p ; în cazul în care p este situat pe un segment sau coincide cu un vârf, este precizat acest lucru.

Formalizare

- ▶ Fie \mathcal{S} o subdiviziune planară cu n muchii. *Problema localizării unui punct* revine la
 - ▶ a reține informațiile referitoare la \mathcal{S} pentru a putea răspunde la interogări de tipul:
 - ▶ dat un punct p , se raportează fața f care îl conține pe p ; în cazul în care p este situat pe un segment sau coincide cu un vârf, este precizat acest lucru.
- ▶ Lucrul cu coordonate: folosirea relației de ordine!

Intuiție



Intuiție - concluzie

- Subdivizare a planului în fâșii (benzi) verticale (cf. “*slabs*”)

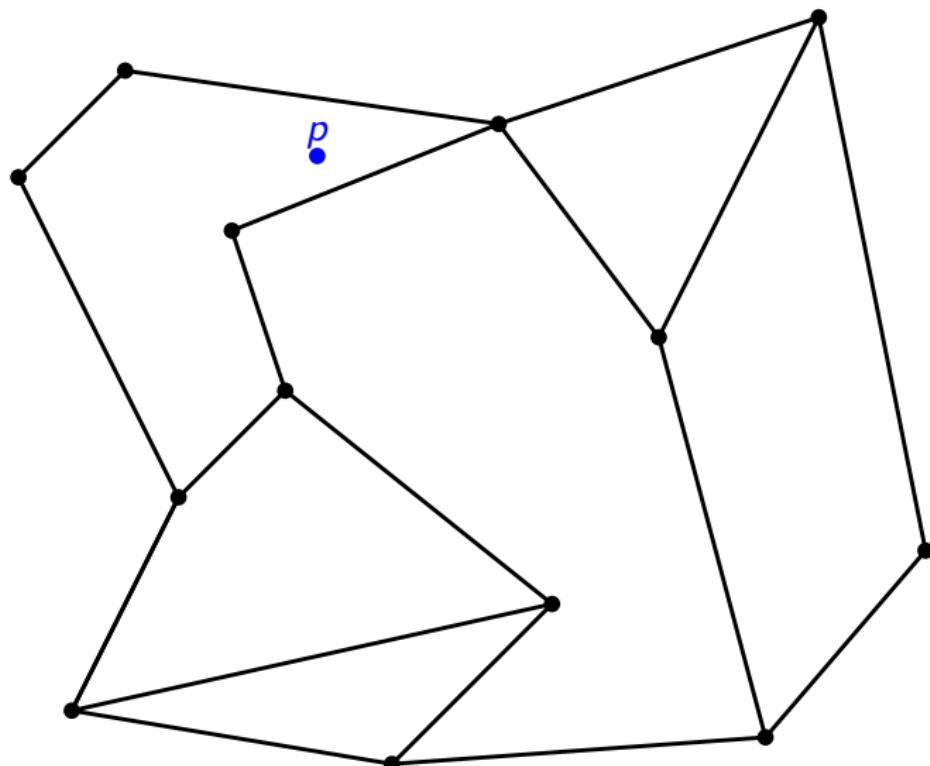
Intuiție - concluzie

- ▶ Subdivizare a planului în fâșii (benzi) verticale (cf. “*slabs*”)
 - ▶ **căutare după abscisă** - pentru identificarea fâșiei verticale (timp de căutare $O(\log n)$);

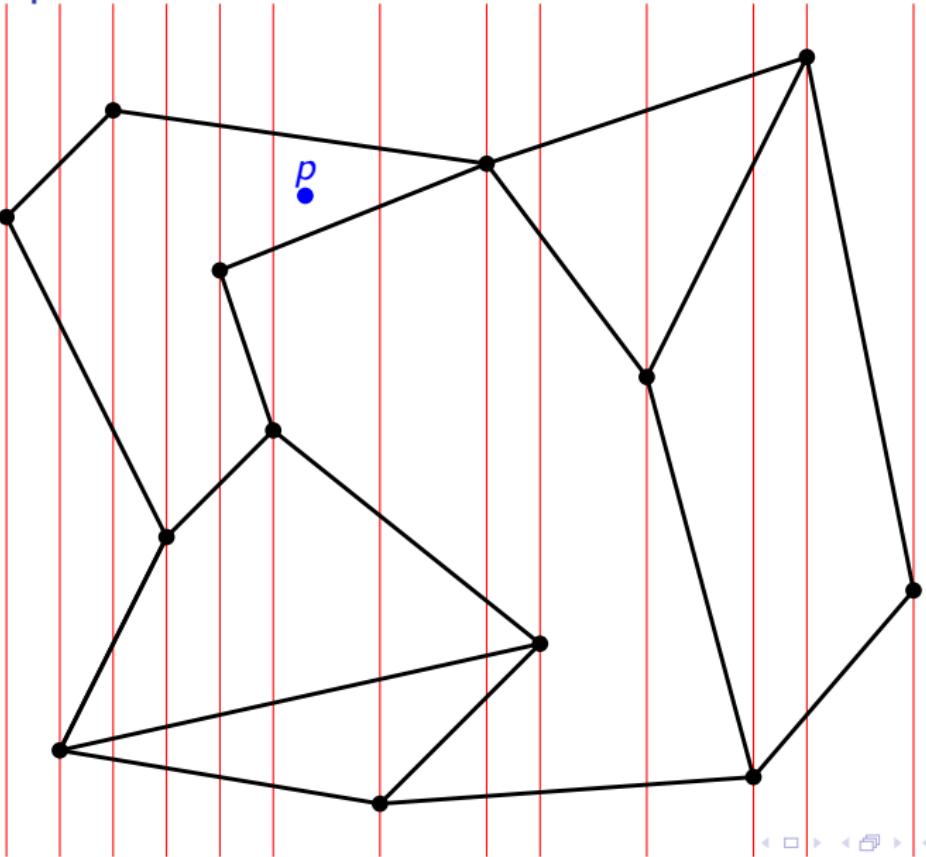
Intuiție - concluzie

- ▶ Subdivizare a planului în fâșii (benzi) verticale (cf. “*slabs*”)
 - ▶ **căutare după abscisă** - pentru identificarea fâșiei verticale (timp de căutare $O(\log n)$);
 - ▶ **căutare în cadrul unei fâșii** - pentru localizare în cadrul fâșiei verticale, realizată în raport cu segmente.

Exemplu

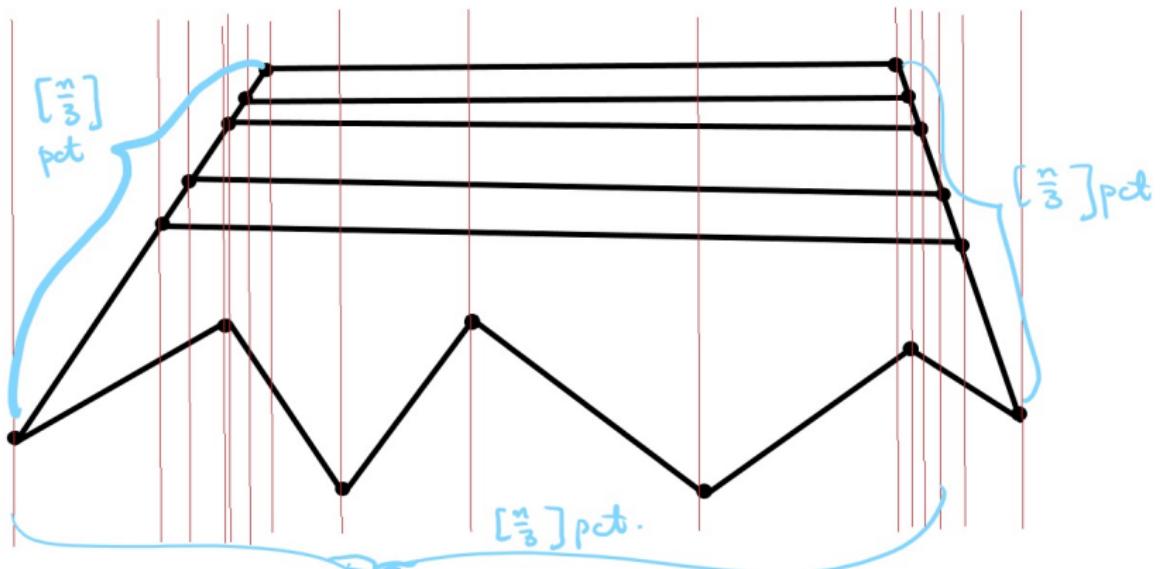


Exemplu - rafinare folosind benzi verticale

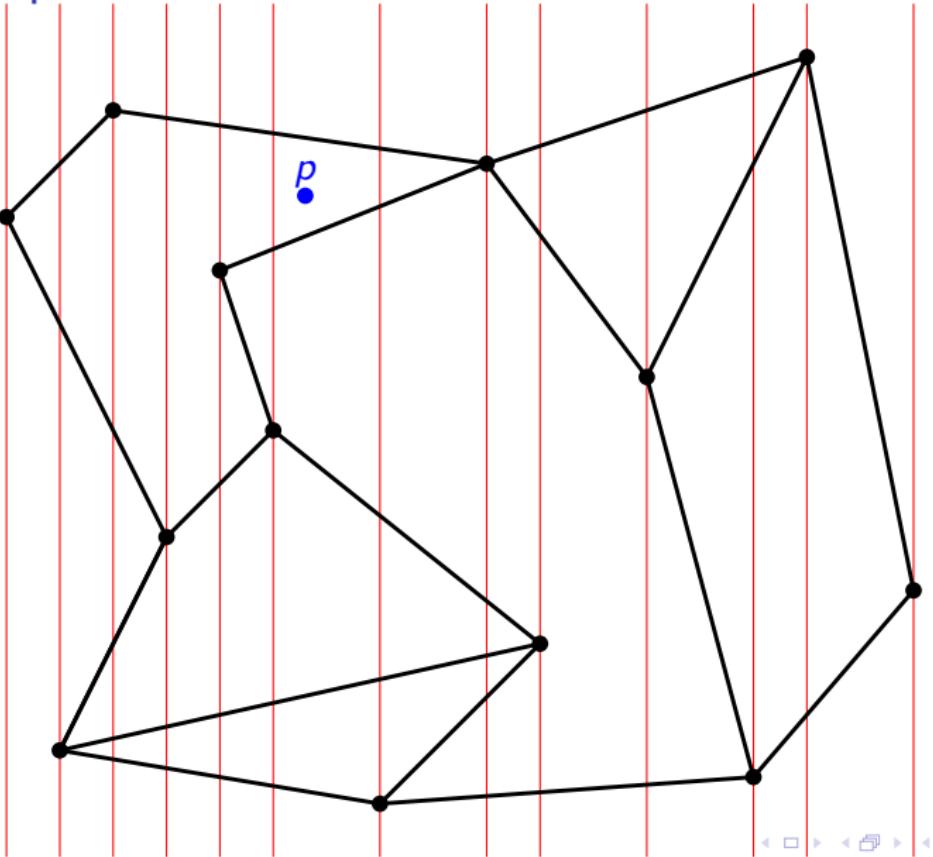


O astfel de rafinare nu este eficientă

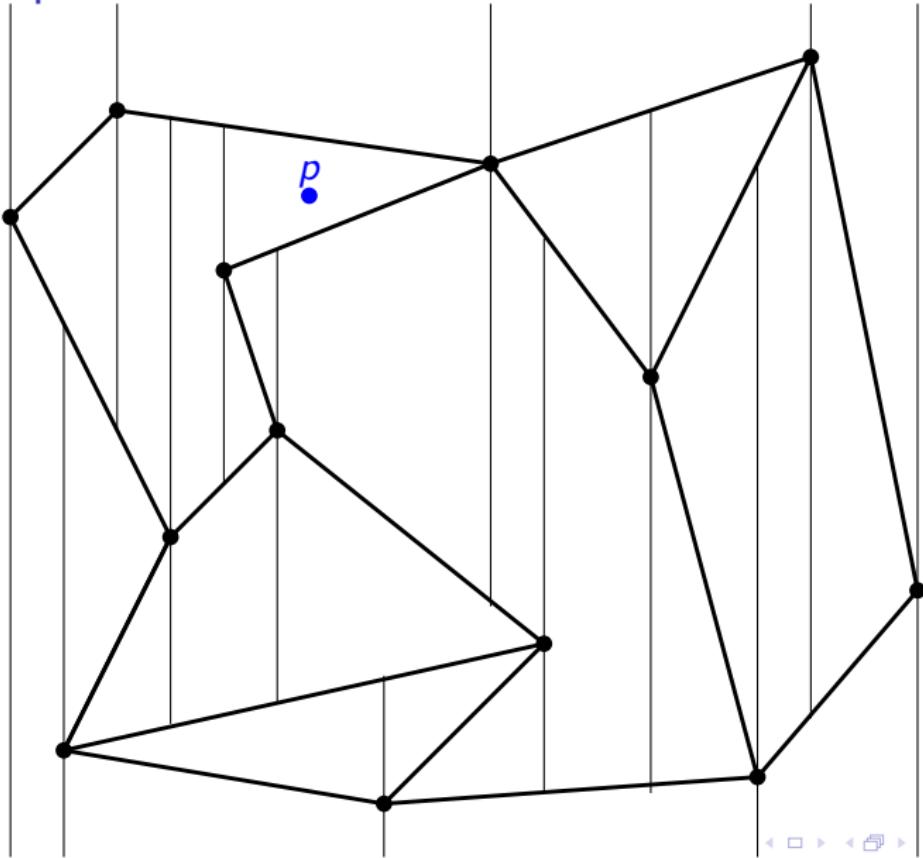
Memoria necesară poate fi uneori $O(n^2)$



Exemplu - rafinare folosind benzi verticale



Exemplu - rafinare eficientă



Simplificări și ipoteze

- ▶ Se consideră o mulțime S de n segmente astfel ca oricare două dintre ele (i) fie nu au niciun punct comun; (ii) fie au un vârf comun.

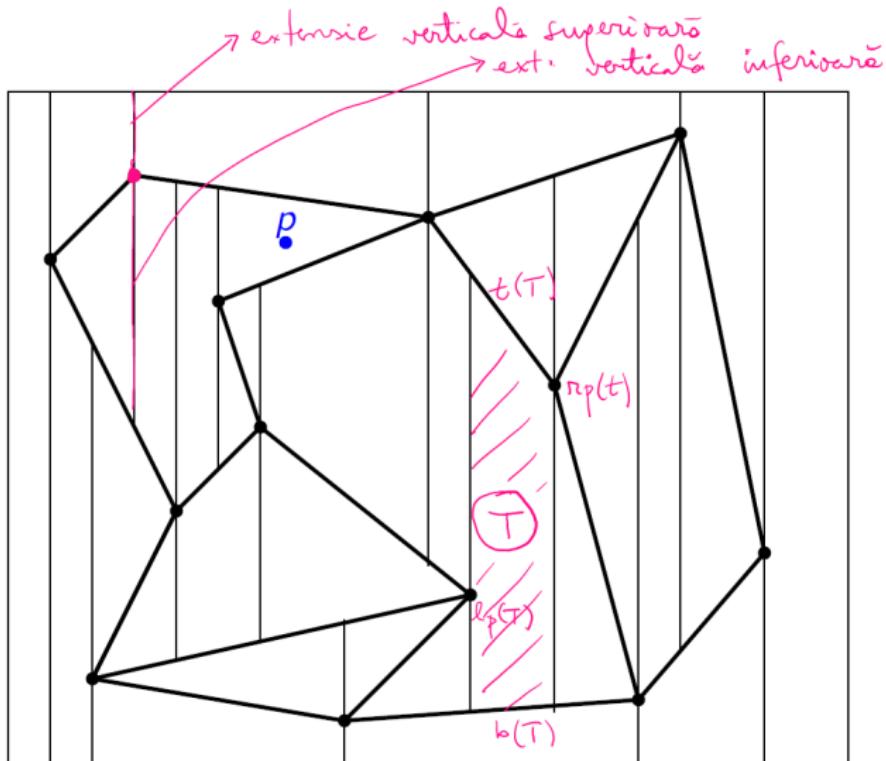
Simplificări și ipoteze

- ▶ Se consideră o mulțime S de n segmente astfel ca oricare două dintre ele (i) fie nu au niciun punct comun; (ii) fie au un vârf comun.
- ▶ *Simplificare 1:* Se consideră un dreptunghi D cu laturile paralele cu axele de coordinate care include toată subdiviziunea inițială.
- ▶ *Simplificare 2:* Se presupune că nu există două vârfuri (extremități ale segmentelor din S) distincte care au aceeași coordonată x (în particular nu există segmente verticale).

Simplificări și ipoteze

- ▶ Se consideră o mulțime S de n segmente astfel ca oricare două dintre ele (i) fie nu au niciun punct comun; (ii) fie au un vârf comun.
- ▶ *Simplificare 1:* Se consideră un dreptunghi D cu laturile paralele cu axele de coordinate care include toată subdiviziunea inițială.
- ▶ *Simplificare 2:* Se presupune că nu există două vârfuri (extremități ale segmentelor din S) distincte care au aceeași coordonată x (în particular nu există segmente verticale).
- ▶ *Concluzie:* Se consideră o mulțime de n segmente S care verifică ipotezele de mai sus: *mulțime de segmente în poziție generală*. **Harta trapezoidală / descompunere verticală / descompunere cu trapeze** (*trapezoidal map*) $\mathcal{T}(S)$ a lui S este subdiviziunea indusă de S , dreptunghiul D și de extensiile verticale inferioare și superioare (concept introdus de Seidel, 1991).

Exemplu - hartă trapezoidală, extensii verticale



Hărți trapezoidale – probleme studiate

- ▶ Descrierea obiectelor geometrice din care sunt formate – ce informații se rețin?

Hărți trapezoidale – probleme studiate

- ▶ Descrierea obiectelor geometrice din care sunt formate – ce informații se rețin?
- ▶ Aspecte legate de complexitate?

Hărți trapezoidale – probleme studiate

- ▶ Descrierea obiectelor geometrice din care sunt formate – ce informații se rețin?
- ▶ Aspecte legate de complexitate?
- ▶ Structuri de date adecvate?

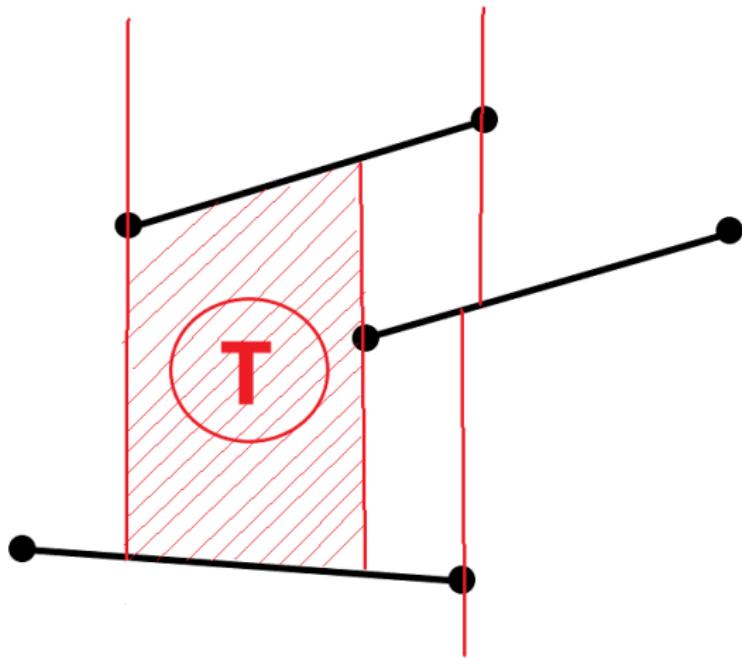
Hărți trapezoidale – probleme studiate

- ▶ Descrierea obiectelor geometrice din care sunt formate – ce informații se rețin?
- ▶ Aspecte legate de complexitate?
- ▶ Structuri de date adecvate?
- ▶ Un algoritm eficient?

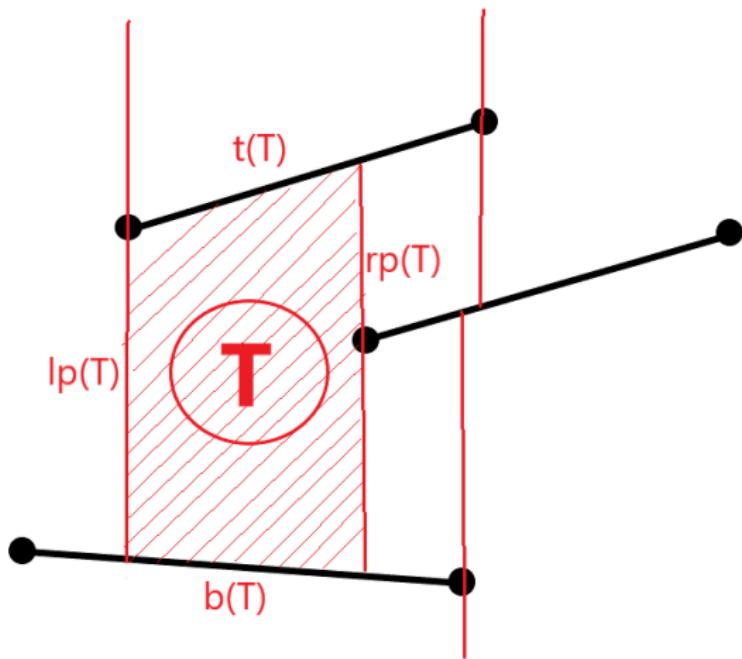
Descrierea obiectelor

- ▶ **Lema 1.** Fie S o mulțime de segmente în poziție generală. Fiecare față a unei hărți trapezoidale $\mathcal{T}(S)$ are una sau două margini verticale și exact două margini ne-verticale.
De fapt: fiecare față este un trapez, sau un dreptunghi sau un triunghi (ultimele putând fi privite drept cazuri particulare de trapeze).

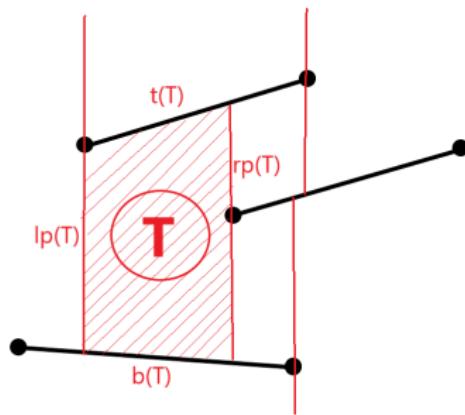
Informații geometrice sunt reținute pentru un trapez



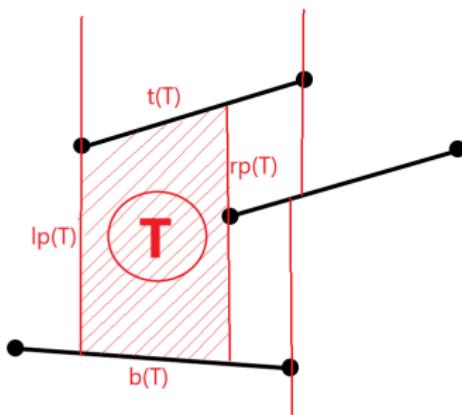
Informații geometrice sunt reținute pentru un trapez



Informații geometrice sunt reținute pentru un trapez

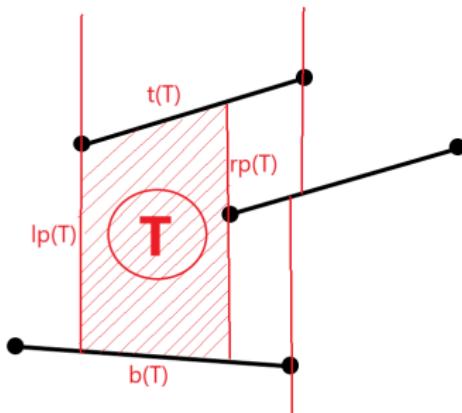


Informații geometrice sunt reținute pentru un trapez



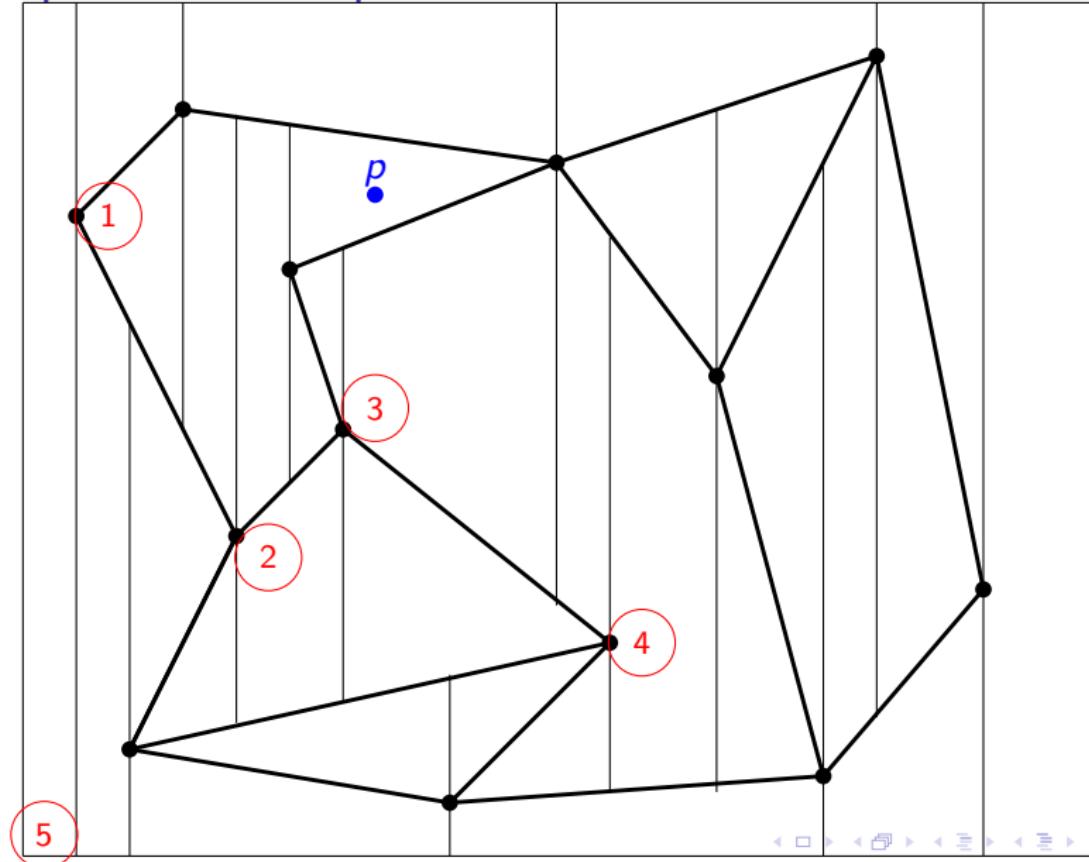
- ▶ $t(T)$, $b(T)$, $lp(T)$, $rp(T)$ determină în mod unic un trapez fixat T .
 $t(T)$, $b(T)$ sunt **segmente**, iar $lp(T)$, $rp(T)$ sunt **vârfuri**
 (extremități ale segmentelor)

Informații geometrice sunt reținute pentru un trapez



- ▶ $t(T)$, $b(T)$, $lp(T)$, $rp(T)$ determină în mod unic un trapez fixat T .
 $t(T)$, $b(T)$ sunt **segmente**, iar $lp(T)$, $rp(T)$ sunt **vârfuri** (extremități ale segmentelor)
- ▶ Există cinci cazuri posibile pentru marginea stângă lp (analog pentru marginea dreaptă rp).

Exemplu - hartă trapezoidală

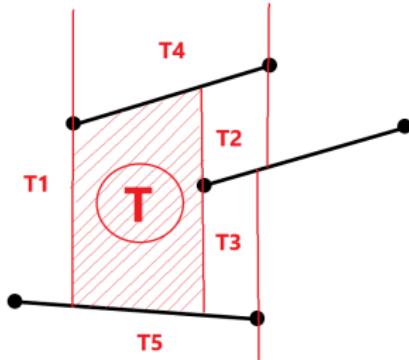


Complexitate și alte aspecte cantitative

- ▶ **Lema 2.** Fie S o mulțime de n segmente în poziție generală. Harta trapezoidală $\mathcal{T}(S)$ conține cel mult $6n + 4$ vârfuri și cel mult $3n + 1$ trapeze.

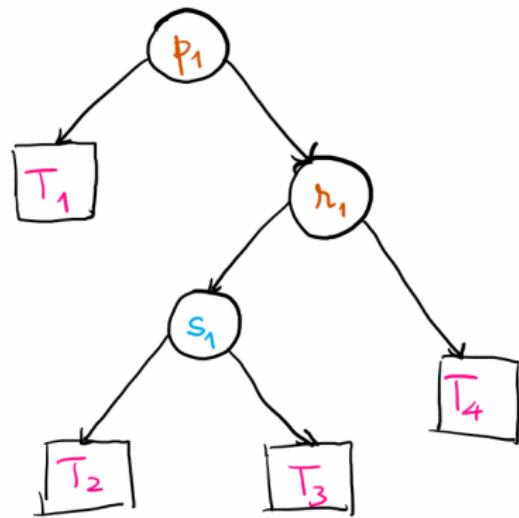
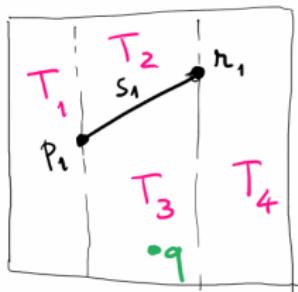
Complexitate și alte aspecte cantitative

- ▶ **Lema 2.** Fie S o mulțime de n segmente în poziție generală. Harta trapezoidală $\mathcal{T}(S)$ conține cel mult $6n + 4$ vârfuri și cel mult $3n + 1$ trapeze.
- ▶ **Lema 3.** Fie S o mulțime de n segmente în poziție generală. Fiecare trapez T este adiacent cu cel mult patru trapeze (cel mult un vecin stânga superior, cel mult un vecin stânga inferior, cel mult un vecin dreapta superior, cel mult un vecin dreapta inferior).



Trapezul T este adiacent cu T_1 , T_2 și T_3 , dar nu este adiacent cu T_4 și T_5 .

Exemplul 1 - structură de căutare asociată

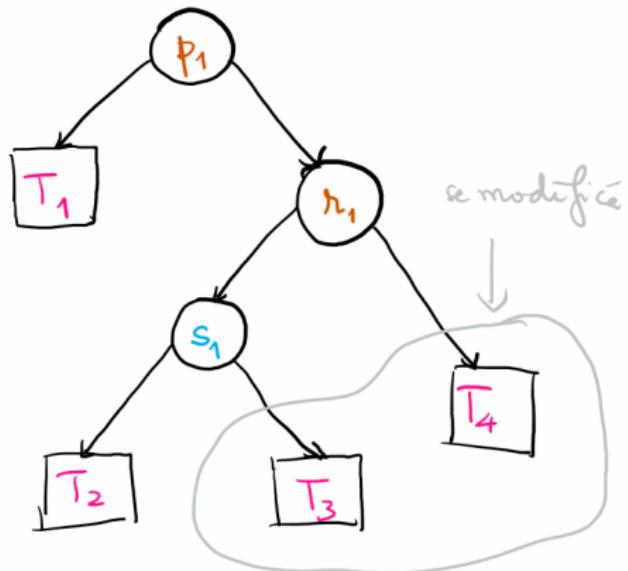
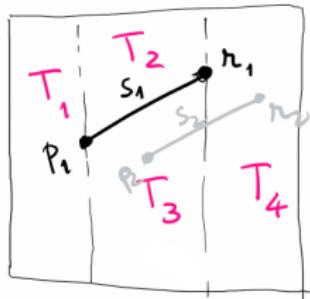


$$q \in T_3$$

Structura de căutare în cazul în care subdiviziunea planară este dată de un segment, s_1 .

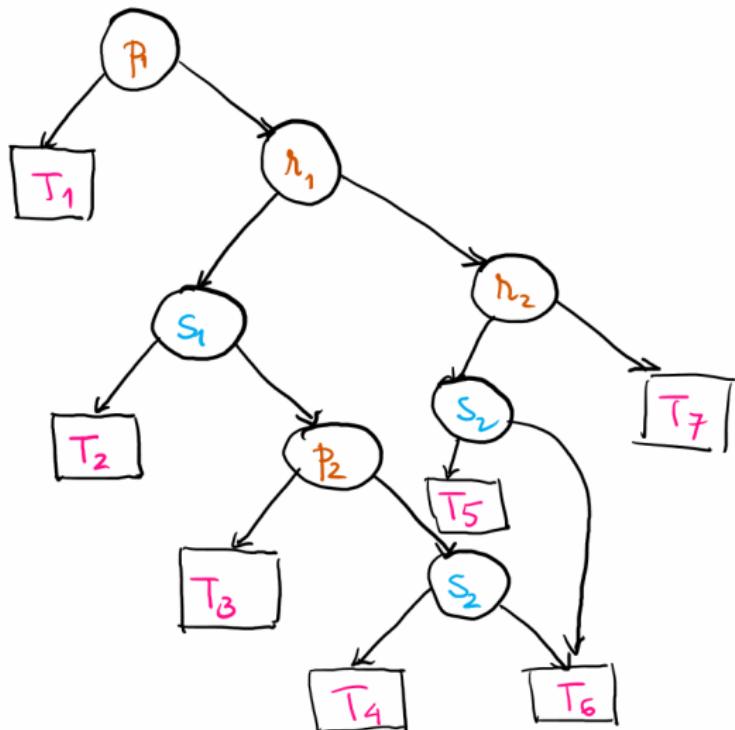
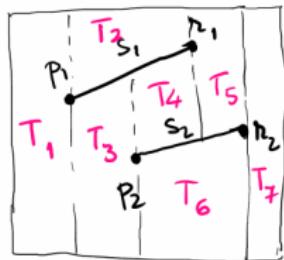
Folosind structura de căutare se poate stabili că punctul q este situat în trapezul T_3 .

Exemplul 1 - structură de căutare asociată



Structura de căutare în cazul în care subdiviziunea planară este dată de un segment, s_1 . Presupunem că mai adăgăm un segment, s_2 . Folosind structura de căutare se stabilește poziția extremităților segmentelor, urmând ca structura să fie actualizată prin înlocuirea trapezelor respective cu noduri/trapeze noi, în mod adecvat.

Exemplul 2 - structură de căutare asociată



Structura de căutare în cazul în care subdiviziunea planară este dată de două segmente, s_1 și s_2 .

Căutarea într-o hartă trapezoidală

x -nod (p)



q este la stg / dreapta
de vîrticele care trece
prin p
(comparații de abscise)

y -nod (s)



q este deasupra /
de dedesubtul lui s
(testul de orientare)

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).
- ▶ Înregistrări pentru trapeze: pointeri t , b , lp , rp și pointeri către cei (cel mult) patru vecini.

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).
- ▶ Înregistrări pentru trapeze: pointeri t , b , lp , rp și pointeri către cei (cel mult) patru vecini.
- ▶ **Structura de căutare:** \mathcal{D} este un graf orientat aciclic cu o singură rădăcină și exact o frunză pentru fiecare trapez din $\mathcal{T}(S)$.

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).
- ▶ Înregistrări pentru trapeze: pointeri t , b , lp , rp și pointeri către cei (cel mult) patru vecini.
- ▶ **Structura de căutare:** \mathcal{D} este un graf orientat aciclic cu o singură rădăcină și exact o frunză pentru fiecare trapez din $\mathcal{T}(S)$.
- ▶ **Noduri și teste asociate:**

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).
- ▶ Înregistrări pentru trapeze: pointeri t , b , lp , rp și pointeri către cei (cel mult) patru vecini.
- ▶ **Structura de căutare:** \mathcal{D} este un graf orientat aciclic cu o singură rădăcină și exact o frunză pentru fiecare trapez din $\mathcal{T}(S)$.
- ▶ **Noduri și teste asociate:**
 - ▶ x-nod, etichetat cu o extremitate a unui segment; pentru un punct p testul asociat: *este punctul p situat la stânga sau la dreapta dreptei verticale care trece prin extremitatea memorată în acest nod?*

Structura de date

- ▶ Înregistrări pentru segmentele din S și vârfuri (extremitățile segmentelor).
- ▶ Înregistrări pentru trapeze: pointeri t , b , lp , rp și pointeri către cei (cel mult) patru vecini.
- ▶ **Structura de căutare:** \mathcal{D} este un graf orientat aciclic cu o singură rădăcină și exact o frunză pentru fiecare trapez din $\mathcal{T}(S)$.
- ▶ **Noduri și teste asociate:**
 - ▶ x-nod, etichetat cu o extremitate a unui segment; pentru un punct p testul asociat: *este punctul p situat la stânga sau la dreapta dreptei verticale care trece prin extremitatea memorată în acest nod?*
 - ▶ y-nod, etichetat cu un segment; pentru un punct p testul asociat: *este punctul p situat deasupra sau dedesubtul segmentului memorat în acest nod?*

Algoritm HARTA TRAPEZOIDALA

- **Input.** O mulțime S de n segmente în poziție generală.

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
- ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .
 2. Generează o permutare s_1, s_2, \dots, s_n a segmentelor din S .

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .
 2. Generează o permutare s_1, s_2, \dots, s_n a segmentelor din S .
 3. **for** $i \leftarrow 1$ **to** n

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .
 2. Generează o permutare s_1, s_2, \dots, s_n a segmentelor din S .
 3. **for** $i \leftarrow 1$ **to** n
 4. **do** găsește mulțimea de trapeze T_0, T_1, \dots, T_k care intersectează segmentul s_i

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .
 2. Generează o permutare s_1, s_2, \dots, s_n a segmentelor din S .
 3. **for** $i \leftarrow 1$ **to** n
 4. **do** găsește mulțimea de trapeze T_0, T_1, \dots, T_k care intersectează segmentul s_i
 5. elimină T_0, \dots, T_k și le înlocuiește cu trapezele nou apărute

Algoritm HARTA TRAPEZOIDALA

- ▶ **Input.** O mulțime S de n segmente în poziție generală.
 - ▶ **Output.** Harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$, într-un dreptunghi D cu laturile paralele cu axele.
1. Determină dreptunghiuul D .
 2. Generează o permutare s_1, s_2, \dots, s_n a segmentelor din S .
 3. **for** $i \leftarrow 1$ **to** n
 4. **do** găsește mulțimea de trapeze T_0, T_1, \dots, T_k care intersectează segmentul s_i
 5. elimină T_0, \dots, T_k și le înlocuiește cu trapezele nou apărute
 6. elimină frunzele corespunzătoare din \mathcal{D} și creează noi frunze, actualizează \mathcal{D}

Rezultatul principal

- ▶ **Teoremă.** Fie S o mulțime de n segmente în poziție generală. Algoritmul HARTA TRAPEZOIDALA determină harta trapezoidală $\mathcal{T}(S)$ și o structură de căutare $\mathcal{D} = \mathcal{D}(\mathcal{T}(S))$ pentru $\mathcal{T}(S)$ în timp mediu $O(n \log n)$. Memoria medie ocupată de structura de căutare este $O(n)$ și pentru un punct arbitrar p timpul mediu de localizare este $O(\log n)$.

Cadru (simplificat)

- Context 2D



Cadru (simplificat)

- ▶ Context 2D



- ▶ **Obstacolele:** reprezentate de poligoane disjuncte P_1, P_2, \dots, P_k având n vârfuri
- ▶ **Robot:** punct \mathcal{R} care se deplasează prin translație

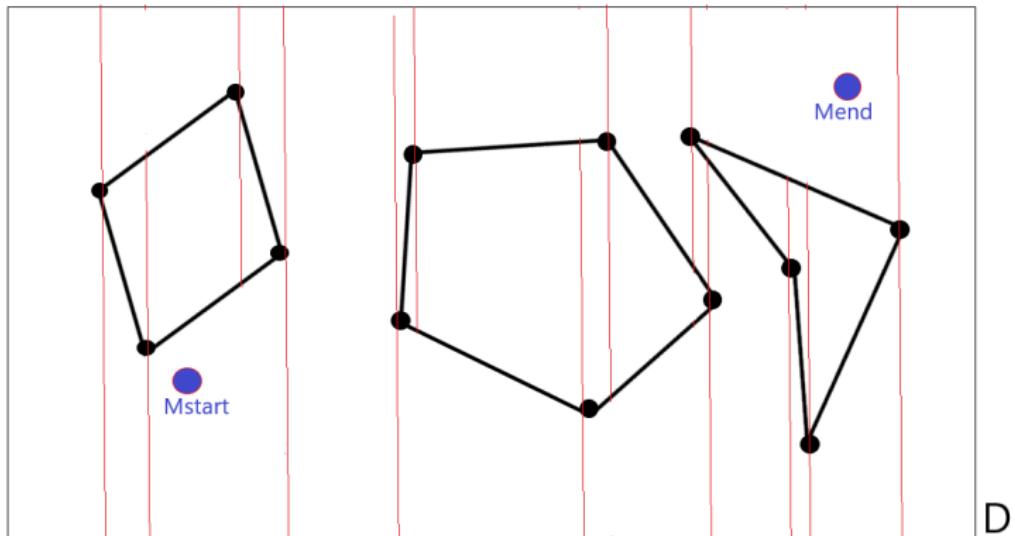
Cadru (simplificat)

- ▶ Context 2D



- ▶ **Obstacolele:** reprezentate de poligoane disjuncte P_1, P_2, \dots, P_k având n vârfuri
- ▶ **Robot:** punct \mathcal{R} care se deplasează prin translație
- ▶ **Simplificare (mărginire):** figura este inclusă într-un dreptunghi D ("bounding box").

Illustrare



Pe unde se poate mișca robotul? (spațiul liber)
Găsirea unui drum?

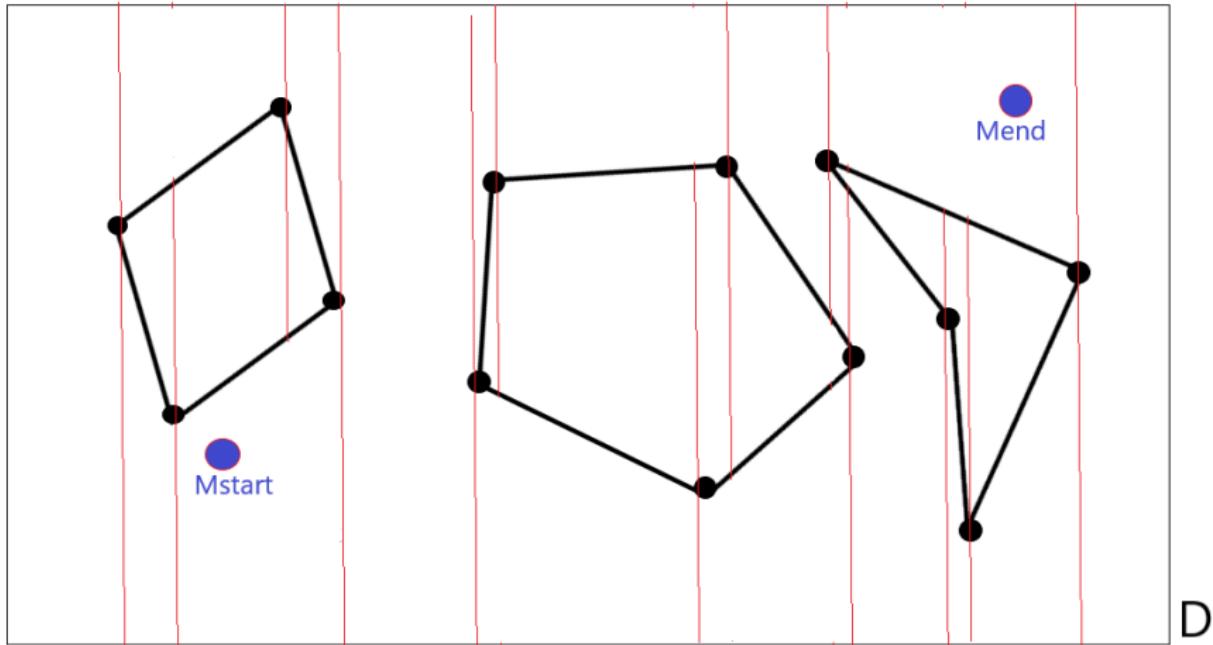
Pasul 1: determinarea spațiului liber

- ▶ Trapezele din interiorul obstacolelor au muchia “top” inclusă în frontiera superioară a unui obstacol: **aceste trapeze sunt eliminate**

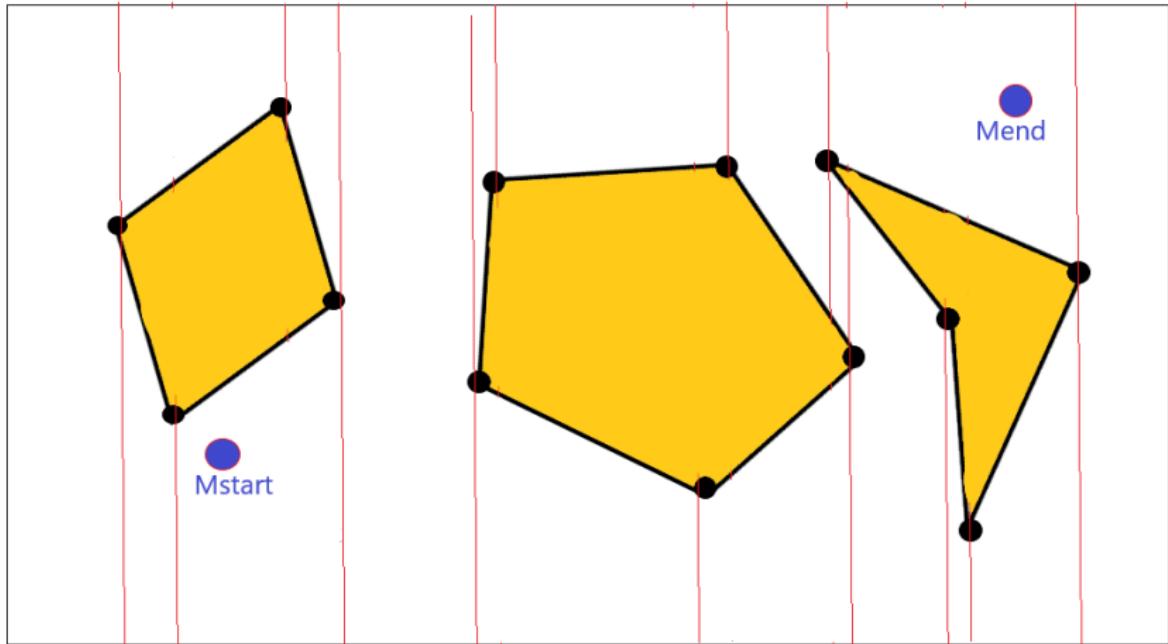
Pasul 1: determinarea spațiului liber

- ▶ Trapezele din interiorul obstacolelor au muchia “top” inclusă în frontiera superioară a unui obstacol: **aceste trapeze sunt eliminate**
- ▶ După acest pas se obține o hartă trapezoidală a spațiului liber \mathcal{C}_I , notată $\mathcal{T}(\mathcal{C}_I)$.

Illustrare



Illustrare



Pasul 2: determinarea unui drum

Date M_{start} , M_{end} se caută un drum de la M_{start} la M_{end} în interiorul spațiului liber.

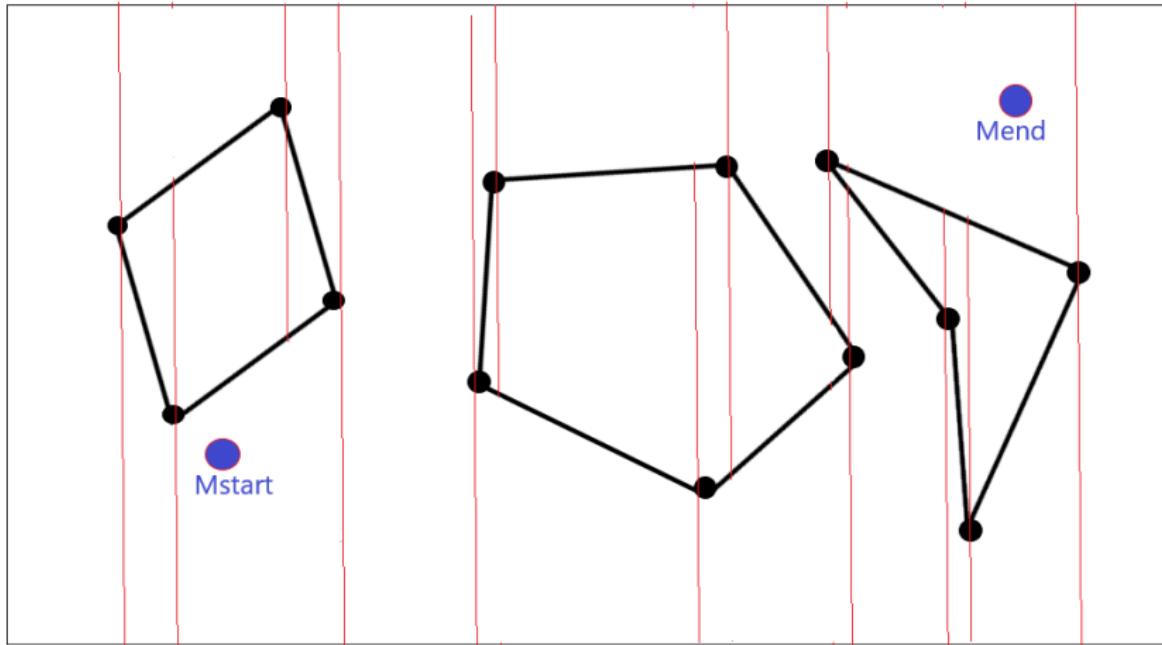
- Dacă M_{start} și M_{end} sunt în interiorul același trapez: segmentul $[M_{\text{start}} M_{\text{end}}]$.

Pasul 2: determinarea unui drum

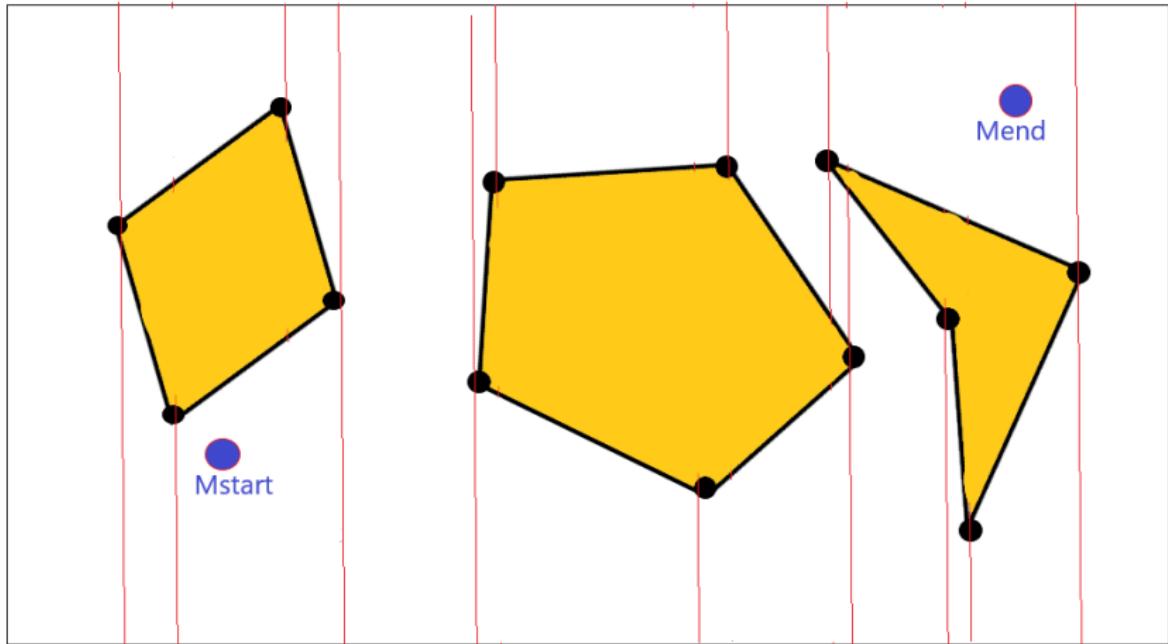
Date M_{start} , M_{end} se caută un drum de la M_{start} la M_{end} în interiorul spațiului liber.

- ▶ Dacă M_{start} și M_{end} sunt în interiorul același trapez: segmentul $[M_{\text{start}} M_{\text{end}}]$.
- ▶ Dacă sunt în trapeze diferite: se folosesc centrele de greutate (mijloacele liniilor mijlocii) ale trapezelor în care sunt situate punctele și mijloacele laturilor adiacente (muchii verticale!). Se utilizează un graf asociat spațiului liber (graful drumurilor - \mathcal{G}_d), care poate fi construit în timp liniar din $\mathcal{T}(\mathcal{C}_I)$. Nodurile acestui graf sunt centrele de greutate ale trapezelor și mijloacele extensiilor verticale.

Illustrare

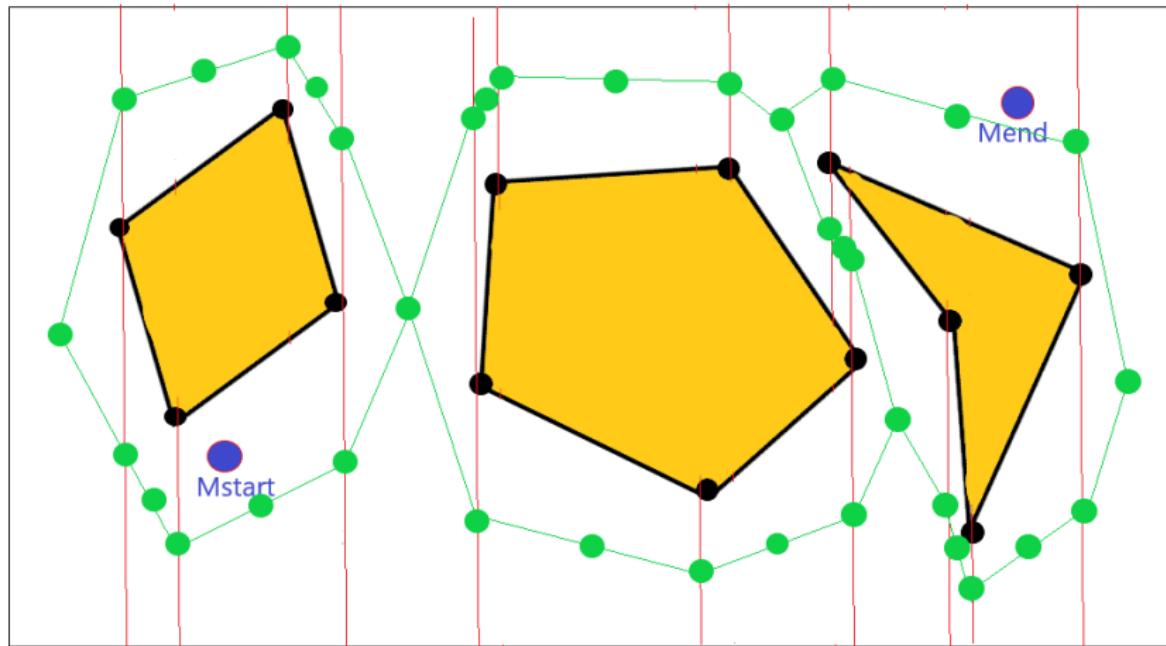


Illustrare



D

Illustrare



D

Algoritm DETERMINA SPATIU LIBER (S)

- ▶ **Input.** O mulțime \mathcal{P} de poligoane disjuncte.

Algoritm DETERMINA SPATIUL LIBER (S)

- ▶ **Input.** O mulțime \mathcal{P} de poligoane disjuncte.
- ▶ **Output.** O hartă trapezoidală \mathcal{C}_l a spațiului liber (pentru un robot-punct).

Algoritm DETERMINA SPATIU LIBER (S)

- ▶ **Input.** O mulțime \mathcal{P} de poligoane disjuncte.
 - ▶ **Output.** O hartă trapezoidală \mathcal{C}_l a spațiului liber (pentru un robot-punct).
1. Fie S mulțimea muchiilor poligoanelor din \mathcal{P} .

Algoritm DETERMINA SPATIUL LIBER (S)

- ▶ **Input.** O mulțime \mathcal{P} de poligoane disjuncte.
 - ▶ **Output.** O hartă trapezoidală \mathcal{C}_l a spațiului liber (pentru un robot-punct).
1. Fie S mulțimea muchiilor poligoanelor din \mathcal{P} .
 2. Determină harta trapezoidală $\mathcal{T}(S)$, folosind algoritmul HARTA TRAPEZOIDALA.

Algoritm DETERMINA SPATIU LIBER (S)

- ▶ **Input.** O mulțime \mathcal{P} de poligoane disjuncte.
 - ▶ **Output.** O hartă trapezoidală \mathcal{C}_l a spațiului liber (pentru un robot-punct).
1. Fie S mulțimea muchiilor poligoanelor din \mathcal{P} .
 2. Determină harta trapezoidală $\mathcal{T}(S)$, folosind algoritmul HARTA TRAPEZOIDALA.
 3. Elimină trapezele situate în interiorul poligoanelor și returnează subdiviziunea obținuta.

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_l)$, \mathcal{G}_d , M_{start} , M_{end})

- **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_l)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_l)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_l)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
- ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_l)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_l)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. caută trapeze în $\mathcal{T}(\mathcal{C}_l)$ conținând M_{start} , respectiv M_{end} .

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. caută trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_l)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_l)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_l)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)
 4. cauță un drum în \mathcal{G}_d de la v_{start} la v_{end} folosind BFS

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)
 4. cauță un drum în \mathcal{G}_d de la v_{start} la v_{end} folosind BFS
 5. **if** există drum δ

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)
 4. cauță un drum în \mathcal{G}_d de la v_{start} la v_{end} folosind BFS
 5. **if** există drum δ
 6. **then** indică drumul $[M_{\text{start}} v_{\text{start}}] \cup \delta \cup [v_{\text{end}} M_{\text{end}}]$

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)
 4. cauță un drum în \mathcal{G}_d de la v_{start} la v_{end} folosind BFS
 5. **if** există drum δ
 6. **then** indică drumul $[M_{\text{start}} v_{\text{start}}] \cup \delta \cup [v_{\text{end}} M_{\text{end}}]$
 7. **else** raportează că nu există drum de la M_{start} la M_{end}

Algoritm DETERMINADRUM ($\mathcal{T}(\mathcal{C}_I)$, \mathcal{G}_d , M_{start} , M_{end})

- ▶ **Input.** Harta trapezoidală $\mathcal{T}(\mathcal{C}_I)$ a spațiului liber, graful drumurilor \mathcal{G}_d , punctul de start M_{start} , punctul final M_{end} .
 - ▶ **Output.** Un drum de la M_{start} la M_{end} , dacă există. În caz contrar, algoritmul precizează că nu există un drum.
1. cauță trapeze în $\mathcal{T}(\mathcal{C}_I)$ conținând M_{start} , respectiv M_{end} .
 2. **if** există Δ_{start} , respectiv Δ_{end} conținând M_{start} , respectiv M_{end}
 3. **then** fie v_{start} și v_{end} centrele Δ_{start} , Δ_{end} (noduri din \mathcal{G}_d)
 4. cauță un drum în \mathcal{G}_d de la v_{start} la v_{end} folosind BFS
 5. **if** există drum δ
 6. **then** indică drumul $[M_{\text{start}} v_{\text{start}}] \cup \delta \cup [v_{\text{end}} M_{\text{end}}]$
 7. **else** raportează că nu există drum de la M_{start} la M_{end}
 8. **else** raportează că nu există drum de la M_{start} la M_{end}

Rezultatul principal

- ▶ **Teoremă.** Fie \mathcal{R} un robot-punct care se deplasează într-o mulțime S de obstacole poligonale, având în total n muchii. Utilizând timp mediu de preprocesare $O(n \log n)$ pentru mulțimea S , un drum liber de coliziuni între două puncte fixate poate fi calculat (dacă există!) în timp mediu $O(n)$.