MODEL 1 (2023-2024)

Problema 1 (10p)
Dați exemplu de o problemă de decizie care să fie în clasa NP dar nu și în clasa NP-hard. Justificați!

Soluție: Un exemplu de problemă care se crede că este în NP dar nu este NP-hard este problema Izomorfismului de Grafuri (Graph Isomorphism).

Problema Izomorfismului de Grafuri:

Intrare: Două grafuri $G_1 = (V_1, E_1)$ și $G_2 = (V_2, E_2)$
Întrebare: Există o bijecție $f: V_1 \rightarrow V_2$ astfel încât $(u,v) \in E_1$ dacă și numai dacă $(f(u),f(v)) \in E_2$?
Justificare:

Problema este în NP:

Dacă ni se dă o soluție (o bijecție f între nodurile celor două grafuri), putem verifica în timp polinomial dacă această bijecție păstrează structura muchiilor.
Pentru fiecare muchie (u,v) din E_1, verificăm dacă $(f(u),f(v)) \in E_2$.
Aceasta necesită $O(|E_1|)$ verificări, fiecare în timp constant, deci verificarea se face în timp polinomial.
Nu este cunoscută ca fiind NP-hard:

În ciuda eforturilor substanțiale, nu s-a găsit o reducere polinomială de la o problemă NP-completă la Izomorfismul de Grafuri.
În 2015, László Babai a anunțat un algoritm cvasipolinomial pentru această problemă (timp de rulare 2^(log^c n) pentru o constantă c), sugerând că problema este de fapt mai "ușoară" decât problemele NP-complete.
Dacă Izomorfismul de Grafuri ar fi NP-hard, ar exista o reducere polinomială de la SAT la Izomorfismul de Grafuri, ceea ce ar implica că toate problemele NP se pot rezolva în timp cvasipolinomial. Acest lucru este considerat improbabil în teoria complexității.
Statusul intermediar:

Izomorfismul de Grafuri este considerat a ocupa un loc special în teoria complexității, fiind una dintre puținele probleme naturale care par a fi în NP dar nici în P (nu se cunoaște un algoritm polinomial), nici NP-complete (nu se poate reduce SAT la ea).
Dacă $P \neq NP$, atunci se crede că există probleme între P și NP-complete, iar Izomorfismul de Grafuri este un candidat principal pentru această categorie.
Izomorfismul de Grafuri ilustrează existența problemelor care nu se încadrează perfect în categoriile clasice din teoria complexității, fiind un exemplu verosimil de problemă în NP dar nu în NP-hard.

Problema 2 (10p)
Fie instanța de skip list din figura de mai jos. Să se insereze, pe rând, elementele 40, 23, 60. Când dăm cu banul și primim B, atunci adăugăm un element la nivel H (head) și ne oprim. Dacă primim M, adăugăm un element atunci când avem M (middle) si ne oprim la T (tails). De asemenea lucrăm cu varianta în care fiecare inserție poate adăuga cel mult un singur nivel nou la structură. Să se descrie cum arată structura de skip list după fiecare inserție. Pentru prima inserție indicați (eventual folosind piste săgeți) traseul parcurs pentru a ajunge la poziția pe care ați inserat primul element.

Analiza structurii inițiale de skip list:
Structura corectă a skip list-ului inițial din imagine este:

Nivel 3 (H): -∞, 17, 55, +∞
Nivel 2 (M): -∞, 17, 25, 31, 55, +∞
Nivel 1 (M): -∞, 12, 17, 25, 31, 38, 44, 55, +∞
Nivel 0 (L): -∞, 12, 17, 20, 25, 31, 38, 39, 44, 50, 55, +∞
Inserarea elementului 40:
Traseul de căutare pentru 40:

Începem la vârful listei (-∞) la nivelul 3 (H)
Comparăm 40 cu 17 → 40 > 17, avansăm la 17
Comparăm 40 cu 55 → 40 < 55, coborâm la nivelul 2 (M)
Comparăm 40 cu 31 → 40 > 31, avansăm la 31
Comparăm 40 cu 55 → 40 < 55, coborâm la nivelul 1 (M)
Comparăm 40 cu 38 → 40 > 38, avansăm la 38
Comparăm 40 cu 44 → 40 < 44, coborâm la nivelul 0 (L)
Comparăm 40 cu 39 → 40 > 39, avansăm la 39
Comparăm 40 cu 44 → 40 < 44, aceasta este poziția de inserare (între 39 și 44)
Determinarea nivelurilor pentru 40:

Presupunem că aruncăm banul și obținem M (probabilitate 1/2)
Inserăm 40 la nivelurile 0 și 1 (M)
Skip list după inserarea lui 40:

Nivel 3 (H): -∞, 17, 55, +∞
Nivel 2 (M): -∞, 17, 25, 31, 55, +∞
Nivel 1 (M): -∞, 12, 17, 25, 31, 38, 40, 44, 55, +∞ (40 adăugat aici)
Nivel 0 (L): -∞, 12, 17, 20, 25, 31, 38, 39, 40, 44, 50, 55, +∞ (40 adăugat aici)
Inserarea elementului 23:
Traseul de căutare pentru 23:

Începem la vârful listei (-∞) la nivelul 3 (H)
Comparăm 23 cu 17 → 23 > 17, avansăm la 17
Comparăm 23 cu 55 → 23 < 55, coborâm la nivelul 2 (M)
Comparăm 23 cu 25 → 23 < 25, coborâm la nivelul 1 (M)
Comparăm 23 cu 17 → 23 > 17, avansăm la 17
Comparăm 23 cu 25 → 23 < 25, coborâm la nivelul 0 (L)
Comparăm 23 cu 20 → 23 > 20, avansăm la 20

Comparăm 23 cu 25 → 23 < 25, aceasta este poziția de inserare (între 20 și 25)
Determinarea nivelurilor pentru 23:

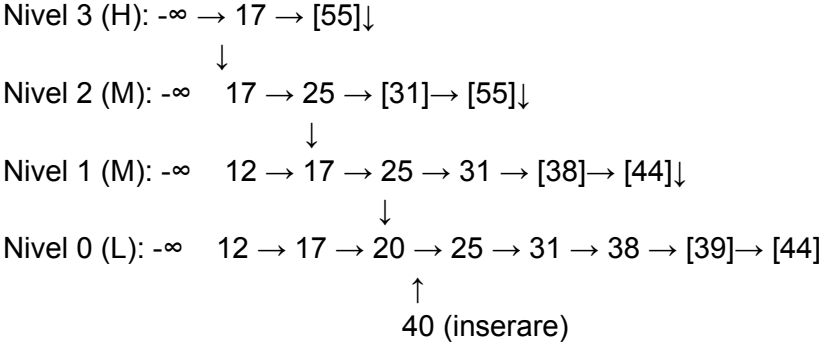
Presupunem că aruncăm banul și obținem B (probabilitate 1/4)
Inserăm 23 la toate nivelurile existente (0, 1, 2, 3)
Skip list după inserarea lui 23:

Nivel 3 (H): -∞, 17, 23, 55, +∞ (23 adăugat aici)
Nivel 2 (M): -∞, 17, 23, 25, 31, 55, +∞ (23 adăugat aici)
Nivel 1 (M): -∞, 12, 17, 23, 25, 31, 38, 40, 44, 55, +∞ (23 adăugat aici)
Nivel 0 (L): -∞, 12, 17, 20, 23, 25, 31, 38, 39, 40, 44, 50, 55, +∞ (23 adăugat aici)
Inserarea elementului 60:
Traseul de căutare pentru 60:

Începem la vârful listei (-∞) la nivelul 3 (H)
Comparăm 60 cu 17 → 60 > 17, avansăm la 17
Comparăm 60 cu 23 → 60 > 23, avansăm la 23
Comparăm 60 cu 55 → 60 > 55, avansăm la 55
Comparăm 60 cu +∞ → 60 < +∞, coborâm la nivelul 2 (M)
Avansăm până la 55 prin comparații similare
Comparăm 60 cu +∞ → 60 < +∞, coborâm la nivelul 1 (M)
Avansăm până la 55 prin comparații similare
Comparăm 60 cu +∞ → 60 < +∞, coborâm la nivelul 0 (L)
Avansăm până la 55 și identificăm poziția de inserare (între 55 și +∞)
Determinarea nivelurilor pentru 60:

Presupunem că aruncăm banul și obținem M (probabilitate 1/2)
Inserăm 60 la nivelurile 0 și 1 (M)
Skip list după inserarea lui 60:

Nivel 3 (H): -∞, 17, 23, 55, +∞
Nivel 2 (M): -∞, 17, 23, 25, 31, 55, +∞
Nivel 1 (M): -∞, 12, 17, 23, 25, 31, 38, 40, 44, 55, 60, +∞ (60 adăugat aici)
Nivel 0 (L): -∞, 12, 17, 20, 23, 25, 31, 38, 39, 40, 44, 50, 55, 60, +∞ (60 adăugat aici)
Reprezentarea traseului pentru prima inserție:
Code



Problema 3 (10p)
Numim minimum degree spanning tree pentru un graf conex G, un arbore parțial cu proprietatea că gradul maxim al nodurilor este minimizat.

Teoremă: Problema de decizie dacă un graf conex admite un arbore parțial cu noduri de grad cel mult 2 (lanț hamiltonian) este NP-Completă.

Cerință: Arătați că nu poate exista un algoritm aproximativ pentru problema găsirii unui minimum degree spanning tree pentru un graf conex cu factorul de aproximare < 3/2, în ipoteza că P ≠ NP.

Soluție:

Abordarea prin reducere și contradicție:
Vom demonstra prin contradicție că nu poate exista un algoritm aproximativ cu factor < 3/2.
Presupunem că ar exista un astfel de algoritm A cu factor de aproximare $\rho < 3/2$.

Pasul 1: Definirea proprietăților algoritmului presupus

Algoritmul A ar primi ca intrare un graf conex G și ar returna un arbore parțial T astfel încât:

$\text{maxdeg}(T) \leq \rho \cdot \text{maxdeg}(OPT)$, unde maxdeg(T) reprezintă gradul maxim al unui nod în T și OPT este arborele parțial optim cu gradul maxim minimizat.
ρ este factorul de aproximare și presupunem că $\rho < 3/2$.
Pasul 2: Construirea unei reduceri pentru a obține o contradicție

Să considerăm un graf conex arbitrar G pentru care dorim să determinăm dacă admite un lanț hamiltonian (un arbore parțial cu toate nodurile de grad cel mult 2).

Dacă aplicăm algoritmul A pe G:

Dacă există un lanț hamiltonian în G, atunci valoarea optimă pentru maxdeg(OPT) este 2.
În acest caz, algoritmul A ar returna un arbore parțial T cu $\text{maxdeg}(T) \leq \rho \cdot 2 < 3/2 \cdot 2 = 3$.
Deoarece maxdeg(T) trebuie să fie un număr întreg și $\text{maxdeg}(T) < 3$, rezultă că $\text{maxdeg}(T) \leq 2$.

Pasul 3: Analiza implicațiilor

Din analiza de mai sus, rezultă că algoritmul A poate fi folosit pentru a rezolva problema lanțului hamiltonian după cum urmează:

Aplicăm algoritmul A pe graful G
Dacă arborele parțial T returnat are $\text{maxdeg}(T) \leq 2$, atunci G admite un lanț hamiltonian
Dacă $\text{maxdeg}(T) > 2$, atunci G nu admite un lanț hamiltonian
Dar acest lucru ar însemna că am rezolvat în timp polinomial problema lanțului hamiltonian, care este NP-completă conform teoremei date.

Pasul 4: Contradicția

Deoarece am reușit să rezolvăm în timp polinomial o problemă NP-completă, acest lucru ar implica P = NP, ceea ce contrazice ipoteza noastră inițială că P ≠ NP.

Concluzie: Prin urmare, sub ipoteza P ≠ NP, nu poate exista un algoritm de aproximare pentru problema minimum degree spanning tree cu un factor de aproximare $\rho < 3/2$.

Acest rezultat este strâns legat de imposibilitatea aproximării sub un anumit prag pentru probleme NP-hard, similar cu rezultatele pentru alte probleme de optimizare cum ar fi acoperirea cu mulțimi sau problema comis-voiajorului.

Problema 4
Fie M = {P_1, P_2, ..., P_9}, unde P_1 = (-6,1), P_2 = (-3,0), P_3 = (-2,1), P_4 = (-1,-2), P_5 = (1,-1), P_6 = (2,0), P_7 = (3,2), P_8 = (4,0), P_9 = (α,β), cu α > 0, β ∈ ℝ. Notăm cu L_1 lista vârfurilor care determină marginea inferioară a frontierei acoperirii convexe a lui M, obținută pe parcursul Graham's scan, varianta Andrew.

a) Pentru α = 0, β = 0 detaliați cum evoluează L_1 pe parcursul Graham's scan, varianta Andrew. (6p)
Soluție: Pentru α = 0, β = 0, avem mulțimea M = {P_1, P_2, ..., P_8, P_9}, unde P_9 = (0,0).

Algoritmul Graham's scan, varianta Andrew, pentru construcția marginii inferioare a acoperirii convexe:

Sortăm punctele după coordonata x (și după y în caz de egalitate): Punctele sortate: P_9(0,0), P_1(-6,1), P_2(-3,0), P_3(-2,1), P_4(-1,-2), P_5(1,-1), P_6(2,0), P_7(3,2), P_8(4,0)

Construim lanțul inferior L_1 examinând punctele de la stânga la dreapta și menținând o cotitură la dreapta:

Inițializare: L_1 = [P_1]

Adăugăm P_2(-3,0): L_1 = [P_1, P_2]

Adăugăm P_3(-2,1): Verificăm dacă [P_1, P_2, P_3] formează o cotitură la dreapta: (P_3.x - P_2.x)(P_2.y - P_1.y) - (P_3.y - P_2.y)(P_2.x - P_1.x) = ((-2) - (-3))(0 - 1) - (1 - 0)((-3) - (-6)) = (1)(-1) - (1)(3) = -1 - 3 = -4 < 0 Este cotitură la dreapta, deci P_3 nu este adăugat, L_1 rămâne [P_1, P_2]

Adăugăm P_4(-1,-2): Verificăm dacă [P_1, P_2, P_4] formează o cotitură la dreapta: (P_4.x - P_2.x)(P_2.y - P_1.y) - (P_4.y - P_2.y)(P_2.x - P_1.x) = ((-1) - (-3))(0 - 1) - ((-2) - 0)((-3) - (-6)) = (2)(-1) - (-2)(3) = -2 + 6 = 4 > 0 Nu este cotitură la dreapta, eliminăm P_2 din L_1

Verificăm acum dacă [P_1, P_4] formează o cotitură la dreapta: Cum avem doar două puncte, le păstrăm automat L_1 = [P_1, P_4]

Adăugăm P_9(0,0): Verificăm dacă [P_1, P_4, P_9] formează o cotitură la dreapta: (P_9.x - P_4.x)(P_4.y - P_1.y) - (P_9.y - P_4.y)(P_4.x - P_1.x) = ((0) - (-1))((-2) - 1) - ((0) - (-2))((-1) - (-6)) = (1)(-3) - (2)(5) = -3 - 10 = -13 < 0 Este cotitură la dreapta, deci P_9 este adăugat L_1 = [P_1, P_4, P_9]

Adăugăm P_5(1,-1): Verificăm dacă [P_4, P_9, P_5] formează o cotitură la dreapta: (P_5.x - P_9.x)(P_9.y - P_4.y) - (P_5.y - P_9.y)(P_9.x - P_4.x) = (1 - 0)(0 - (-2)) - ((-1) - 0)(0 - (-1)) = (1)(2) - (-1)(1) = 2 + 1 = 3 > 0 Nu este cotitură la dreapta, eliminăm P_9

Verificăm acum dacă [P_1, P_4, P_5] formează o cotitură la dreapta: (P_5.x - P_4.x)(P_4.y - P_1.y) - (P_5.y - P_4.y)(P_4.x - P_1.x) = (1 - (-1))((-2) - 1) - ((-1) - (-2))((-1) - (-6)) = (2)(-3) - (1)(5) = -6 - 5 = -11 < 0 Este cotitură la dreapta, deci P_5 este adăugat L_1 = [P_1, P_4, P_5]

Adăugăm $P_6(2,0)$: Verificăm dacă $[P_4, P_5, P_6]$ formează o cotitură la dreapta: $(P_6.x - P_5.x)(P_5.y - P_4.y) - (P_6.y - P_5.y)(P_5.x - P_4.x) = (2 - 1)((-1) - (-2)) - (0 - (-1))(1 - (-1)) = (1)(1) - (1)(2) = 1 - 2 = -1 < 0$ Este cotitură la dreapta, deci P_6 este adăugat $L_1 = [P_1, P_4, P_5, P_6]$

Adăugăm $P_7(3,2)$: Verificăm dacă $[P_5, P_6, P_7]$ formează o cotitură la dreapta: $(P_7.x - P_6.x)(P_6.y - P_5.y) - (P_7.y - P_6.y)(P_6.x - P_5.x) = (3 - 2)(0 - (-1)) - (2 - 0)(2 - 1) = (1)(1) - (2)(1) = 1 - 2 = -1 < 0$ Este cotitură la dreapta, deci P_7 este adăugat $L_1 = [P_1, P_4, P_5, P_6, P_7]$

Adăugăm $P_8(4,0)$: Verificăm dacă $[P_6, P_7, P_8]$ formează o cotitură la dreapta: $(P_8.x - P_7.x)(P_7.y - P_6.y) - (P_8.y - P_7.y)(P_7.x - P_6.x) = (4 - 3)(2 - 0) - (0 - 2)(3 - 2) = (1)(2) - (-2)(1) = 2 + 2 = 4 > 0$ Nu este cotitură la dreapta, eliminăm P_7

Verificăm dacă $[P_5, P_6, P_8]$ formează o cotitură la dreapta: $(P_8.x - P_6.x)(P_6.y - P_5.y) - (P_8.y - P_6.y)(P_6.x - P_5.x) = (4 - 2)(0 - (-1)) - (0 - 0)(2 - 1) = (2)(1) - (0)(1) = 2 > 0$ Nu este cotitură la dreapta, eliminăm P_6

Verificăm dacă $[P_4, P_5, P_8]$ formează o cotitură la dreapta: $(P_8.x - P_5.x)(P_5.y - P_4.y) - (P_8.y - P_5.y)(P_5.x - P_4.x) = (4 - 1)((-1) - (-2)) - (0 - (-1))(1 - (-1)) = (3)(1) - (1)(2) = 3 - 2 = 1 > 0$ Nu este cotitură la dreapta, eliminăm P_5

Verificăm dacă $[P_1, P_4, P_8]$ formează o cotitură la dreapta: $(P_8.x - P_4.x)(P_4.y - P_1.y) - (P_8.y - P_4.y)(P_4.x - P_1.x) = (4 - (-1))((-2) - 1) - (0 - (-2))((-1) - (-6)) = (5)(-3) - (2)(5) = -15 - 10 = -25 < 0$ Este cotitură la dreapta, deci P_8 este adăugat $L_1 = [P_1, P_4, P_8]$

Prin urmare, evoluția completă a listei L_1 este:

$[P_1]$
$[P_1, P_2]$
$[P_1, P_2]$ (P_3 respins)
$[P_1, P_4]$ (P_2 eliminat)
$[P_1, P_4, P_9]$
$[P_1, P_4, P_5]$ (P_9 eliminat)
$[P_1, P_4, P_5, P_6]$
$[P_1, P_4, P_5, P_6, P_7]$
$[P_1, P_4, P_8]$ (P_7, P_6, P_5 eliminate)
Rezultat final: $L_1 = [P_1, P_4, P_8] = [(-6,1), (-1,-2), (4,0)]$

b) Pentru $\alpha = 2$, $\beta = 0$, detaliați cum evoluează L_1 pe parcursul Graham's scan, varianta Andrew. (6p)
Soluție: Pentru $\alpha = 2$, $\beta = 0$, avem $P_9 = (2,0)$, care este exact același punct ca P_6. Vom ține cont că avem două puncte identice.

Sortăm punctele după coordonata x: $P_1(-6,1)$, $P_2(-3,0)$, $P_3(-2,1)$, $P_4(-1,-2)$, $P_5(1,-1)$, $P_6/P_9(2,0)$, $P_7(3,2)$, $P_8(4,0)$

Aplicăm algoritmul Graham-Andrew pentru a construi lanțul inferior L_1:

Inițializare: $L_1 = [P_1]$

Continuăm similar cu punctul anterior, dar trebuie să fim atenți că P_6 și P_9 sunt identice. Când ajungem la P_9, vom avea deja P_6 în L_1, deci acest punct nu va aduce nicio modificare.

Parcurgând algoritmul similar cu cazul precedent, obținem evoluția listei L_1:

$[P_1]$
$[P_1, P_2]$
$[P_1, P_2]$ (P_3 respins)
$[P_1, P_4]$ (P_2 eliminat)
$[P_1, P_4, P_5]$
$[P_1, P_4, P_5, P_6/P_9]$
$[P_1, P_4, P_5, P_6/P_9, P_7]$
$[P_1, P_4, P_8]$ (P_7, P_6/P_9, P_5 eliminate)
Rezultat final: $L_1 = [P_1, P_4, P_8] = [(-6,1), (-1,-2), (4,0)]$

Observăm că rezultatul final este același ca în cazul anterior, deoarece repetarea unui punct din interiorul lanțului nu afectează marginea inferioară a acoperirii convexe.

c) Dați exemplu de pereche (α,β) pentru care L_1 are un număr minim de puncte. Justificați! (2p)
Soluție: Lista L_1 reprezintă marginea inferioară a acoperirii convexe. Un număr minim de puncte în L_1 ar fi 2, ceea ce ar însemna că avem doar punctul cel mai din stânga și cel mai din dreapta.

Acest lucru s-ar întâmpla dacă toate punctele intermediare ar fi fie deasupra dreptei care unește punctele extreme, fie eliminate în timpul algoritmului Graham-Andrew.

Pentru a obține un L_1 cu doar 2 puncte, putem plasa P_9 foarte jos, astfel încât să devină cel mai jos punct și să elimine toate celelalte puncte intermediare.

Alegem $(\alpha,\beta) = (0,-10)$.

Justificare:

Cu $P_9 = (0,-10)$, acest punct devine extrem de jos
După sortare, punctul $P_9(0,-10)$ va fi procesat înainte de $P_5(1,-1)$ și celelalte puncte din dreapta
Când algoritmul ajunge la P_9, verificăm cotitura făcută de $[P_1, P_4, P_9]$
Deoarece P_9 este atât de jos, va crea o cotitură la dreapta și va fi adăugat în L_1
Ulterior, toate punctele din dreapta lui P_9 care sunt procesate vor fi comparate cu $[P_4, P_9]$
Aceste puncte (P_5, P_6, etc.) vor crea o cotitură la stânga și vor elimina P_9
Procesul continuă și se ajunge la un lanț inferior format doar din punctele extreme P_1 și P_8
Prin urmare, pentru $(\alpha,\beta) = (0,-10)$, L_1 va conține doar $[P_1, P_8] = [(-6,1), (4,0)]$, care reprezintă numărul minim de puncte (2) ce pot forma marginea inferioară a acoperirii convexe.

Problema 5
Fie punctele A = (8,5), B = (7,2), C = (3,2), D = (6,8), E = (10,4), F = (7,6), G = (6,-1).

a) Justificați că mulțimea {A, B, C, D, E, F, G} este mulțimea vârfurilor unui poligon y-monoton P. (3p)
Soluție:

Un poligon este y-monoton dacă orice linie orizontală îl intersectează în cel mult două puncte sau într-un segment conex.

Pasul 1: Sortăm punctele după coordonata y în ordine descrescătoare:

D(6,8)
F(7,6)
A(8,5)
E(10,4)
B(7,2)
C(3,2)
G(6,-1)
Pasul 2: Construim poligonul unind punctele în această ordine: P = [D, F, A, E, B, C, G, D]

Pasul 3: Verificăm y-monotonia:

Pentru orice valoare y între 8 și -1, o linie orizontală la acea înălțime va intersecta poligonul în exact două puncte (unul pe lanțul din stânga și unul pe lanțul din dreapta).
Excepție fac valorile y = 8, y = 6, y = 5, y = 4, y = 2, și y = -1, care corespund înălțimilor vârfurilor. La aceste înălțimi, linia orizontală fie trece printr-un singur vârf, fie intersectează poligonul într-un segment (în cazul y = 2, când trecem prin B și C).
Poligonul P format de aceste puncte poate fi împărțit în două lanțuri monotone în y:

Lanțul din stânga: [D, C, G]
Lanțul din dreapta: [D, F, A, E, B, G]
Ambele lanțuri sunt strict monotone în coordonata y (valorile y scad strict când parcurgem lanțurile de sus în jos).

Prin urmare, mulțimea {A, B, C, D, E, F, G} poate forma vârfurile unui poligon y-monoton.

b) Se aplică pentru P algoritmul de triangulare a poligoanelor monotone în timp liniar. Indicați ce vârfuri corespund cazurilor 1, 2a, respectiv 2b. (4p)
Soluție:

Algoritmul de triangulare a poligoanelor y-monotone funcționează astfel:

Sortăm vârfurile după coordonata y
Inițializăm un stack S cu primele două vârfuri
Procesăm restul vârfurilor de sus în jos, aplicând unul dintre cazurile:
Cazul 1: Vârful curent și cele două de pe stack sunt pe lanțuri diferite

Cazul 2a: Vârful curent și cele două de pe stack sunt pe același lanț și formează un unghi reflex
Cazul 2b: Vârful curent și cele două de pe stack sunt pe același lanț și formează un unghi convex
Poligonul nostru sortat după y: [D, F, A, E, B, C, G]

Stabilim lanțurile:

Lanțul stâng: D, C, G
Lanțul drept: D, F, A, E, B, G
Aplicăm algoritmul:

Inițializăm S = [D, F]

Procesăm A:

D și F sunt pe lanțul drept, deci A este pe același lanț
Verificăm unghiul format de D, F și A: este un unghi convex
Corespunde Cazului 2b
Adăugăm diagonala D-A
Stack devine S = [D, A]
Procesăm E:

D și A sunt pe lanțul drept, deci E este pe același lanț
Verificăm unghiul format de D, A și E: este un unghi convex
Corespunde Cazului 2b
Adăugăm diagonala D-E
Stack devine S = [D, E]
Procesăm B:

D și E sunt pe lanțul drept, deci B este pe același lanț
Verificăm unghiul format de D, E și B: este un unghi reflex
Corespunde Cazului 2a
Eliminăm E din stack și adăugăm diagonala A-B
Stack devine S = [D, B]
Procesăm C:

D este pe lanțul stâng, B pe lanțul drept, iar C pe lanțul stâng
Corespunde Cazului 1
Adăugăm diagonala B-C
Stack devine S = [D, C]
Procesăm G:

D și C sunt pe lanțul stâng, G este pe ambele lanțuri (fiind ultimul vârf)
Verificăm unghiul format de D, C și G: este un unghi convex
Corespunde Cazului 2b
Adăugăm diagonala D-G
Stack devine S = [D, G]

Acum G este ultimul vârf și terminăm algoritmul
Vârfurile corespunzătoare fiecărui caz:

Cazul 1: C (când procesăm C, avem vârfurile D, B, C pe lanțuri diferite)
Cazul 2a: B (când procesăm B, formează un unghi reflex cu D și E pe același lanț)
Cazul 2b: A, E, G (formează unghiuri convexe cu vârfurile precedente de pe același lanț)
c) Aplicați metoda din teorema galeriei de artă în cazul poligonului P, indicând o posibilă amplasare a camerelor. (3p)
Soluție:

Teorema galeriei de artă afirmă că pentru un poligon simplu cu n vârfuri, sunt suficienți ⌊n/3⌋ gardieni (camere) pentru a supraveghea întregul poligon.

Pentru poligonul P cu 7 vârfuri, avem nevoie de ⌊7/3⌋ = 2 camere.

Metoda de demonstrație a teoremei implică:

Triangularea poligonului
Colorarea vârfurilor cu trei culori astfel încât niciun triunghi să nu aibă două vârfuri de aceeași culoare
Alegerea culorii care apare cel mai puțin și amplasarea camerelor în aceste vârfuri
Folosind triangularea obținută la punctul b), avem triunghiurile:

T1: D-F-A
T2: D-A-E
T3: D-E-B
T4: D-B-C
T5: D-C-G
Colorăm vârfurile cu trei culori (Roșu, Verde, Albastru):

D: Roșu
F: Verde
A: Albastru
E: Verde
B: Albastru
C: Verde
G: Albastru
Numărul de vârfuri pentru fiecare culoare:

Roșu: 1 vârf (D)
Verde: 3 vârfuri (F, E, C)
Albastru: 3 vârfuri (A, B, G)
Culoarea care apare cel mai puțin este Roșu, dar avem nevoie de cel puțin 2 camere conform teoremei. Putem alege să amplasăm camere în vârfurile colorate cu Roșu și cu o a doua culoare care apare în număr minim (alegem oricare dintre Verde sau Albastru, ambele având 3 vârfuri).

O amplasare posibilă a camerelor ar fi:

O cameră în vârful D (Roșu)
O cameră în vârful F (Verde) sau alternativ în A (Albastru)
Această amplasare garantează supravegherea întregului poligon P, conform teoremei galeriei de artă.

Problema 6 (10p)
Dați exemplu de mulțime M cu 6 elemente din R^2 care să admită o triangulare ce conține 7 fețe, una dintre submulțimile sale cu 5 elemente N_1 să admită o triangulare ce conține 3 fețe și altă submulțime a lui M cu 5 elemente, N_2, să admită o triangulare ce conține 5 fețe. Justificați alegerea făcută.

Soluție:

Vom aplica formula ce leagă numărul de puncte, numărul de puncte de pe frontiera acoperirii convexe și numărul de triunghiuri într-o triangulare:

$T = 2n - h - 2$

unde:

T = numărul de triunghiuri (fețe)
n = numărul total de puncte
h = numărul de puncte pe frontiera acoperirii convexe
Condiții pentru mulțimea M:
M are 6 elemente (n = 6)
Triangularea lui M conține 7 fețe (T = 7)
Din formula de mai sus: 7 = 2(6) - h - 2 7 = 12 - h - 2 h = 3

Deci, 3 puncte din M trebuie să fie pe frontiera acoperirii convexe (formând un triunghi), iar 3 puncte trebuie să fie în interior.

Condiții pentru submulțimea N_1:
N_1 are 5 elemente (n = 5)
Triangularea lui N_1 conține 3 fețe (T = 3)
Din formula: 3 = 2(5) - h_1 - 2 3 = 10 - h_1 - 2 h_1 = 5

Deci, toate cele 5 puncte din N_1 trebuie să fie pe frontiera acoperirii convexe (un pentagon).

Condiții pentru submulțimea N_2:
N_2 are 5 elemente (n = 5)
Triangularea lui N_2 conține 5 fețe (T = 5)
Din formula: 5 = 2(5) - h_2 - 2 5 = 10 - h_2 - 2 h_2 = 3

Deci, 3 puncte din N_2 trebuie să fie pe frontiera acoperirii convexe (un triunghi), iar 2 puncte trebuie să fie în interior.

Construcția mulțimii M:
Considerăm următoarele 6 puncte:

P_1 = (0,0)
P_2 = (10,0)
P_3 = (5,10)
P_4 = (3,3)

$P_5 = (5,2)$
$P_6 = (7,3)$
Aceste puncte sunt astfel poziționate:

P_1, P_2, P_3 formează un triunghi mare (acoperirea convexă a lui M)
P_4, P_5, P_6 sunt în interiorul triunghiului
Submulțimea N_1:
Pentru N_1 cu 5 puncte toate pe frontiera acoperirii convexe, alegem: $N_1 = \{P_1, P_2, P_3, P_4, P_6\}$

Când luăm doar aceste 5 puncte, P_4 și P_6 nu mai sunt în interiorul acoperirii convexe, ci devin parte din frontieră. Aceste 5 puncte formează un pentagon, iar triangularea va conține 3 triunghiuri.

Submulțimea N_2:
Pentru N_2 cu 3 puncte pe frontieră și 2 în interior, alegem: $N_2 = \{P_1, P_2, P_3, P_4, P_5\}$

Aceste puncte formează o configurație cu triunghiul $P_1P_2P_3$ ca acoperire convexă și P_4, P_5 în interior. Triangularea va conține 5 triunghiuri.

Verificare:
Pentru $M = \{P_1, P_2, P_3, P_4, P_5, P_6\}$:

Acoperirea convexă este triunghiul $P_1P_2P_3$ ($h = 3$)
Număr de fețe: $T = 2(6) - 3 - 2 = 7$ ✓
Pentru $N_1 = \{P_1, P_2, P_3, P_4, P_6\}$:

Punctele formează un pentagon ($h_1 = 5$)
Număr de fețe: $T = 2(5) - 5 - 2 = 3$ ✓
Pentru $N_2 = \{P_1, P_2, P_3, P_4, P_5\}$:

Acoperirea convexă este triunghiul $P_1P_2P_3$ ($h_2 = 3$)
Număr de fețe: $T = 2(5) - 3 - 2 = 5$ ✓
Această construcție satisface toate condițiile cerute în problemă. Observăm că alegerea punctelor este critică pentru a obține configurațiile specifice pentru submulțimi, asigurând numărul corect de puncte pe frontieră și în interior pentru fiecare caz.

Problema 7

a) Fie semiplanele: $H_1 : -y - 2 \leq 0$, $H_2 : -x - 2 \leq 0$. Alegeți două semiplane H_3 și H_4 astfel încât intersecția $P = H_1 \cap H_2 \cap H_3 \cap H_4$ să fie un patrulater pentru care punctul (2,2) este vârf al lui P. (3p)

Soluție:

Avem:

H_1: $-y - 2 \leq 0 \Rightarrow y \geq -2$ (semiplnul de deasupra dreptei y = -2)
H_2: $-x - 2 \leq 0 \Rightarrow x \geq -2$ (semiplnul din dreapta dreptei x = -2)
Intersecția $H_1 \cap H_2$ reprezintă cadranul I extins, adică $\{(x,y) \mid x \geq -2, y \geq -2\}$.

Pentru ca intersecția finală să fie un patrulater, trebuie să alegem H_3 și H_4 astfel încât:

Intersecția tuturor semiplanelor să fie mărginită
Punctul (2,2) să fie un vârf al patrulaterului rezultat
Pentru ca (2,2) să fie un vârf, trebuie să fie la intersecția a cel puțin două frontiere de semiplane. Întrucât (2,2) nu se află pe frontierele lui H_1 sau H_2, trebuie să fie la intersecția frontierelor lui H_3 și H_4.

Alegem:

H_3: $x - 2 \leq 0 \Rightarrow x \leq 2$ (semiplnul din stânga dreptei x = 2)
H_4: $y - 2 \leq 0 \Rightarrow y \leq 2$ (semiplnul de sub dreapta y = 2)
Verificăm că (2,2) se află pe frontierele ambelor semiplane H_3 și H_4:

Pentru H_3: $x - 2 = 2 - 2 = 0$ ✓
Pentru H_4: $y - 2 = 2 - 2 = 0$ ✓
Intersecția $P = H_1 \cap H_2 \cap H_3 \cap H_4 = \{(x,y) \mid -2 \leq x \leq 2, -2 \leq y \leq 2\}$ este un pătrat cu vârfurile (-2,-2), (2,-2), (2,2) și (-2,2).

Așadar, prin alegerea semiplanelor H_3: $x \leq 2$ și H_4: $y \leq 2$, obținem un patrulater (un pătrat) care are (2,2) ca vârf.

b) Fie $R = \{H_1, H_2, H_3, H_4\}$. Dați exemplu de două funcții obiectiv f_1 și f_2 astfel că problema de programare liniară dată de R și f_1 să aibă ca mulțime de optim (puncte de maxim) un segment. (7p)

Soluție:

Regiunea fezabilă $R = H_1 \cap H_2 \cap H_3 \cap H_4$ este pătratul $\{(x,y) \mid -2 \leq x \leq 2, -2 \leq y \leq 2\}$.

Pentru prima funcție obiectiv f_1:
Pentru ca problema max f_1 să aibă un unic punct optim, acest punct trebuie să fie unul dintre vârfurile pătratului. Alegem f_1 astfel încât să atingă maximul în vârful (2,2):

$f_1(x,y) = x + y$

Verificăm valorile lui f_1 în vârfurile pătratului:

$f_1(-2,-2) = -2 + (-2) = -4$
$f_1(2,-2) = 2 + (-2) = 0$
$f_1(-2,2) = -2 + 2 = 0$
$f_1(2,2) = 2 + 2 = 4$
Deci, maximul lui f_1 este atins doar în punctul (2,2), care este un unic optimum.

Pentru a doua funcție obiectiv f_2:
Pentru ca problema max f_2 să aibă ca mulțime de optim un segment, trebuie ca f_2 să atingă valoarea maximă de-a lungul unei laturi a pătratului.

Alegem: $f_2(x,y) = y$

Verificăm valorile lui f_2 în vârfurile pătratului:

$f_2(-2,-2) = -2$
$f_2(2,-2) = -2$
$f_2(-2,2) = 2$
$f_2(2,2) = 2$
Observăm că f_2 atinge valoarea maximă 2 în punctele (-2,2) și (2,2), precum și în toate punctele de pe latura superioară a pătratului: $\{(x,2) \mid -2 \leq x \leq 2\}$.

Astfel, mulțimea optimă pentru max f_2 este segmentul $\{(x,2) \mid -2 \leq x \leq 2\}$.

Justificare și desen geometric:
În problema de programare liniară, funcția obiectiv definește un gradient sau o direcție de creștere în planul (x,y). Punctele optime se află la "cea mai îndepărtată" poziție în direcția gradientului, dar limitate la regiunea fezabilă.

Pentru $f_1(x,y) = x + y$, gradientul este (1,1), orientat spre nord-est. Poziția optimă este vârful (2,2), cel mai îndepărtat punct în această direcție.
Pentru $f_2(x,y) = y$, gradientul este (0,1), orientat direct spre nord. Toate punctele de pe latura superioară a pătratului sunt la aceeași "înălțime" maximă, formând un segment de puncte optime.
Aceasta ilustrează un principiu fundamental al programării liniare: soluția optimă se atinge întotdeauna la un vârf al regiunii fezabile (sau de-a lungul unei laturi sau fețe, dacă există multiple soluții optime).

Problema 8
a) Indicați cinci algoritmi menționați la curs pentru care este utilizat sau este relevant testul de orientare. În fiecare caz descrieți, foarte pe scurt, contextul și modul în care este folosit. (15p)
Soluție:

1. Algoritmul Graham's scan pentru calculul acoperirii convexe

Context: Algoritmul determină acoperirea convexă a unui set de puncte în plan.

Modul de utilizare: Testul de orientare este folosit pentru a determina dacă trei puncte consecutive P, Q și R formează o cotitură la stânga sau la dreapta. În varianta clasică a algoritmului, punctele sunt sortate polar în jurul unui punct interior, apoi se parcurge secvența și se elimină punctele care nu formează cotituri la stânga. Formula utilizată este:

Code
orientare(P, Q, R) = (Q.x - P.x) * (R.y - Q.y) - (Q.y - P.y) * (R.x - Q.x)
Dacă rezultatul este pozitiv, avem o cotitură la stânga; dacă este negativ, avem o cotitură la dreapta.

2. Algoritmul de triangulare a poligoanelor monotone

Context: Algoritmul împarte un poligon y-monoton în triunghiuri.

Modul de utilizare: Testul de orientare determină tipul unghiului format de trei vârfuri consecutive - convex sau reflex. Algoritmul procesează vârfurile de sus în jos și folosește orientarea pentru a decide când să adauge diagonale. Este esențial pentru identificarea cazurilor 2a și 2b menționate în problema 5, când vârfurile aparțin aceluiași lanț al poligonului monoton.

3. Algoritmul pentru verificarea dacă un punct se află în interiorul unui poligon

Context: Se dorește să se determine dacă un punct P se află în interiorul unui poligon convex sau arbitrar.

Modul de utilizare: Pentru poligoane convexe, se verifică dacă punctul P se află în aceeași parte a fiecărei laturi a poligonului. Pentru fiecare latură definită de punctele consecutice A și B, se calculează orientare(A, B, P). Dacă toate rezultatele au același semn, punctul este în interiorul poligonului. Pentru poligoane arbitrare, se folosește algoritmul ray-casting, care folosește tot teste de orientare pentru a determina dacă o semidreaptă trasată din punct intersectează un număr par sau impar de laturi.

4. Algoritmul line-sweep (măturare cu linie) pentru determinarea intersecțiilor între segmente

Context: Se dorește găsirea tuturor intersecțiilor dintre un set de segmente în plan.

Modul de utilizare: Testul de orientare este folosit pentru a determina dacă două segmente se intersectează. Dacă avem segmentele AB și CD, verificăm dacă:

C și D sunt de părți diferite ale dreptei AB (orientare(A,B,C) * orientare(A,B,D) < 0)
A și B sunt de părți diferite ale dreptei CD (orientare(C,D,A) * orientare(C,D,B) < 0) Dacă ambele condiții sunt îndeplinite, segmentele se intersectează.
5. Algoritmul Jarvis March (Gift Wrapping) pentru acoperirea convexă

Context: Un alt algoritm pentru calculul acoperirii convexe, care "împachetează" punctele similar cu o bandă elastică.

Modul de utilizare: Algoritmul pornește de la punctul cel mai din stânga și găsește punctul următor prin calcularea unghiului minim față de orizontală. Testul de orientare determină care punct formează cel mai mic unghi cu ultima latură adăugată. Pentru fiecare candidat Q, se verifică orientarea față de ultimele două puncte P și R deja adăugate în acoperirea convexă (orientare(P, R, Q)).

b) Care este complexitatea-timp necesară procesării tuturor evenimentelor în algoritmul lui Fortune? Justificați succint, punctând principalele idei! (5p)
Soluție:

Complexitatea-timp a algoritmului Fortune pentru diagrama Voronoi este O(n log n), unde n este numărul de puncte de intrare.

Justificare succintă:

Numărul de evenimente:

Există n evenimente de tip "site" (unul pentru fiecare punct de intrare)
Există cel mult n-1 evenimente de tip "cerc" (fiecare corespunzând unei muchii Voronoi)
Total: O(n) evenimente
Structuri de date folosite:

Coadă de priorități pentru evenimente: operațiile de inserare și extragere necesită O(log n) timp
Structură de tip balanced binary search tree (arbore binar echilibrat) pentru linia de baleiaj (beach line): operațiile de căutare, inserare și ștergere necesită O(log n) timp
Procesarea evenimentelor:

Pentru fiecare eveniment de tip "site", se inserează o nouă parabolă în linia de baleiaj (beach line) și se verifică dacă apar noi evenimente de cerc: O(log n) timp per eveniment
Pentru fiecare eveniment de tip "cerc", se elimină o parabolă din linia de baleiaj și se verifică dacă apar noi evenimente de cerc: O(log n) timp per eveniment
Totalul:

O(n) evenimente × O(log n) timp per eveniment = O(n log n) timp total
Puncte-cheie:

Algoritmul Fortune folosește o abordare de tip line-sweep (măturare cu linie)
Complexitatea este optimală pentru construirea diagramei Voronoi

Eficiența provine din gestionarea inteligentă a evenimentelor și utilizarea structurilor de date adecvate

Această complexitate este comparabilă cu algoritmii optimi pentru sortare și pentru calculul acoperirii convexe

Algoritmul Fortune reușește să construiască diagrama Voronoi în timp O(n log n), care este optim teoretic, deoarece problema are un lower bound de Ω(n log n) (poate fi redusă la sortare).

MODEL 2 (2021-2022)

Problema 1: Algoritmul de Load Balance - Analiza raportului de aproximare 7/6
Problema se referă la algoritmul de "Load Balance" (echilibrarea sarcinilor) prezentat în cadrul Cursului 2, slide-ul 19. Trebuie să analizăm de ce acest algoritm, în anumite condiții specifice, este 7/6-aproximativ.

Ce înțelegem prin algoritm α-aproximativ
Un algoritm este α-aproximativ dacă, pentru orice instanță a problemei, soluția oferită de algoritm (ALG) satisface relația ALG ≤ α·OPT, unde OPT reprezintă valoarea soluției optime. Cu cât α este mai aproape de 1, cu atât algoritmul este mai bun.

Datele problemei
Avem următoarele informații:

n task-uri care trebuie procesate
m = 6 mașini disponibile pentru procesare
Fiecare task i durează t_i unități de timp, unde $t_i \leq 50$ pentru orice i
Timpul total al tuturor task-urilor este $T = \Sigma_{i=1}^{n} t_i = 3000$
Analiza teoretică a algoritmului
Pentru a înțelege de ce algoritmul este 7/6-aproximativ în aceste condiții, trebuie să derivăm bound-uri (limite) pentru valorile OPT și ALG.

1. Bound inferior pentru OPT
În orice problemă de scheduling, valoarea optimului nu poate fi mai mică decât:

Timpul maxim al unui task individual ($\max(t_i)$)
Timpul mediu necesar dacă task-urile ar fi distribuite perfect (T/m)
Așadar: $OPT \geq \max(\max(t_i), T/m)$

În cazul nostru: $OPT \geq \max(50, 3000/6) = \max(50, 500) = 500$

Prin urmare, știm că soluția optimă nu poate termina mai repede de 500 unități de timp.

2. Bound superior pentru ALG (List Scheduling)
În algoritmul List Scheduling, task-urile sunt asignate în ordine la mașina care devine disponibilă prima. Se poate demonstra matematic că acest algoritm satisface:

$ALG \leq T/m + (m-1)/m \cdot \max(t_i)$

Această formulă poate fi înțeleasă intuitiv astfel:

T/m reprezintă timpul de procesare mediu dacă task-urile ar fi distribuite uniform
$(m-1)/m \cdot \max(t_i)$ reprezintă un termen de corecție care ia în considerare dezechilibrul între mașini
Demonstrația acestui bound implică analiza momentului în care se termină ultimul task:

Fie t momentul când începe ultimul task care se termină
La momentul t, toate mașinile sunt ocupate (altfel ultimul task ar fi început mai devreme)
Timpul total de procesare până la momentul t este cel puțin $t \cdot m$
Timpul total este T, deci $T \geq t \cdot m$
Ultimul task durează cel mult $\max(t_i)$
Deci, $ALG \leq t + \max(t_i) \leq T/m + \max(t_i)$
O analiză mai rafinată ne arată că: $ALG \leq T/m + (m-1)/m \cdot \max(t_i)$

3. Calculul raportului de aproximare
Înlocuind valorile din problema noastră:

$ALG \leq 3000/6 + (6-1)/6 \cdot 50 = 500 + 5/6 \cdot 50 = 500 + 41.67 = 541.67$

Astfel, raportul de aproximare este:

$ALG/OPT \leq 541.67/500 = 1.0833...$

Cum $1.0833... < 7/6 = 1.1667...$, deducem că pentru condițiile specificate ($m = 6$, $t_i \leq 50$, $T = 3000$), algoritmul este 7/6-aproximativ.

Concluzii
Algoritmul de Load Balance, deși este în general 2-aproximativ pentru probleme arbitrare de scheduling, devine 7/6-aproximativ în contextul specific al acestei probleme datorită:

Numărului specific de mașini ($m = 6$)
Limitei superioare pentru durata task-urilor ($t_i \leq 50$)
Raportului favorabil între durata maximă a unui task și timpul total (50 fiind relativ mic față de 3000)
Aceste condiții permit o distribuție mai echilibrată a task-urilor între mașini, reducând astfel diferența între soluția algoritmului și soluția optimă.

Problema 2: Împărțirea cadourilor - Algoritm genetic și programare liniară
Această problemă se referă la împărțirea echitabilă a unui set de cadouri între doi copii. Avem n cadouri cu valori (costuri) c_1, c_2, ..., c□ și dorim să le împărțim astfel încât diferența dintre suma cadourilor primite de fiecare copil să fie minimă.

Partea a) Dezvoltarea unui algoritm genetic
Algoritmii genetici sunt metode de optimizare inspirate din procesul de selecție naturală și evoluție biologică. Ei sunt potriviți pentru probleme de optimizare combinatorială precum aceasta.

Reprezentarea cromozomului
În problema noastră, un "cromozom" reprezintă o posibilă împărțire a cadourilor între cei doi copii. Folosim reprezentarea binară:

Un cromozom este un șir binar $X = (x_1, x_2, ..., x$□$)$ unde:
$x_i = 1$ dacă cadoul i merge la primul copil
$x_i = 0$ dacă cadoul i merge la al doilea copil
De exemplu, pentru n = 5 cadouri, cromozomul (1,0,1,0,0) înseamnă că primul copil primește cadourile 1 și 3, iar al doilea copil primește cadourile 2, 4 și 5.

Funcția fitness
Scopul nostru este să minimizăm diferența dintre sumele cadourilor primite de fiecare copil. Definim:

Suma totală a valorilor cadourilor: $S = \Sigma_{i=1}^{n} c_i$
Suma pentru primul copil: $S_1 = \Sigma_{i=1}^{n} c_i \cdot x_i$
Suma pentru al doilea copil: $S_2 = S - S_1 = \Sigma_{i=1}^{n} c_i \cdot (1-x_i)$
Funcția fitness este definită ca: $f(X) = -|S_1 - S_2| = -|2 \cdot S_1 - S|$

Folosim semnul minus pentru că algoritmii genetici tradițional maximizează funcția fitness, dar noi dorim să minimizăm diferența absolută.

Operatori genetici detaliați
Inițializarea populației

Generăm P cromozomi aleatori (de ex., P = 100)
Pentru fiecare cromozom, fiecare bit x_i este 0 sau 1 cu probabilitate egală
Evaluarea fitness-ului

Pentru fiecare cromozom X, calculăm $f(X) = -|2 \cdot \Sigma_{i=1}^{n} c_i \cdot x_i - \Sigma_{i=1}^{n} c_i|$
Selecția Implementăm selecția prin turneu:

Pentru fiecare nouă poziție în generația următoare
Selectăm aleator k cromozomi din populație (de ex., k = 3)
Alegem cromozomul cu cel mai mare fitness din acest grup de k
Încrucișarea (crossover) Implementăm încrucișarea într-un punct:

Cu probabilitate p^c (de ex., $p^c = 0.8$), pentru fiecare pereche de cromozomi părinți

Alegem aleator un punct de tăiere j între 1 și n-1
Creăm doi cromozomi descendenți:
Primul primește primele j bite de la primul părinte și restul de la al doilea
Al doilea primește primele j bite de la al doilea părinte și restul de la primul
Matematic, dacă părinții sunt $X = (x_1, x_2, ..., x_n)$ și $Y = (y_1, y_2, ..., y_n)$, și punctul de tăiere este j, descendenții vor fi:

$X' = (x_1, ..., x_j, y_{j+1}, ..., y_n)$
$Y' = (y_1, ..., y_j, x_{j+1}, ..., x_n)$
Mutația

Cu probabilitate p_m mică (de ex., $p_m = 0.01$), pentru fiecare bit din fiecare cromozom
Inversăm valoarea bitului (0 devine 1, 1 devine 0)
Înlocuirea

Înlocuim complet vechea populație cu noii cromozomi generați prin operațiile de mai sus
Alternativ, putem folosi strategia elitism, păstrând unul sau mai mulți dintre cei mai buni cromozomi
Criteriul de oprire
Algoritmul se oprește când:

S-a atins un număr maxim de generații (de ex., 100)
Fitness-ul celui mai bun cromozom nu s-a îmbunătățit pentru un anumit număr de generații consecutive
S-a găsit o soluție perfectă (diferență zero, dacă este posibil)
Pseudocod complet al algoritmului genetic

Partea b) Formalizarea ca problemă de programare liniară
Programarea liniară este o tehnică pentru optimizarea unei funcții liniare obiectiv sub constrângeri liniare. Pentru problema noastră, trebuie să transformăm minimizarea diferenței absolute într-o problemă de programare liniară.

Variabilele de decizie
$x_i \in \{0, 1\}$ pentru i = 1, 2, ..., n: indică căruia dintre copii îi este dat cadoul i
$x_i = 1$ dacă primul copil primește cadoul i
$x_i = 0$ dacă al doilea copil primește cadoul i
$z \geq 0$: reprezintă diferența absolută dintre sumele celor două grupe de cadouri
Formularea problemei
Obiectivul este minimizarea diferenței absolute:

Minimizează z

Sub constrângerile:

Diferența între suma primului copil și suma celui de-al doilea ≤ z: $S_1 - S_2 \leq z$

Diferența între suma celui de-al doilea și suma primului ≤ z: $S_2 - S_1 \leq z$

Variabilele de decizie sunt binare: $x_i \in \{0,1\}$ pentru i = 1, 2, ..., n

Dezvoltarea constrângerilor
Substituim $S_1 = \Sigma_{i=1}^{n} c_i \cdot x_i$ și $S_2 = \Sigma_{i=1}^{n} c_i \cdot (1-x_i)$:

$S_1 - S_2 \leq z \; \Sigma_{i=1}^{n} c_i \cdot x_i - \Sigma_{i=1}^{n} c_i \cdot (1-x_i) \leq z \; \Sigma_{i=1}^{n} c_i \cdot x_i - \Sigma_{i=1}^{n} c_i + \Sigma_{i=1}^{n} c_i \cdot x_i \leq z \; 2 \cdot \Sigma_{i=1}^{n} c_i \cdot x_i - \Sigma_{i=1}^{n} c_i \leq z \; \Sigma_{i=1}^{n} c_i \cdot (2x_i-1) \leq z$

$S_2 - S_1 \leq z \; \Sigma_{i=1}^{n} c_i \cdot (1-x_i) - \Sigma_{i=1}^{n} c_i \cdot x_i \leq z \; \Sigma_{i=1}^{n} c_i - \Sigma_{i=1}^{n} c_i \cdot x_i - \Sigma_{i=1}^{n} c_i \cdot x_i \leq z \; \Sigma_{i=1}^{n} c_i - 2 \cdot \Sigma_{i=1}^{n} c_i \cdot x_i \leq z \; -\Sigma_{i=1}^{n} c_i \cdot (2x_i-1) \leq z$

Aceste două constrângeri pot fi scrise mai compact ca: $-z \leq \Sigma_{i=1}^{n} c_i \cdot (2x_i-1) \leq z$

Formulare finală a problemei de programare liniară
Minimizează z Sub constrângerile:

$\Sigma_{i=1}^{n} c_i \cdot (2x_i-1) \leq z$
$-\Sigma_{i=1}^{n} c_i \cdot (2x_i-1) \leq z$
$x_i \in \{0,1\}$ pentru i = 1, 2, ..., n
$z \geq 0$
Această formulare este o problemă de programare liniară binară (BILP - Binary Integer Linear Programming), care poate fi rezolvată utilizând algoritmi specifici pentru programare liniară cu variabile întregi, precum branch-and-bound sau branch-and-cut.

Problema 3: Punctul Mα la stânga muchiei orientate AB
În această problemă, trebuie să determinăm pentru ce valori ale parametrului α punctul Mα = (3, 4 + 2α) este situat la stânga muchiei orientate AB, unde A = (1, -2) și B = (5, 6).

Concepte fundamentale
În geometria computațională, determinarea dacă un punct P este la stânga, la dreapta sau pe o dreaptă orientată AB se face utilizând produsul vectorial.

Fie AB vectorul de la A la B și AP vectorul de la A la P. Calculăm produsul vectorial AB × AP în plan:

AB × AP = (xB - xA)(yP - yA) - (yB - yA)(xP - xA)

Interpretarea rezultatului:

Dacă AB × AP > 0, atunci P este la stânga dreptei orientate AB
Dacă AB × AP < 0, atunci P este la dreapta dreptei orientate AB
Dacă AB × AP = 0, atunci P este colinear cu A și B
Rezolvarea detaliată
Avem punctele:

A = (1, -2)
B = (5, 6)
Mα = (3, 4 + 2α)
Pasul 1: Calculăm vectorii AB și AMα
AB = B - A = (5, 6) - (1, -2) = (4, 8) AMα = Mα - A = (3, 4 + 2α) - (1, -2) = (2, 6 + 2α)

Pasul 2: Calculăm produsul vectorial AB × AMα
AB × AMα = (xB - xA)(yMα - yA) - (yB - yA)(xMα - xA) = (5 - 1)((4 + 2α) - (-2)) - (6 - (-2))(3 - 1) = 4(4 + 2α + 2) - 8(2) = 4(6 + 2α) - 16 = 24 + 8α - 16 = 8 + 8α

Pasul 3: Determinăm pentru ce valori ale lui α acest produs este pozitiv
Pentru ca Mα să fie la stânga muchiei orientate AB, trebuie ca AB × AMα > 0: 8 + 8α > 0 8α > -8 α > -1

Interpretare geometrică
Această problemă poate fi înțeleasă geometric astfel:

AB este o muchie orientată (o săgeată) de la A la B
Mα se deplasează de-a lungul unei drepte verticale x = 3 pe măsură ce α variază
Când α > -1, Mα este deasupra dreptei care trece prin A și B suficient de mult încât să fie la stânga muchiei orientate
Când α = -1, Mα se află exact pe dreapta AB
Când α < -1, Mα este la dreapta muchiei orientate AB
O observație importantă este că "stânga" unei muchii orientate depinde de direcția muchiei. Dacă am avea muchia orientată BA (în loc de AB), atunci conceptele de "stânga" și "dreapta" ar fi inversate.

Problema 4: Acoperirea convexă folosind algoritmul Graham scan (varianta Andrew)
Această problemă se referă la determinarea acoperirii convexe pentru o mulțime de puncte utilizând algoritmul Graham's scan, varianta lui Andrew. Acoperirea convexă este cel mai mic poligon convex care conține toate punctele din mulțime.

Descrierea algoritmului Graham-Andrew
Algoritmul constă în următorii pași:

Sortare: Sortăm punctele după coordonata x (sau după coordonata y, dacă dorim să procesăm de jos în sus)
Construcția lanțului inferior: Parcurgem punctele de la stânga la dreapta și construim lanțul inferior al acoperirii convexe
Construcția lanțului superior: Parcurgem punctele de la dreapta la stânga și construim lanțul superior al acoperirii convexe
Concatenare: Unim cele două lanțuri pentru a obține acoperirea convexă completă
Test de cotitură
Pentru a determina dacă un punct trebuie inclus în lanț, folosim un test de cotitură pentru a verifica dacă punctul curent face o cotitură la stânga sau la dreapta față de ultimele două puncte din lanț.

Pentru trei puncte P, Q și R, calculăm produsul vectorial: $(Q-P) \times (R-Q) = (xQ - xP)(yR - yQ) - (yQ - yP)(xR - xQ)$

Dacă rezultatul este pozitiv, avem o cotitură la stânga
Dacă rezultatul este negativ, avem o cotitură la dreapta
Dacă rezultatul este zero, punctele sunt coliniare
Aplicarea algoritmului pe datele problemei
Avem mulțimea de puncte:

$P_1 = (-6, 2)$
$P_2 = (-4, 0)$
$P_3 = (-2, -2)$
$P_4 = (1, -2)$
$P_5 = (2, -1)$
$P_6 = (3, 2)$
$P_7 = (4, 5)$
$P_8 = (7, 2)$
$P_9 = (8, 3)$
Pasul 1: Sortarea punctelor
Punctele sunt deja sortate după coordonata x.

Pasul 2: Construcția lanțului inferior
Inițializăm lanțul inferior cu primele două puncte P_1 și P_2.

Pentru P_3:

Verificăm dacă P_1, P_2, P_3 formează o cotitură la dreapta: $(P_2 - P_1) \times (P_3 - P_2) = ((-4) - (-6), 0 - 2) \times ((-2) - (-4), (-2) - 0) = (2, -2) \times (2, -2) = 2 \cdot (-2) - (-2) \cdot 2 = -4 - (-4) = 0$ Rezultat = 0, punctele sunt coliniare. Îl păstrăm pe P_3.
Pentru P_4:

Verificăm dacă P_2, P_3, P_4 formează o cotitură la dreapta: $(P_3 - P_2) \times (P_4 - P_3) = ((-2) - (-4), (-2) - 0) \times (1 - (-2), (-2) - (-2)) = (2, -2) \times (3, 0) = 2 \cdot 0 - (-2) \cdot 3 = 0 + 6 = 6$ Rezultat > 0, cotitură la stânga, ceea ce nu dorim. Eliminăm P_3.

Verificăm acum dacă P_1, P_2, P_4 formează o cotitură la dreapta: $(P_2 - P_1) \times (P_4 - P_2) = ((-4) - (-6), 0 - 2) \times (1 - (-4), (-2) - 0) = (2, -2) \times (5, -2) = 2 \cdot (-2) - (-2) \cdot 5 = -4 + 10 = 6$ Rezultat > 0, cotitură la stânga, ceea ce nu dorim. Eliminăm P_2.

Verificăm acum dacă P_1, P_4 formează o pereche validă. Avem doar două puncte, deci le păstrăm pe ambele.

Continuăm procesul pentru toate punctele, verificând la fiecare pas dacă ultimele trei puncte din lanț formează o cotitură la dreapta și eliminând penultimul punct dacă nu formează.

La final, lanțul inferior conține punctele $\{P_1, P_4, P_5, P_8, P_9\}$.

Pasul 3: Construcția lanțului superior
Pentru lanțul superior, parcurgem punctele de la dreapta la stânga și aplicăm același algoritm, dar căutăm cotituri la dreapta.

Inițializăm lanțul superior cu ultimele două puncte P_9 și P_8.

Pentru P_7:

Verificăm dacă P_9, P_8, P_7 formează o cotitură la dreapta: $(P_8 - P_9) \times (P_7 - P_8) = ((7) - (8), 2 - 3) \times ((4) - (7), (5) - (2)) = (-1, -1) \times (-3, 3) = (-1) \cdot 3 - (-1) \cdot (-3) = -3 - 3 = -6$ Rezultat < 0, cotitură la dreapta, ceea ce dorim. Păstrăm P_7.
Continuăm procesul pentru toate punctele rămase, obținând lanțul superior.

Pasul 4: Concatenarea lanțurilor
Unim lanțul inferior cu lanțul superior (excluzând primul și ultimul punct din lanțul superior, deoarece acestea sunt deja incluse în lanțul inferior). Obținem acoperirea convexă completă.

Acoperirea convexă finală, lista C_4, este: $\{P_1, P_4, P_5, P_8, P_9, P_7, P_6\}$.

Complexitate și Eficiență
Algoritmul Graham-Andrew are o complexitate totală de $O(n \log n)$, dominată de pasul de sortare inițială. Construcția efectivă a acoperirii convexe (după sortare) are complexitate $O(n)$.

Acest algoritm este eficient și robust, fiind unul dintre cei mai utilizați algoritmi pentru determinarea acoperirii convexe în plan.

Problema 5: Galeria de artă și puncte simetrice
Această problemă se referă la aplicarea teoremei galeriei de artă și la analiza punctelor simetrice într-un poligon.

Partea a) Aplicarea teoremei galeriei de artă
Teorema galeriei de artă - formulare precisă
Teorema galeriei de artă, demonstrată de Václav Chvátal în 1975, afirmă că pentru un poligon simplu cu n vârfuri, sunt întotdeauna suficiente $\lfloor n/3 \rfloor$ camere de supraveghere plasate în vârfuri pentru a supraveghea întregul interior al poligonului.

Metoda de demonstrație a lui Fisk
Demonstrația clasică dată de Steve Fisk implică următorii pași:

Triangularea poligonului: Împărțim poligonul în n-2 triunghiuri prin adăugarea de diagonale care nu se intersectează.
Colorarea vârfurilor: Colorăm vârfurile poligonului cu trei culori astfel încât niciun triunghi să nu aibă două vârfuri de aceeași culoare.
Acest lucru este întotdeauna posibil pentru o triangulare a unui poligon simplu.
Algoritm de colorare:
Alegem un triunghi arbitrar și îi colorăm vârfurile cu trei culori diferite (roșu, verde, albastru)
Pentru fiecare triunghi adiacent, dacă două vârfuri sunt deja colorate, colorăm al treilea vârf cu culoarea rămasă
Continuăm acest proces până când toate vârfurile sunt colorate
Alegerea camerelor: Alegem una din cele trei culori care apare cel mai puțin frecvent. Numărul de vârfuri cu această culoare este cel mult $\lfloor n/3 \rfloor$.
Amplasarea camerelor: Plasăm camere de supraveghere în toate vârfurile de culoarea aleasă. Fiecare triunghi din triangulare va avea exact un vârf de această culoare, deci întregul poligon va fi supravegheat.
Aplicarea teoremei la poligonul $A_1A_2...A_{10}$
Pentru un poligon cu 10 vârfuri:

Numărul maxim de camere necesar este $\lfloor 10/3 \rfloor = 3$
Triangularea poligonului va conține 10-2 = 8 triunghiuri (nr varfuri - 2)
După colorarea vârfurilor cu 3 culori, una din culori va apărea în cel mult $\lfloor 10/3 \rfloor = 3$ vârfuri
Plasăm camere în aceste vârfuri (de exemplu, A_1, A_4, A_7)
Acest aranjament garantează că fiecare triunghi din triangulare va conține cel puțin un vârf cu cameră, asigurând astfel supravegherea întregului poligon.

Partea b) Valori convexe și vârfuri convexe în cazul simetriei
Definiții și concepte
Un vârf al unui poligon este convex dacă unghiul interior la acel vârf este mai mic de 180°. În termeni de produs vectorial, pentru trei vârfuri consecutive A_{i-1}, A_i, A_{i+1}:

Vârful A_i este convex dacă $(A_i-A_{i-1}) \times (A_{i+1}-A_i) > 0$ (pentru un poligon orientat în sens trigonometric)
În problema noastră, se menționează că punctele A_1, A_3, A_5, etc. sunt simetrice față de dreapta de ecuație y = x față de punctele A_2, A_4, etc. Această simetrie influențează convexitatea vârfurilor.

Efectul simetriei asupra convexității
Simetria față de dreapta y = x înseamnă că dacă un punct are coordonatele (a,b), punctul său simetric are coordonatele (b,a).

Pentru un poligon în care alternanța punctelor respectă această proprietate de simetrie, se poate demonstra matematic că:

Dacă poligonul este orientat în sens trigonometric, vârfurile de indici impari (A_1, A_3, etc.) vor tinde să fie convexe dacă sunt "mai departe" de dreapta y = x decât vecinii lor
Vârfurile de indici pari (A_2, A_4, etc.) vor tinde să fie convexe dacă sunt "mai aproape" de dreapta y = x decât vecinii lor
Analiza concretă a poligonului
Fără coordonatele exacte ale vârfurilor, putem oferi o analiză teoretică:

Un poligon regulat cu 10 vârfuri ar avea toate vârfurile convexe
În cazul unui poligon cu simetrie față de dreapta y = x între vârfuri alternante, convexitatea depinde de configurația specifică
Dacă poligonul este "destul de aproape" de a fi regulat, dar cu perturbări datorate simetriei, atunci majoritatea vârfurilor vor rămâne convexe
În absența informațiilor suplimentare, putem estima că aproximativ 6-8 vârfuri din 10 ar putea fi convexe, restul fiind concave (cu unghi interior > 180°).

Problema 6: Mulțime cu triangulare de exact cinci fețe
În această problemă, trebuie să identificăm și să caracterizăm mulțimile de puncte care admit o triangulare cu exact cinci fețe (triunghiuri).

Partea a) Exemplu de mulțime cu triangulare de exact cinci fețe
Relația fundamentală între numărul de fețe și puncte
Pentru o triangulare a unei mulțimi de n puncte în plan, dintre care h puncte se află pe frontiera acoperirii convexe, numărul de triunghiuri T este dat de formula:

$T = 2n - h - 2$

Această formulă poate fi derivată din formula lui Euler pentru grafuri planare:

Fie V = numărul de vârfuri = n
Fie E = numărul de muchii în triangulare
Fie F = numărul de fețe = T + 1 (incluzând și fața exterioară)
Formula lui Euler: V - E + F = 2
Într-o triangulare completă: E = 3n - h - 3 (fiecare triunghi are 3 muchii, dar muchiile de pe contur sunt numărate o singură dată)
Substituind, obținem: n - (3n - h - 3) + (T + 1) = 2, ceea ce duce la T = 2n - h - 2
Determinarea parametrilor necesari
Pentru a găsi o mulțime cu T = 5 triunghiuri în triangulare: 5 = 2n - h - 2 7 = 2n - h 2n = 7 + h n = (7 + h)/2

Valoarea minimă pentru h este 3 (un triunghi fiind cea mai simplă acoperire convexă posibilă). Substituim h = 3: n = (7 + 3)/2 = 10/2 = 5

Deci, avem nevoie de n = 5 puncte, dintre care h = 3 sunt pe frontiera acoperirii convexe.

Construcția explicită
Un exemplu concret de mulțime cu aceste proprietăți este: M = {(0,0), (4,0), (2,4), (1,1), (3,1)}

Verificăm proprietățile acestei mulțimi:

Acoperirea convexă este triunghiul format de punctele (0,0), (4,0), (2,4)
Celelalte două puncte, (1,1) și (3,1), sunt în interiorul acestui triunghi
Triangularea conține 5 triunghiuri:
[(0,0), (1,1), (2,4)]
[(0,0), (4,0), (1,1)]
[(4,0), (3,1), (1,1)]
[(4,0), (2,4), (3,1)]
[(1,1), (3,1), (2,4)]
Putem verifica corectitudinea folosind formula: T = 2n - h - 2 = 2(5) - 3 - 2 = 10 - 5 = 5 ✓

Partea b) Caracterizarea completă a mulțimilor cu triangulare de 5 fețe
O mulțime M din R^2 admite o triangulare cu exact 5 fețe dacă și numai dacă îndeplinește următoarele condiții:

Relația între numărul total de puncte și punctele de pe frontieră: 2n - h - 2 = 5, sau echivalent, 2n - h = 7

Poziția generală: Punctele trebuie să fie în poziție generală (nu există trei puncte coliniare), altfel nu se poate realiza o triangulare completă.

Configurația validă: Punctele din interior nu trebuie să fie suficient de multe pentru a permite mai multe triunghiuri sau insuficient de multe pentru a nu permite 5 triunghiuri.

Cazuri posibile
Rezolvând ecuația 2n - h = 7 pentru diferite valori ale lui h, obținem:

Pentru h = 3: n = (7 + 3)/2 = 5 puncte
Pentru h = 5: n = (7 + 5)/2 = 6 puncte
Pentru h = 7: n = (7 + 7)/2 = 7 puncte
Pentru h = 9: n = (7 + 9)/2 = 8 puncte
etc.
Însă, pentru valori h > 7, configurațiile nu pot fi realizate geometric. Un poligon convex cu h vârfuri poate fi triangulat în exact h-2 triunghiuri. Deci:

Pentru h = 3: avem nevoie de 3-2 = 1 triunghi pe frontieră și 5-1 = 4 triunghiuri în interior
Pentru h = 5: avem nevoie de 5-2 = 3 triunghiuri pe frontieră și 5-3 = 2 triunghiuri în interior
Pentru h = 7: avem nevoie de 7-2 = 5 triunghiuri pe frontieră, ceea ce epuizează cele 5 fețe necesare
Astfel, cazurile valide sunt:

n = 5, h = 3:

5 puncte în total
3 puncte formează un triunghi (acoperirea convexă)
2 puncte sunt în interiorul triunghiului
Triangularea conține 5 triunghiuri
n = 6, h = 5:

6 puncte în total
5 puncte formează un pentagon (acoperirea convexă)
1 punct este în interiorul pentagonului
Triangularea conține 5 triunghiuri
n = 7, h = 7:

7 puncte în total, toate pe frontieră
Formează un heptagon (poligon cu 7 laturi)
Triangularea conține 7-2 = 5 triunghiuri
Toate aceste configurații respectă formula T = 2n - h - 2 = 5 și reprezintă toate modalitățile posibile de a obține o triangulare cu exact 5 fețe.

Problema 7: Intersecția semiplanelor în funcție de parametrul α
În această problemă, analizăm intersecția a patru semiplane în funcție de parametrul α:

$H\alpha$: $-x + \alpha \leq 0 \Rightarrow x \geq \alpha$
$H'\alpha$: $y + \alpha - 8 \leq 0 \Rightarrow y \leq 8 - \alpha$
S_1: $-y + 2 \leq 0 \Rightarrow y \geq 2$
S_2: $2x - y - 6 \leq 0 \Rightarrow y \geq 2x - 6$
Pasul 1: Analiza semiplanelor individuale
Pentru a înțelege regiunea de intersecție, analizăm mai întâi ce reprezintă fiecare semiplan:

$H\alpha$: Reprezintă semiplnul din dreapta dreptei verticale $x = \alpha$. Poziția acestei drepte variază în funcție de α.
$H'\alpha$: Reprezintă semiplnul de sub dreapta orizontală $y = 8 - \alpha$. Poziția acestei drepte scade pe măsură ce α crește.
S_1: Reprezintă semiplnul de deasupra dreptei orizontale $y = 2$ (fix, nu depinde de α).
S_2: Reprezintă semiplnul de deasupra dreptei oblice $y = 2x - 6$ (fix, nu depinde de α).
Pasul 2: Analiza intersecției $S_1 \cap S_2$ (independentă de α)
Începem prin analizarea intersecției semiplanelor fixe S_1 și S_2:

S_1: $y \geq 2$
S_2: $y \geq 2x - 6$
Punctul de intersecție al frontierelor: $y = 2$ $y = 2x - 6 \Rightarrow 2 = 2x - 6 \Rightarrow 8 = 2x \Rightarrow x = 4$

Deci, frontierele se intersectează în punctul (4, 2).

Intersecția $S_1 \cap S_2$ este regiunea: $\{(x, y) \mid x \geq 0, y \geq 2, y \geq 2x - 6\}$

Aceasta este un semiplana delimitat de:

Dreapta orizontală $y = 2$ pentru $x \geq 4$
Dreapta oblică $y = 2x - 6$ pentru $3 \leq x \leq 4$ (unde $2x - 6 \geq 2$)
Pasul 3: Incorporarea constrângerilor $H\alpha$ și $H'\alpha$
Acum analizăm cum influențează $H\alpha$ și $H'\alpha$ (care depind de α) intersecția totală:

Constrângerea $H\alpha$: $x \geq \alpha$

Limitează regiunea la dreapta dreptei $x = \alpha$
Pentru ca intersecția să fie nevidă, avem nevoie de $x = \alpha$ să intersecteze $S_1 \cap S_2$
Deoarece punctul cel mai din stânga din $S_1 \cap S_2$ este (3, 0), avem nevoie de $\alpha \leq 4$
Constrângerea $H'\alpha$: $y \leq 8 - \alpha$

Limitează regiunea sub dreapta $y = 8 - \alpha$
Pentru ca intersecția să fie nevidă, avem nevoie de $y = 8 - \alpha$ să intersecteze $S_1 \cap S_2$
Deoarece punctul cel mai de jos din $S_1 \cap S_2$ are $y = 2$, avem nevoie de $8 - \alpha \geq 2 \Rightarrow \alpha \leq 6$
Pasul 4: Analiza detaliată a intersecției complete în funcție de α
Combinând toate constrângerile, avem:

$x \geq \alpha$

$y \leq 8 - \alpha$
$y \geq 2$
$y \geq 2x - 6$
Pentru ca aceste constrângeri să fie compatibile, avem nevoie de:

$\alpha \leq 4$ (din prima constrângere)
$\alpha \leq 6$ (din a doua și a treia constrângeri)
Deci, $\alpha \leq 4$ este condiția globală pentru ca intersecția să fie nevidă.

Calculul vârfurilor regiunii de intersecție
Pentru $\alpha \leq 4$, regiunea de intersecție $H\alpha \cap H'\alpha \cap S_1 \cap S_2$ este un poligon cu următoarele vârfuri:

Vârful la intersecția $x = \alpha$, $y = 2$: $(\alpha, 2)$
Vârful la intersecția $x = \alpha$, $y = 8 - \alpha$ (dacă $\alpha \leq 3$): $(\alpha, 8 - \alpha)$
Vârful la intersecția $y = 8 - \alpha$, $y = 2x - 6$: $8 - \alpha = 2x - 6$ $14 - \alpha = 2x$ $x = (14 - \alpha)/2 = 7 - \alpha/2$ Deci vârful este $(7 - \alpha/2, 8 - \alpha)$
Vârful la intersecția $y = 2$, $y = 2x - 6$ (independent de α): $(4, 2)$
Analiza formei regiunii în funcție de α
Pentru $\alpha > 6$:

Incompatibilitate între $y \geq 2$ și $y \leq 8 - \alpha$
Intersecția este vidă
Pentru $4 < \alpha \leq 6$:

Dreapta $x = \alpha$ nu intersectează $S_1 \cap S_2$
Intersecția este vidă
Pentru $3 < \alpha \leq 4$:

Vârfurile regiunii sunt:
$(\alpha, 2)$ - unde $x = \alpha$ intersectează $y = 2$
$(7 - \alpha/2, 8 - \alpha)$ - unde $y = 8 - \alpha$ intersectează $y = 2x - 6$
$(4, 2)$ - unde $y = 2$ intersectează $y = 2x - 6$
Regiunea este un triunghi
Pentru $0 < \alpha \leq 3$:

Vârfurile regiunii sunt:
$(\alpha, 2)$ - unde $x = \alpha$ intersectează $y = 2$
$(\alpha, 8 - \alpha)$ - unde $x = \alpha$ intersectează $y = 8 - \alpha$
$(7 - \alpha/2, 8 - \alpha)$ - unde $y = 8 - \alpha$ intersectează $y = 2x - 6$
$(4, 2)$ - unde $y = 2$ intersectează $y = 2x - 6$
Regiunea este un patrulater
Pentru $\alpha \leq 0$:

Similar cu cazul anterior, dar $x = \alpha$ nu mai intersectează regiunea $S_1 \cap S_2$
Vârfurile regiunii sunt:
$(0, 2)$ - unde $x = 0$ intersectează $y = 2$
$(0, 8 - \alpha)$ - unde $x = 0$ intersectează $y = 8 - \alpha$

(7 - α/2, 8 - α) - unde y = 8 - α intersectează y = 2x - 6
(4, 2) - unde y = 2 intersectează y = 2x - 6
Regiunea este un patrulater
Concluzie pentru intersecția semiplanelor
În funcție de parametrul α, natura intersecției $H\alpha \cap H'\alpha \cap S_1 \cap S_2$ este:

Pentru α > 6 sau α > 4: intersecția este vidă
Pentru 3 < α ≤ 4: intersecția este un triunghi
Pentru α ≤ 3: intersecția este un patrulater
Această variație a formei regiunii de intersecție ilustrează cum parametrii din definițiile semiplanelor pot influența dramatic geometria regiunii rezultate.

Problema 8: Distanța maximă între puncte și drepte
În această problemă, avem n pătrate P_1, P_2, ..., P_n și n drepte paralele cu prima bisectoare, iar trebuie să determinăm distanța maximă între un punct din oricare pătrat și oricare dreaptă.

Concepte fundamentale și formule
Formula distanței de la un punct la o dreaptă
Pentru o dreaptă cu ecuația generală $ax + by + c = 0$ și un punct $P(x_0, y_0)$, distanța de la punct la dreaptă este dată de formula:

$d(P, \text{dreaptă}) = |ax_0 + by_0 + c| / \sqrt{(a^2 + b^2)}$

Pentru o dreaptă cu ecuația $y = x + a$ (paralelă cu prima bisectoare), ecuația generală este $-x + y - a = 0$, deci $a = -1$, $b = 1$, $c = -a$. Astfel:

$d(P, \text{dreaptă}) = |-x_0 + y_0 - a| / \sqrt{((-1)^2 + 1^2)} = |y_0 - x_0 - a| / \sqrt{2}$

Proprietăți geometrice importante
Distanța maximă la un poligon convex: Distanța maximă de la o dreaptă la un poligon convex este întotdeauna atinsă la unul din vârfurile poligonului.

Direcția de maximizare: Pentru o dreaptă cu ecuația $y = x + a$, vectorul normal este $(-1, 1)/\sqrt{2}$. Distanța se maximizează în direcția acestui vector sau în direcția opusă.

Descrierea detaliată a algoritmului
Input-ul problemei
n pătrate P_1, P_2, ..., P_n, fiecare cu centrul (x_i, y_i) și latura de lungime q_i
n drepte d_1, d_2, ..., d_n cu ecuațiile $y = x + a_1$, $y = x + a_2$, ..., $y = x + a_n$
Pașii algoritmului
Pentru fiecare pătrat P_i:

Calculăm coordonatele celor 4 vârfuri ale pătratului:
$V_1 = (x_i - q_i/2, y_i - q_i/2)$ - vârful stânga-jos
$V_2 = (x_i + q_i/2, y_i - q_i/2)$ - vârful dreapta-jos
$V_3 = (x_i + q_i/2, y_i + q_i/2)$ - vârful dreapta-sus
$V_4 = (x_i - q_i/2, y_i + q_i/2)$ - vârful stânga-sus
Pentru fiecare dreaptă d_j: $y = x + a_j$:

Calculăm distanța de la fiecare vârf al fiecărui pătrat la dreaptă folosind formula: $d(V, d_j) = |y_v - x_v - a_j| / \sqrt{2}$
Pentru optimizare, calculăm doar distanțele pentru vârfurile semnificative:
Dacă $a_j \geq 0$, verificăm doar V_2 și V_4
Dacă $a_j < 0$, verificăm doar V_1 și V_3
Determinarea distanței maxime:

Păstrăm maximul global dintre toate distanțele calculate
Exemplu concret
Considerăm un pătrat P cu centrul (0,0) și latura $q = 2$, și o dreaptă d: $y = x + 1$.

Vârfurile pătratului sunt:

$V_1 = (-1, -1)$
$V_2 = (1, -1)$
$V_3 = (1, 1)$
$V_4 = (-1, 1)$
Calculăm distanțele:

$d(V_1, d) = |(-1) - (-1) - 1| / \sqrt{2} = |0 - 1| / \sqrt{2} = 1/\sqrt{2}$
$d(V_2, d) = |(-1) - 1 - 1| / \sqrt{2} = |-3| / \sqrt{2} = 3/\sqrt{2}$
$d(V_3, d) = |1 - 1 - 1| / \sqrt{2} = |-1| / \sqrt{2} = 1/\sqrt{2}$
$d(V_4, d) = |1 - (-1) - 1| / \sqrt{2} = |1| / \sqrt{2} = 1/\sqrt{2}$
Distanța maximă este $3/\sqrt{2} \approx 2.12$, atinsă la vârful V_2.

Optimizarea algoritmului
Observație cheie pentru eficiență
Pentru o dreaptă $y = x + a$ și un pătrat cu vârfurile V_1, V_2, V_3, V_4:

Când $a \geq 0$:

Pentru vârful $V_1(x-q/2, y-q/2)$: $d = |y-q/2 - (x-q/2) - a| / \sqrt{2} = |y-x-a| / \sqrt{2}$
Pentru vârful $V_3(x+q/2, y+q/2)$: $d = |y+q/2 - (x+q/2) - a| / \sqrt{2} = |y-x-a| / \sqrt{2}$
Pentru vârful $V_2(x+q/2, y-q/2)$: $d = |y-q/2 - (x+q/2) - a| / \sqrt{2} = |y-x-q-a| / \sqrt{2}$
Pentru vârful $V_4(x-q/2, y+q/2)$: $d = |y+q/2 - (x-q/2) - a| / \sqrt{2} = |y-x+q-a| / \sqrt{2}$
Observăm că pentru $a \geq 0$, $|y-x-q-a|$ este mai mic decât $|y-x+q-a|$, deci V_4 va da distanța maximă.

Similar, pentru $a < 0$, V_2 va da distanța maximă.

Această optimizare reduce numărul de calcule de la $4n^2$ la $2n^2$.

Algoritm optimizat în pseudocod

```
funcție DistanțăMaximă(pătrate, drepte):
   maxDist = 0

   pentru fiecare pătrat P cu centrul (x,y) și latura q:
      pentru fiecare dreaptă d: y = x + a:
         dacă a ≥ 0:
            // Verifică doar vârfurile V₂ și V₄
            d₂ = |y-q/2 - (x+q/2) - a| / √2
            d₄ = |y+q/2 - (x-q/2) - a| / √2
            dist = max(d₂, d₄)
         altfel:
            // Verifică doar vârfurile V₁ și V₃
            d₁ = |y-q/2 - (x-q/2) - a| / √2
            d₃ = |y+q/2 - (x+q/2) - a| / √2
```

dist = max(d_1, d_3)

maxDist = max(maxDist, dist)

returnează maxDist

Complexitatea algoritmului

Algoritmul are complexitatea $O(n^2)$ deoarece verificăm fiecare pereche (pătrat, dreaptă). Această complexitate este optimă pentru problema generală, deoarece trebuie să considerăm toate perechile posibile.

În practică, se pot aplica și alte optimizări, cum ar fi partajarea spațiului sau utilizarea structurilor de date speciale pentru interogări de proximitate, dar acestea nu reduc complexitatea asimptotică în cazul general.

MODEL 3 (2020-2021)

Problema 1 (30p): Distribuția cadourilor de Crăciun

Context: O clasă de m copii primește n cadouri cu valori val_1, val_2, ..., val□ de la Moș Crăciun. Trebuie să distribuim cadourile pentru a maximiza valoarea minimă primită de un copil, cu restricția că val(i) ≤ Val/(2m) pentru orice cadou i (unde Val este valoarea totală a cadourilor).

a) Descrieți un algoritm 1/2 aproximativ pentru problema cadourilor în complexitate O(n log m) (10p)

Algoritm:

Sortăm cadourile în ordine descrescătoare a valorii: $val_1 \geq val_2 \geq ... \geq$ val□ (O(n log n))

Menținem un min-heap cu valorile totale curente pentru fiecare copil, toate inițializate cu 0 (O(m))

Procesăm cadourile în ordine:

Extragem copilul cu suma minimă curentă (O(log m))

Îi asignăm cadoul curent (O(1))

Reinserăm copilul în heap cu noua sumă (O(log m))

Repetăm pasul 3 pentru toate cadourile (n iterații)

Complexitatea totală: O(n log m) dacă n > m, sau O(n log n) dacă m > n

Code

```
Algoritm DistribuieCadouri(val[1..n], m):
   // val[i] = valoarea cadoului i
   // m = numărul de copii

   Sortează val[] în ordine descrescătoare  // O(n log n)

   // Inițializează sumele pentru fiecare copil
   suma[1..m] = 0

   // Construiește min-heap cu sumele copiilor
   H = ConstruiesteMinHeap(suma[])  // O(m)

   // Distribuie cadourile
   Pentru i = 1 până la n:
      k = ExtrageMininm(H)  // Extrage copilul cu suma minimă
      suma[k] = suma[k] + val[i]
      InsereazaInHeap(H, k, suma[k])

   // Returnează valoarea minimă
   Returnează ExtrageMininm(H).valoare
```

b) Fie k acel copil pentru care ALG = W(K). Fie q ultimul cadou primit de un copil oarecare Q (q ≠ k). Care este relația între W(K) și W(Q) - val(q)? (5p)

Soluție: Copilul K are suma minimă la finalul algoritmului (ALG = W(K)).

Fie Q un alt copil și q ultimul cadou primit de Q. Înainte ca Q să primească cadoul q, suma sa era W(Q) - val(q).

Relația: $W(K) \geq W(Q) - val(q)$

Justificare: În momentul când copilul Q a primit cadoul q, algoritmul a ales copilul cu suma minimă curentă. Deci $W(Q) - val(q) \leq W(K')$ pentru orice alt copil K' în acel moment. Deoarece după aceea copilul K primește eventual alte cadouri sau rămâne neschimbat, $W(K) \geq W(K')$ la momentul respectiv. Prin urmare, $W(K) \geq W(Q) - val(q)$.

c) Pe baza punctului b) arătați că $ALG \geq Val/(2m)$ (5p)
Soluție: Știm că $W(K) \geq W(Q) - val(q)$ pentru orice copil Q și ultimul său cadou q.

Sumând această relație pentru toți copiii $Q \neq K$: $W(K) \geq W(Q) - val(q)$ pentru fiecare $Q \neq K$

Deci: $(m-1) \cdot W(K) \geq \Sigma(W(Q) - val(q))$ pentru toți $Q \neq K$

Observăm că:

Σ W(Q) pentru toți Q = Val - W(K) (suma totală minus suma lui K)
Σ val(q) pentru ultimele cadouri $\leq Val/2$ (folosind restricția $val(i) \leq Val/(2m)$)
Deci: $(m-1) \cdot W(K) \geq (Val - W(K)) - Val/2 \Rightarrow (m-1) \cdot W(K) \geq Val/2 - W(K) \Rightarrow m \cdot W(K) \geq Val/2 \Rightarrow W(K) = ALG \geq Val/(2m)$

d) Demonstrați că algoritmul descris la punctul a) este 1/2 aproximativ (5p)
Soluție: Trebuie să demonstrăm că $ALG \geq OPT/2$, unde OPT este soluția optimă.

Observăm că $OPT \leq Val/m$, deoarece chiar și în distribuția perfectă, fiecare copil primește în medie Val/m.

Din punctul c), știm că $ALG \geq Val/(2m)$.

Deci: $ALG \geq Val/(2m) \geq (Val/m)/2 \geq OPT/2$

Prin urmare, algoritmul este 1/2-aproximativ.

e) Dați un exemplu format din minimum 2 copii și 4 cadouri pentru care algoritmul nu găsește soluția optimă (5p)
Exemplu:

2 copii
4 cadouri cu valorile: [10, 8, 7, 5]
Valoarea totală Val = 30
Execuția algoritmului:

Sortăm cadourile: [10, 8, 7, 5]
Primul cadou (10) merge la primul copil: [10, 0]
Al doilea cadou (8) merge la al doilea copil: [10, 8]

Al treilea cadou (7) merge la al doilea copil: [10, 15]
Al patrulea cadou (5) merge la primul copil: [15, 15]
Rezultatul algoritmului: ALG = 15 (minimul între 15 și 15)

Soluția optimă:

Primul copil primește cadourile 10 și 7: suma 17
Al doilea copil primește cadourile 8 și 5: suma 13
OPT = 13 (minimul între 17 și 13)

Algoritmul nu găsește soluția optimă în acest caz.

Problema 2 (5p): Testul de orientare
Fie punctul A = (3,2) ∈ R^2. Dați exemplu de puncte B,C ∈ R^2 astfel ca C să fie în stânga muchiei orientate AB și calculați valoarea determinantului care apare în testul de orientare.

Soluție: Pentru ca un punct C să fie în stânga muchiei orientate AB, determinantul trebuie să fie pozitiv:

det = (xB - xA)(yC - yA) - (yB - yA)(xC - xA)

Alegem:

A = (3,2) (dat)
B = (5,4) (ales)
C = (2,5) (ales)
Calculăm determinantul: det = (5 - 3)(5 - 2) - (4 - 2)(2 - 3) = 2 · 3 - 2 · (-1) = 6 + 2 = 8

Determinantul este pozitiv (8 > 0), deci C este în stânga muchiei orientate AB.

Problema 3 (10p): Mulțime de puncte pentru Graham's scan

Dați un exemplu de mulțime M cu 9 elemente din planul R^2 astfel ca (i) $(2,3) \in M$, (ii) la final, L are 3 elemente, unde L este lista vârfurilor care determină marginea inferioară a frontierei acoperirii convexe a lui M, obținută pe parcursul Graham's scan, varianta Andrew, (iii) numărul maxim de elemente ale lui L, pe parcursul aplicării algoritmului, este egal cu 6. Justificați!

Soluție: Trebuie să construim un set M de 9 puncte care să satisfacă toate condițiile.

Alegem: M = {(2,3), (0,1), (1,0), (2,0), (3,0), (4,0), (5,0), (6,1), (3,4)}

Justificare:

$(2,3) \in M$ (condiția i)

Pentru a calcula marginea inferioară a acoperirii convexe:

Sortăm punctele după x: (0,1), (1,0), (2,0), (2,3), (3,0), (3,4), (4,0), (5,0), (6,1)
Construim lanțul inferior, examinând punctele de la stânga la dreapta:
L = [(0,1)]
Adăugăm (1,0): L = [(0,1), (1,0)]
Adăugăm (2,0): L = [(0,1), (1,0), (2,0)]
(2,3) este deasupra, nu îl adăugăm
Adăugăm (3,0): L = [(0,1), (1,0), (2,0), (3,0)]
(3,4) este deasupra, nu îl adăugăm
Adăugăm (4,0): L = [(0,1), (1,0), (2,0), (3,0), (4,0)]
Adăugăm (5,0): L = [(0,1), (1,0), (2,0), (3,0), (4,0), (5,0)] (aici L atinge maximul de 6 elemente)
Adăugăm (6,1): verificăm coliniaritatea ultimelor 3 puncte și eliminăm punctele intermediare
L final = [(0,1), (1,0), (6,1)] (condiția ii)
Numărul maxim de elemente din L pe parcursul algoritmului este 6, atins după adăugarea lui (5,0) (condiția iii)

Problema 4 (10p): Numărul maxim de vârfuri pentru o hartă trapezoidală
Precizați care este numărul maxim de vârfuri pe care îl poate avea o hartă trapezoidală asociată unei mulțimi de n segmente în poziție generală, inclusiv în interiorul unui dreptunghi D cu laturile paralele cu axele de coordonate. Dați un exemplu în care acest maxim este atins și un exemplu în care acest lucru nu se întâmplă.

Soluție: Pentru o hartă trapezoidală într-un dreptunghi D:

Dreptunghiul D contribuie cu 4 vârfuri (colțurile sale)
Fiecare segment poate genera până la 4 vârfuri noi (2 la fiecare capăt, când se trasează linii verticale)
Fiecare intersecție între 2 segmente generează un vârf
Numărul maxim de vârfuri în poziție generală (fără 3 segmente concurente și fără segmente paralele cu axele) este:

4 (colțurile dreptunghiului) +
4n (4 vârfuri pentru fiecare dintre cele n segmente) +
C(n,2) (numărul maxim de intersecții între n segmente) = n(n-1)/2
Număr maxim de vârfuri = $4 + 4n + n(n-1)/2 = 4 + 4n + n^2/2 - n/2 = 4 + 7n/2 + n^2/2$

Exemplu unde maximul este atins:

Un dreptunghi D
n segmente orientate astfel încât:
Niciun segment nu este paralel cu axele
Fiecare segment intersectează toate celelalte segmente
Nicio linie verticală nu trece prin mai multe vârfuri sau intersecții
Niciun capăt de segment nu se află pe aceeași verticală cu alt capăt sau cu o intersecție
Exemplu unde maximul nu este atins:

Un dreptunghi D
n segmente orizontale care nu se intersectează
În acest caz, numărul de vârfuri este doar 4 + 2n (4 colțuri ale dreptunghiului și 2 vârfuri pentru fiecare segment, când liniile verticale întâlnesc segmentele)

Problema 5 (10p): Intersecția semiplanelor în funcție de parametru
Fie semiplanele H_1, H_2, H_5. Semiplanele H_1 și H_2 au dreptele suport paralele cu axa Ox (descrieți explicit aceste semiplane!). Semiplanul H_3 este dat de o inecuație de forma H_3: $x + y - \alpha \geq 0$, unde $\alpha \in R$ este un parametru. Discutați, în funcție de parametrul α, natura intersecției $H_1 \cap H_2 \cap H_3$.

Soluție: Definim explicit semiplanele H_1 și H_2:

H_1: $y \geq a$ (semiplanul de deasupra dreptei $y = a$)
H_2: $y \leq b$ (semiplanul de sub dreapta $y = b$) unde $a < b$ (pentru ca intersecția să fie nevidă)
H_3: $x + y - \alpha \geq 0 \Longrightarrow y \geq \alpha - x$ (semiplanul deasupra dreptei $y = \alpha - x$)

Intersecția $H_1 \cap H_2$ este banda orizontală $\{(x,y) \mid a \leq y \leq b\}$.

Analiza în funcție de α:

Când $\alpha \leq a$:

Dreapta $y = \alpha - x$ se află sub dreapta $y = a$ pentru orice x
H_3 include complet semiplanul H_1
Intersecția $H_1 \cap H_2 \cap H_3 = H_1 \cap H_2 = \{(x,y) \mid a \leq y \leq b\}$
Este o bandă orizontală infinită
Când $a < \alpha < a + b$:

Dreapta $y = \alpha - x$ intersectează banda $H_1 \cap H_2$
Intersecția $H_1 \cap H_2 \cap H_3 = \{(x,y) \mid a \leq y \leq b, x + y \geq \alpha\}$
Este un poligon semimărginit (un "colț" infinit în direcția pozitivă a axei x)
Când $\alpha = a + b$:

Dreapta $y = \alpha - x$ intersectează banda exact în punctul $(b-a, b)$
Intersecția $H_1 \cap H_2 \cap H_3$ devine un triunghi
Când $\alpha > a + b$:

Dreapta $y = \alpha - x$ se află deasupra dreptei $y = b$ pentru $x < \alpha - b$
Intersecția $H_1 \cap H_2 \cap H_3 = \{(x,y) \mid a \leq y \leq b, x + y \geq \alpha\}$
Este un poligon mărginit (un trapez sau triunghi)
Când $\alpha > b + L$ (unde L este o valoare mare):

Dreapta $y = \alpha - x$ se află complet deasupra benzii $H_1 \cap H_2$
Intersecția $H_1 \cap H_2 \cap H_3 = \varnothing$ (mulțimea vidă)
În concluzie, pe măsură ce α crește, intersecția se transformă dintr-o bandă infinită într-un poligon semimărginit, apoi într-un poligon mărginit, și în cele din urmă devine vidă.

Problema 6

a) (10p) Dați exemplu de mulțime M_1 din R^2 astfel ca M_1 să admită o triangulare ce conține 4 triunghiuri și 8 muchii.

Soluție: Pentru o triangulare cu 4 triunghiuri și 8 muchii, folosim formula lui Euler pentru grafuri planare: V - E + F = 2

Unde:

V = numărul de vârfuri (puncte)
E = numărul de muchii = 8 (dat)
F = numărul de fețe = 4 + 1 = 5 (4 triunghiuri + fața exterioară)
Substituind: V - 8 + 5 = 2 V = 5

Deci, avem nevoie de 5 puncte.

Exemplu de mulțime M_1: M_1 = {(0,0), (4,0), (4,4), (0,4), (2,2)}

Acest set formează un pătrat cu un punct în centru. Triangularea rezultată va avea:

5 puncte (cele 4 colțuri ale pătratului plus punctul central)
8 muchii (cele 4 laturi ale pătratului plus 4 muchii de la punctul central la colțuri)
4 triunghiuri (fiecare triunghi format de punctul central cu 2 colțuri consecutive)

b) (10p) Considerăm mulțimea M_1 aleasă la punctul a). Dați exemplu de mulțime M_2 din R^2 formată din 3 puncte, astfel ca diagrama Voronoi asociată mulțimii $M_1 \cup M_2$ să aibă exact 3 muchii de tip semidreaptă.

Soluție: Diagrama Voronoi a mulțimii M_1 are 5 celule, una pentru fiecare punct. Pentru ca diagrama Voronoi a $M_1 \cup M_2$ să aibă exact 3 muchii de tip semidreaptă, trebuie să alegem M_2 astfel încât majoritatea muchiilor Voronoi să fie segmente finite.

Alegem M_2 = {(-2,-2), (6,-2), (2,8)}

Aceste 3 puncte sunt plasate strategic:

(-2,-2) este în exteriorul pătratului, la sud-vest
(6,-2) este în exteriorul pătratului, la sud-est
(2,8) este în exteriorul pătratului, la nord
Cu această configurație:

Fiecare punct din M_2 va avea o celulă Voronoi care se extinde la infinit
Muchiile Voronoi care separă celulele punctelor din M_2 vor fi semidrepte
Vor exista exact 3 muchii de tip semidreaptă: una între (-2,-2) și (6,-2), una între (-2,-2) și (2,8), și una între (6,-2) și (2,8)
Restul muchiilor Voronoi vor fi segmente finite

Problema 7 (5p): Algoritm eficient pentru determinarea unui poligon convex
Date n puncte în plan, scrieți un algoritm cât mai eficient (analizați complexitatea!) care să determine un poligon care are toate aceste puncte ca vârfuri. Exemplificați!

Soluție: Problema cere determinarea acoperirii convexe a mulțimii de puncte, care este cel mai mic poligon convex ce conține toate punctele date.

Algoritm (Graham's scan - varianta Andrew):

Code
```
AlgoritmAcoperireConvexa(puncte[1..n]):
   // Sortăm punctele după coordonata x (și după y în caz de egalitate)
   SorteazaPuncte(puncte) // O(n log n)

   // Construim lanțul superior
   LantSuperior = []
   Pentru i = 1 până la n:
      // Eliminăm punctele care nu formează o cotitură la dreapta
      Cât timp LantSuperior.lungime ≥ 2 și
            Orientare(LantSuperior[ultimul-1], LantSuperior[ultimul], puncte[i]) ≤ 0:
         EliminăUltimulElement(LantSuperior)
      AdaugăLaFinal(LantSuperior, puncte[i])

   // Construim lanțul inferior
   LantInferior = []
   Pentru i = n până la 1 (descrescător):
      // Eliminăm punctele care nu formează o cotitură la dreapta
      Cât timp LantInferior.lungime ≥ 2 și
            Orientare(LantInferior[ultimul-1], LantInferior[ultimul], puncte[i]) ≤ 0:
         EliminăUltimulElement(LantInferior)
      AdaugăLaFinal(LantInferior, puncte[i])

   // Eliminăm primul și ultimul punct din lanțul inferior (duplicate)
   EliminăPrimulElement(LantInferior)
   EliminăUltimulElement(LantInferior)

   // Combinăm lanțurile
   AcoperireConvexa = LantSuperior + LantInferior

   Returnează AcoperireConvexa
```
Complexitate:

Sortarea: O(n log n)
Construcția lanțurilor: O(n) (fiecare punct este adăugat și eliminat cel mult o dată)
Complexitate totală: O(n log n)
Exemplu: Pentru punctele {(0,0), (1,0), (2,1), (1,2), (0,1)}:

Sortăm: {(0,0), (0,1), (1,0), (1,2), (2,1)}
Construim lanțul superior: {(0,0), (0,1), (1,2), (2,1)}
Construim lanțul inferior: {(2,1), (1,0), (0,0)}
Eliminăm duplicatele: lanțul inferior devine {(1,0)}
Combinăm: {(0,0), (0,1), (1,2), (2,1), (1,0)}
Aceasta este acoperirea convexă și reprezintă poligonul cerut.

Algoritmul are complexitate optimă O(n log n), care este limita inferioară pentru determinarea acoperirii convexe (reducerea la problema sortării).