Examen Algoritmi Avansati

Subiect 1
a.

Pentru a transoforma problema intr-o problema de programare liniara pe numere intregi, o sa introduc cateva variabile:

1. ALG, o variabila care dorim sa reprezinte raspunsul
2. A[i][q], care dorim sa reprezinte ca task-ul i este asignat masinii q.

Instanta de programare liniara este urmatoarea (ILP-urile):

1. Dorim sa maximizam -ALG (echivalent cu a minimiza ALG, fix ceea ce ne dorim).
2. A[i][q] este in multimea { 0, 1 } (fie punem un task pe o masina fie nu il punem).
3. A[i][X[i]] + A[i][Y[i]] >= 1 pentru oricare i (altfel spus, fiecare task trebuie sa fie pus pe cel putin una din cele doua masini care pot sa il proceseze.
4. Suma(A[i][q] * T[i]) <= ALG pentru fiecare q (altfel spus, fiecare masina trebuie sa isi termine toata treaba in mai putin de ALG.

OBS: Observam ca nimic nu impiedica un task sa fie pus pe ambele masini care stiu sa il proceseze, sau chiar, in plus de una dintre masinile care stiu sa il proceseze, pe o masina care nu il poate procesa. Totusi, genul asta de stiutatii sunt situatii de "optimizare pesimista", adica fiind situatii ne-optime fie nu influenteaza raspunsul fie nu sunt niciodata considerate.

Asadar, algoritmul descris mai sus reprezinta o codificare corecta a problemei cu un algoritm de optimizare liniara de numere intregi.

Relaxarea problemei consista in a permite un task sa fie "distribuit" pe cele doua masini care il suporta (de ex face 1/3 din munca pe primul aparat, si 2/3 pe al doilea).

Variabilele folosite sunt aceleasi, si noile restrictii sunt:

Instanta de programare liniara este urmatoarea (LP-urile):

1. Dorim sa maximizam -ALG (echivalent cu a minimiza ALG, fix ceea ce ne dorim).
2. A[i][q] >= 0
3. A[i][X[i]] + A[i][Y[i]] >= 1 pentru oricare i (altfel spus, fiecare task trebuie suma a cat este procesat pe cele doua masini sa fie cel putin 1 (i.e. e facut complet).
4. Suma(A[i][q] * T[i]) <= ALG (altfel spus, fiecare masina trebuie sa isi termine toata treaba in mai putin de ALG.

Din nou, observam ca situatiile in care un task este procesat mai mult decat necesar nu sunt optime, deci nu vor influenta raspunsul.

b.

Pentru a gasi un algoritm 2-aproximativ, observam ca solutia problemei de programare liniara pe intregi este optima (nu facem nicio presupunere despre input, si avem ca restrictii minimul necesar ca solutia sa fie corecta).

Asadar, afirmam ca solutia instantei de programare liniara pe numere intregi este chiar OPT.
Cum programarea liniara relaxata este la fel sau strict mai usoara, putem presupune ca output-ul programului relaxat, pe care il notam cu RELAX, este mai mic sau egal cu OPT:

RELAX <= OPT

Pentru a gasi o solutie 2-aproximativa, presupunem urmatorul algoritm:

1. Consideram A[i][q] ca fiind matricea de asocieri de taskuri pe masini gasita de instanta de programare liniara (rezolvata cu simplex).
2. Ne construim o alta matrice booleana (0 sau 1) B, astfel:

B[i][q] = (1 if A[i][q] >= 0.5 else 0).

Claim: Matricea B reprezinta o solutie pentru o asignare de timp cel mult 2 * RELAX <= 2 * OPT.

Demonstratie:

1. Este o solutie valida:
   Fie un task i. Stim ca A[i][X[i]] + A[i][Y[i]] >= 1. Din principiul cutiei, cel putin una din cele doua elemente o sa fie cel putin 0.5

Asadar, conform pasului 2, cel putin unul dintre cele doua elemente corespunzatoare din B va fi 1.

2. Este o solutie 2-aproximativa:

Observam ca pentru fiecare masina, fiecare load asociat ei pateste doua lucruri: fie este sters complet, fie este inmultit cu cel mult 2 (worst case de la 0.5 la 1).

Asadar, algoritmul descris rezolva problema liniara relaxata, si dupa obtine o solutie de cel mult de doua ori mai proasta determinata de asocierile din B.

Asdar:

ALG <= 2 * RELAX <= 2 * OPT.

OBS:
Pe parcursul algoritmilor descrisi mai sus, am presupus ca nu este o problema daca o masina incearca sa proceseze un task incompatibil (in plus de masina corecta - de exemplu doua masini incearca sa proceseze acelasi task).

Daca este totusi o problema, atunci pot fi adaugate restrictii de tipul:

A[i][k] <= 0, A[i][k] >= 0 pentru k masina incompatibila cu taskul i.

Subiect 2

a.

Pentru a rezolva acest subiect, avem nevoie de cateva elemente de probabilitate:

Daca avem N obiecte, fiecare cu probabilitatea P1, P2, ... PN sa ajunga intregi, probabilitatea ca toate sa ajunga intregi, presupunand ca sunt probabilitati independente (desi in practica nu sunt, se spag pentru ca baga camionul un stop brusc), este P1 * P2 * P3 * ... * PN.

Observam de asemenea ca pentru a decide o asignare a obiectelor luate, este suficient sa avem un vector de lungime N de booleeni (daca luam sau nu un obiect).

Asadar, fiecare individ va reprezenta o submultime de obiecte luate. Il putem encoda cu urmatorul cromozom:

Obiecte luate = { O1, O2, ..., Ok }
chromozom = [v1, v2, ..., vN],

unde vi este 1 daca i este in obiectele luate, si 0 altfel.

Cromozomul asta este caracterizat de:

1. Lungimea N
2. Elementul i poate fi:
   0 daca individul nu ia obiectul i
   1 daca individul ia obiectul i.

b.

Pentru a descrie o functie de fitness, propun urmatoarele idei:

1. Nu dorim niciodata un individ care are proabilitatea de-a aduce continutul intact mai mica de P.
2. Daca avem de ales intre doi indivizi, ambii repectand punctul 1., il dorim pe acela cu suma valorilor transportate maxima.

De fapt, problema este similara cu cea a rucsacului cu valori, unde fiecare probabilitate se inmulteste. Cele doua probleme pot fi facute analog daca logaritmam toate probabilitatile (a.i. ele sa se adune in loc sa se inmulteasca.

Formal, avem:

FITNESS([v1, v2, ..., vn]) =
0 daca prob(v1) * prob(v2) * ... * prob(vn) < P
v1 * val(1) + v2 * val(2) + ... + vn * val(n)
daca nu.

Observam ca fitness este bine-definita, nenegativa, si egala cu 0 numai in cazul in care un individ nu este acceptat.

Subiect 3.

$A = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$

Îl aleg pe $B = \begin{pmatrix} 6 \\ 7 \\ 9 \end{pmatrix}$.

Presupun că doresc ca $\pi(A, C, B) = -2$

$\Leftrightarrow \quad AC = -2 \cdot CB$

$\Leftrightarrow \quad C - A = -2(B - C)$

$\Leftrightarrow \quad C - A = -2B + 2C$

$\Leftrightarrow \quad C = 2B - A$

$\Leftrightarrow \quad C = \begin{pmatrix} 12 & -1 \\ 14 & -2 \\ 18 & -4 \end{pmatrix} = \begin{pmatrix} 11 \\ 12 \\ 14 \end{pmatrix}$

$A = \begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}$

$B = \begin{pmatrix} 6 \\ 7 \\ 9 \end{pmatrix}$

$C = \begin{pmatrix} 11 \\ 12 \\ 14 \end{pmatrix}$

$\begin{vmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 4 & 9 & 14 \end{vmatrix} = 0$, deci punctele sunt coliniare.

Subiect 4.

$|M| = N = 7 \Rightarrow 2N - K - 2 = 9$

$\Leftrightarrow 14 - K - 2 = 9$

$\Leftrightarrow K = 3$ noduri pe înf. lui M

$A = \begin{pmatrix} -7 \\ -1 \end{pmatrix}$ $F = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

$B = \begin{pmatrix} 10 \\ -1 \end{pmatrix}$ $G = \begin{pmatrix} 1 \\ 8 \end{pmatrix}$

$C = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$

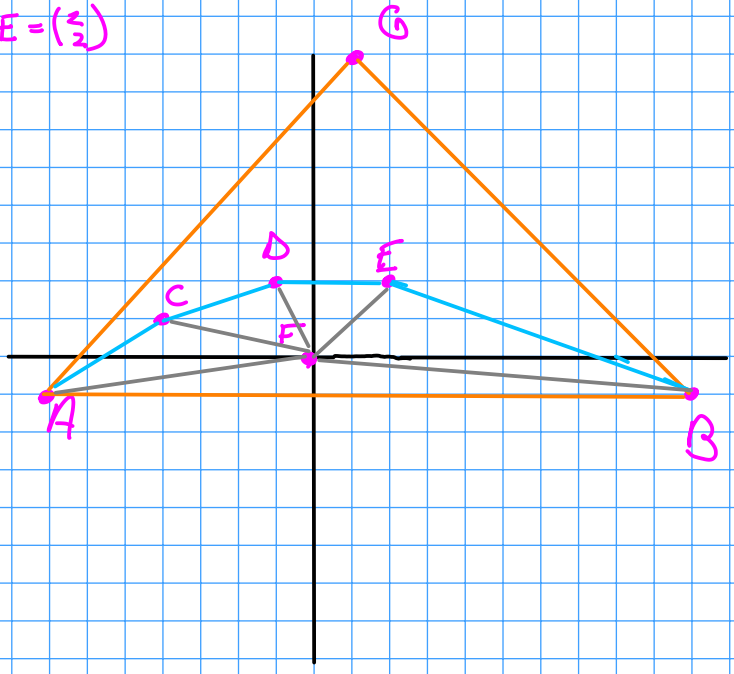
$D = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

$E = \begin{pmatrix} 3 \\ 2 \end{pmatrix}$

$2(N-1) - K' - 2 = 5$

$2 \cdot 7 - 2 - K' - 2 = 5$

$K' = 14 - 2 - 2 - 5 = 5$ noduri pe înf lui $M \setminus \{G\}$



muchii $= 3N - K - 3$

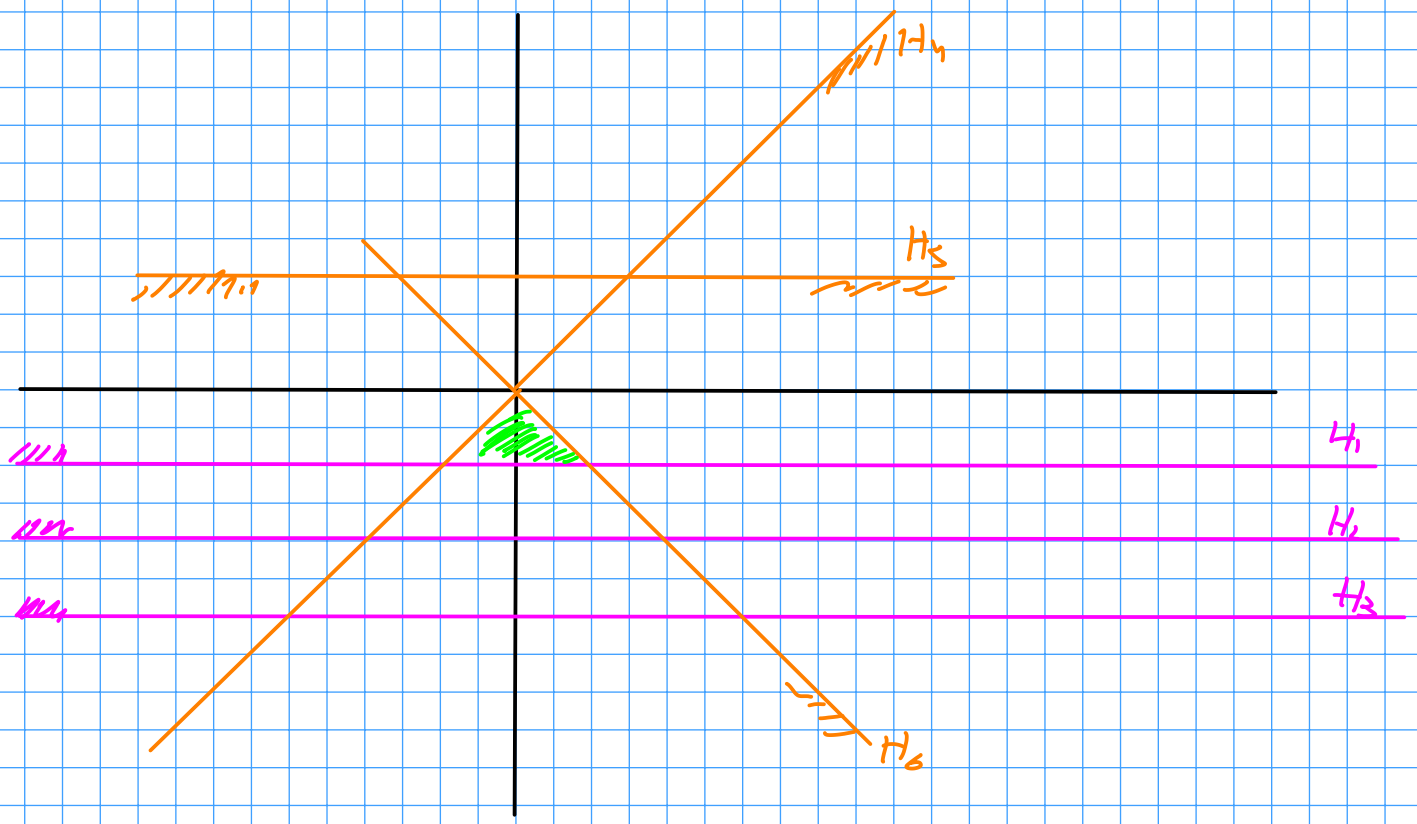
$\Rightarrow$ muchii$_M = 21 - 3 - 3 = 15$

muchii$_{M \setminus \{G\}} = 18 - 5 - 3 = 10$

Pentru a determina # de puncte de pe înf. am folosit formula din curs

Considerând și fața exterioară, prima triunghiulare are $9+1 = 10$ fețe, a doua triunghiulare are $5+1 = 6$ fețe.

Subiect 5



$H_1: -y-2 \leq 0$ $\quad H_4: -x+y \leq 0$

$H_2: -y-4 \leq 0$ $\quad H_5: x+y \leq 0$

$H_3: -y-6 \leq 0$ $\quad H_6: y-3 \leq 0$

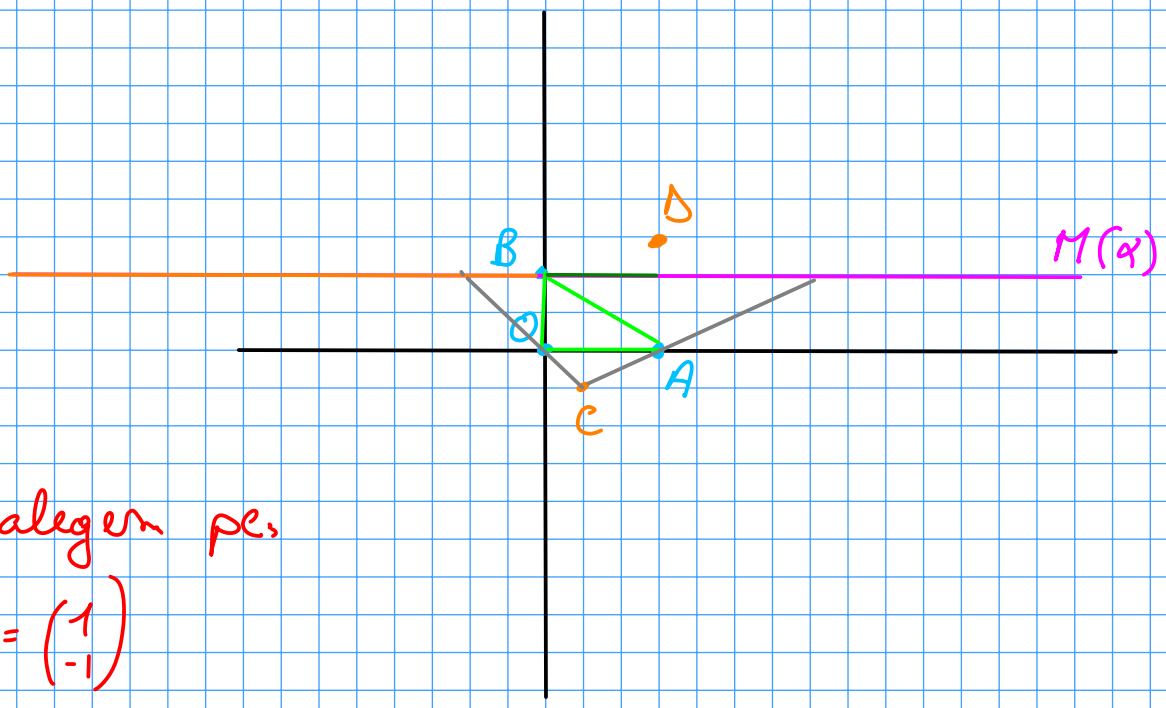
Intersecția este triunghiul cu vârfurile:

$\Delta\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}2\\-2\end{pmatrix}, \begin{pmatrix}-2\\-2\end{pmatrix}\right)$

H_1, H_2, H_3 sunt semiplane superioare

H_4, H_5 și H_6 sunt semiplane inferioare.

Subiect 6.



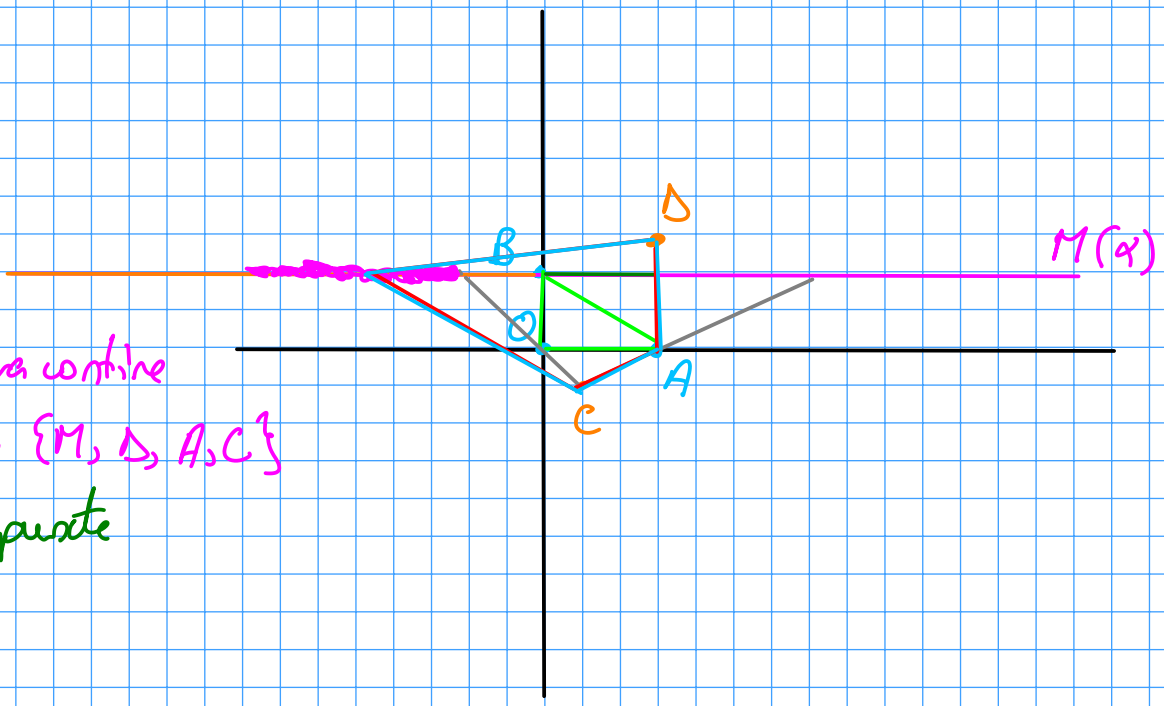
Il alegem pe:

$C = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

$D = \begin{pmatrix} 3 \\ 3 \end{pmatrix}$

Caz I

$\alpha < -2$

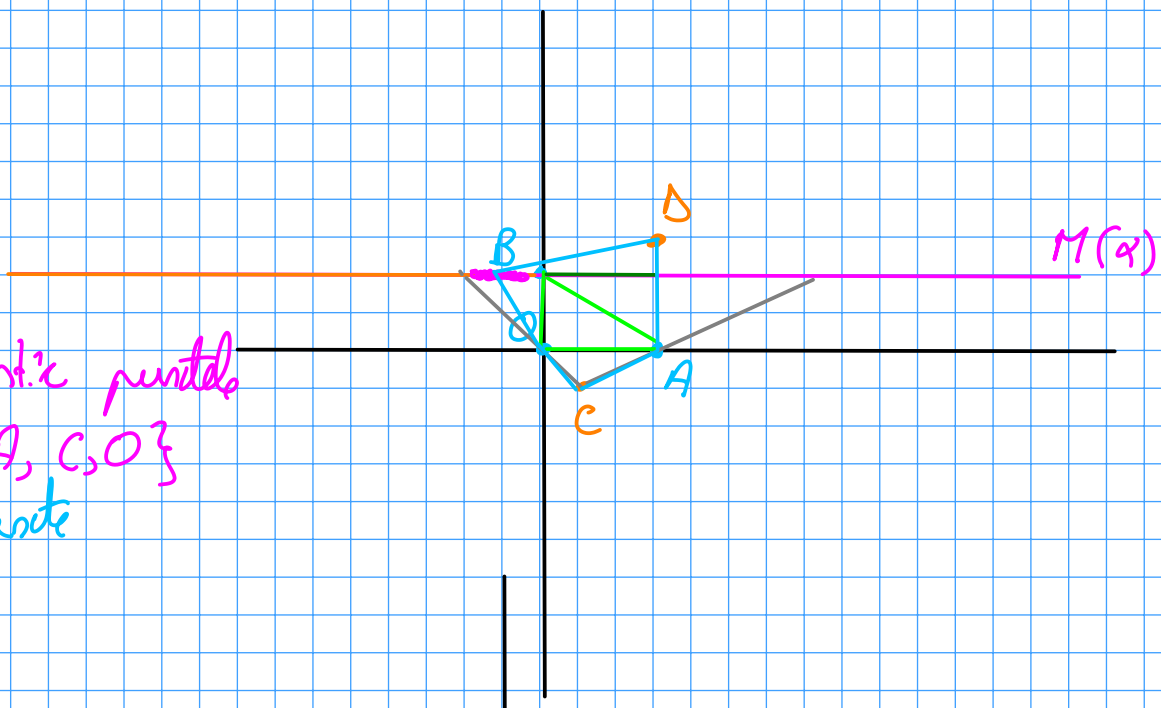


$\Rightarrow$ frontiera contine punctele $\{M, D, A, C\}$

4 puncte

Caz II

$-2 \leq \alpha \leq 0$



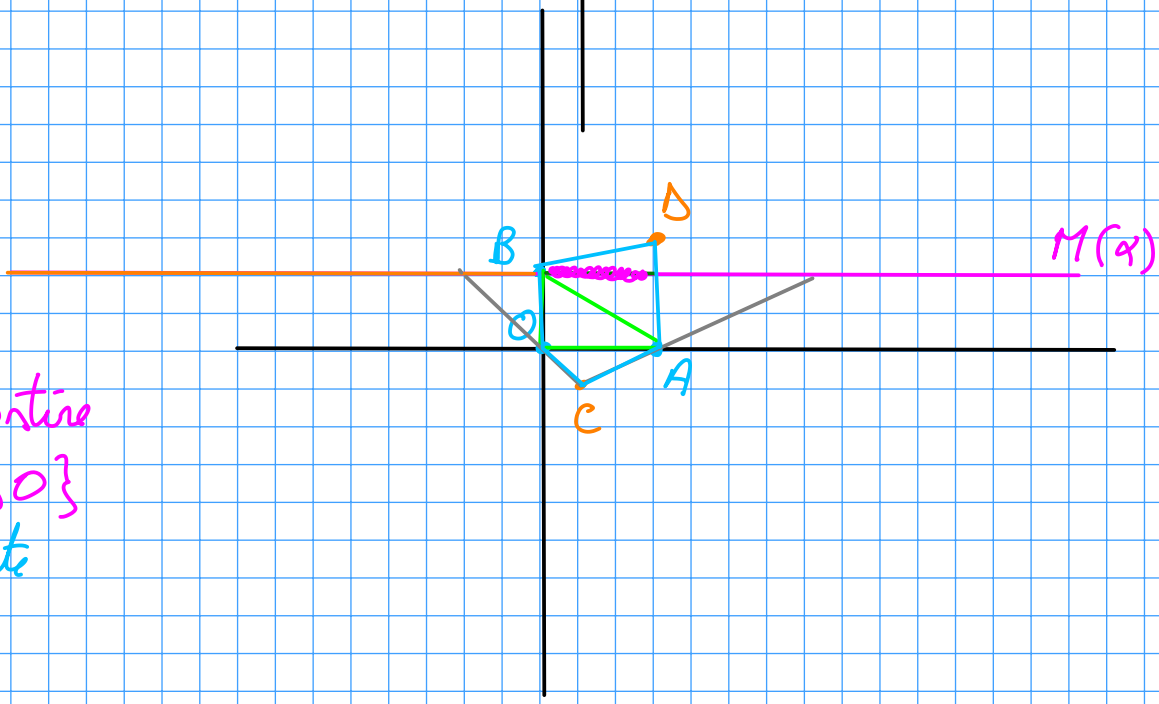
⇒ Frontiera conține punctele

$\{M, D, A, C, O\}$

5 puncte

Caz III

$0 < \alpha < 3$



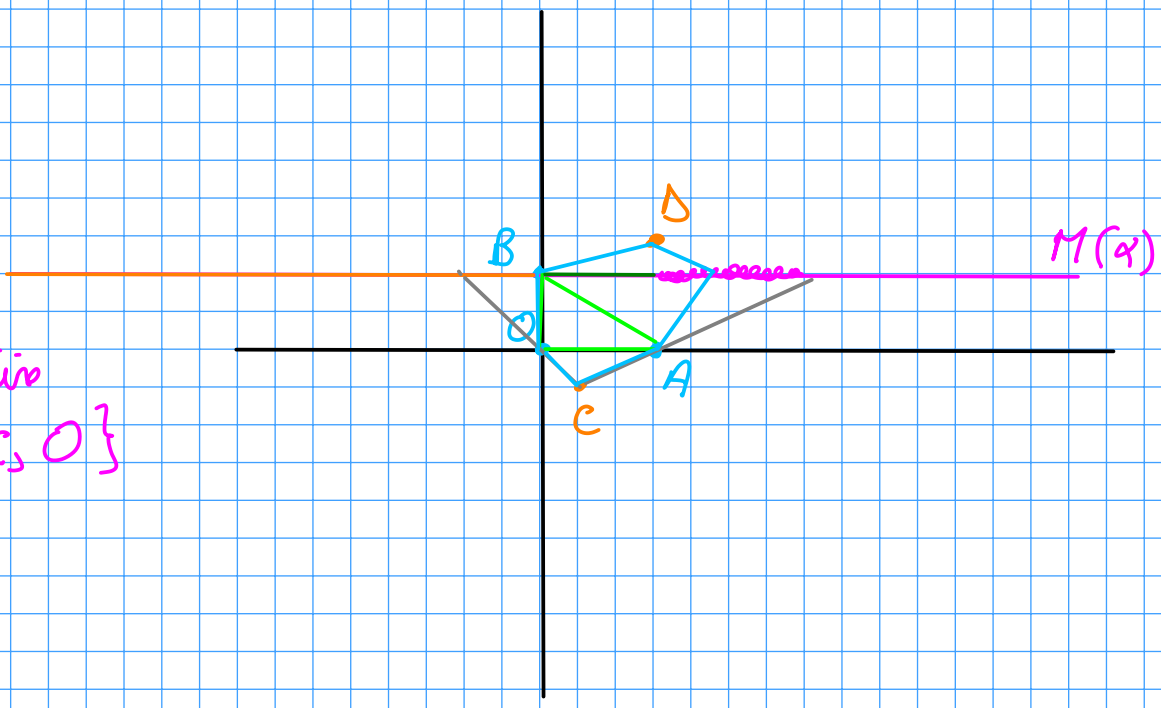
Frontiera conține

$\{B, D, A, C, O\}$

5 puncte

Caz IV

$3 \leq \alpha \leq 7$



Frontiera conține

$\{B, D, M, A, C, O\}$

6 puncte

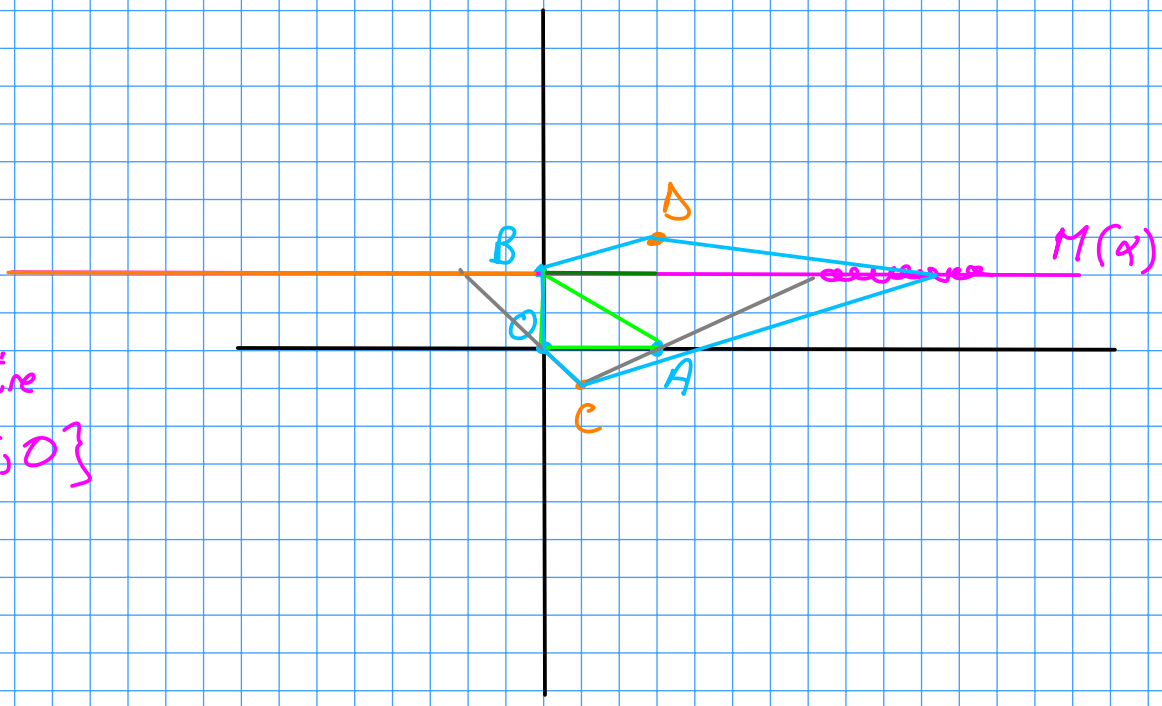
Caz V

$\pi < \alpha$

Frontiera conține

$\{B, D, M, C, O\}$

5 puncte



Când $\alpha = 0$, M și B sunt confundate, și' Ce-am tratat ca un singur punct.

Subiect 7.

a.

Pare 8 vârfuri

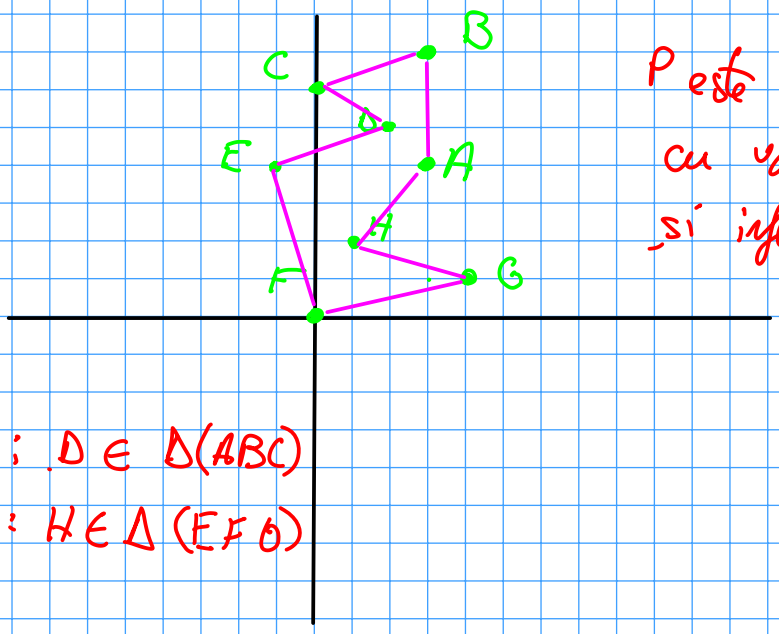
$\binom{B}{4} \in P$

B nu e principal : $D \in \Delta(ABC)$

F nu e principal : $H \in \Delta(EFD)$

B, F convexe

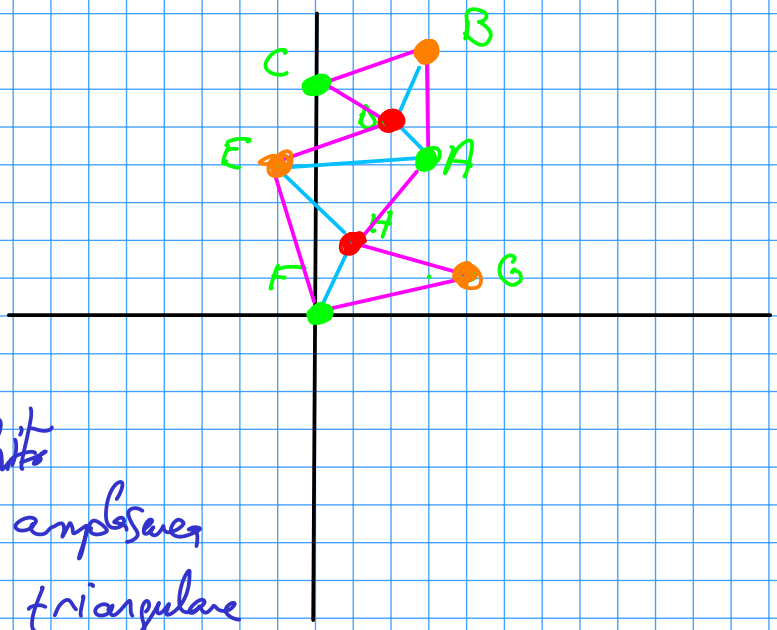
D, H concave



P este y-monoton, cu vârf superior în B și inferior în F.

b.

Am triangulat poligonul, și l-am 3-colorat. Culoarea cea mai puțin întâlnită este roșu, așa că amplasez produsă de această triangulare este în mulțimea $\{D, H\}$



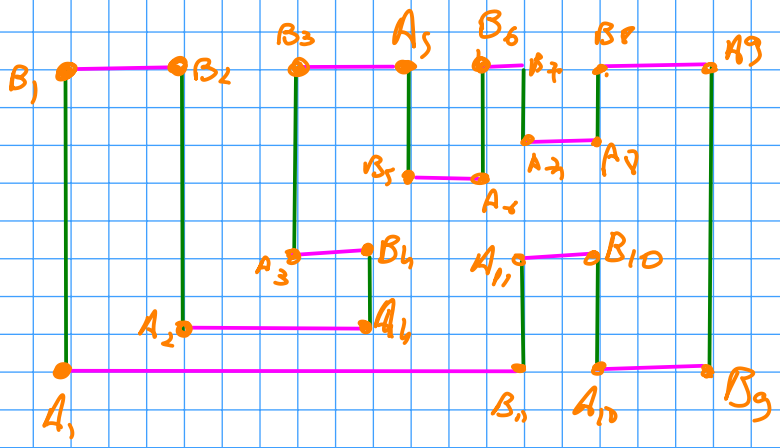
Subiect 8.

Solutia mea, in O(N Log(N)) este urmatoarea:

Separam punctele in functie de y (pe linii orizontale). Pentru fiecare linie orizontala, sortam punctele crescator, si unim punctele doua cate doua (primul cu al doilea, al 3-lea cu al 4-lea etc) - tragem muchie intre puncte.

Unim de asemenea varfurile care fac parte din aceeasi muchie. Putem gasi ordinea varfurilor ca intr-o lista inlantuita / cu un ciclu eulerian.

Rezultatul este pologonul nostru. Cum trebuie sa sortam punctele cu acelasi y crescator (pentru a le uni 2 cate 2), complexitatea este O(N log(N)).



In desenul alaturat, pe aceeasi linie orizontala vor veni punctele B1, B2, B3, A5, B6, B7, B8, A9, care vor fi sortate crescator, si unite doua cate doua:

B1 - B2, B3 - A5, B6 - B7, B8 - A9.

Procedam similar pentru toate celelalte benzi orizontale, si impreuna cu segmentele verticale obtinem poligonul, pe care il putem reprezenta ca o lista dublu-inlantuita.

Asadar, in O(N * log(N)) putem gasi poligonul cerut.

Solutia foloseste o sortare si o baleiere.

```
def Algoritm(Puncte):
        muchii = { }
        puncte_y = map <int, vector <Puncte>>()
        for i in range(0, len(Puncte), 2):
                muchii.add({ puncte[i], puncte[i + 1] })
                puncte_y[puncte[i].y].add(puncte[i])
                puncte_y[puncte[i + 1].y].add(puncte[i + 1])

        for _, pnt in puncte_y:
                pnt = sorted(pnt)
                for i in range(0, len(pnt), 2):
                        muchii.add(pnt[i], pnt[i + 1])

        ordine_muchii = GetOrderFromLinkedList(muchii)
        return ordine_muchii
```