

Teme Algoritmi Avansati
Algoritmi Aproximativi

Knapsack

1.

a) Vom folosi o dinamică similară cu cea de la problema

Knapsack 1/0.

Fie $dp[i][j] = \begin{cases} \text{true}; & \text{dacă suma } i \text{ poate fi obținută folosind} \\ & \text{o submulțime din mulțimea primelor } j \text{ elemente;} \\ \text{false}; & \text{în caz contrar;} \end{cases}$

Vom inițializa astfel:

$$dp[i][j] = \text{false} \quad \forall i \in \{1, 2, \dots, K\} \quad \forall j \in \{0, 1, 2, \dots, m\}$$

$$dp[0][j] = \text{true} \quad \forall j \in \{0, 1, 2, \dots, m\}$$

Cod:

```
for(int i=1; i<=K; ++i) {  
    for(int j=0; j<=m; ++j)  
        dp[i][j] = false;  
}  
for(int j=0; j<=m; ++j) {  
    dp[0][j] = true;  
}
```

```

for (int j = 1; j <= m; ++j) {
    for (int i = k; i >= 0; --i) {
        dp[i][j] = dp[i][j-1];
        if (dp[i][j-1] == true && i + s[j] <= k) {
            dp[i + s[j]][j] = true;
        }
    }
}

```

```

int solutie = 0;
for (int i = k; i >= 0; --i) {
    if (dp[i][n] == true) {
        solutie = i;
        break;
    }
}

```

// Variabila "solutie" are acum raspunsul.

Recurinta folosita a fost: $dp[i][j] = \begin{cases} \text{true, daca } i - s[j] \geq 0 \\ \text{si } dp[i - s[j]][j-1] \text{ este true;} \\ dp[i][j-1]; \text{ in caz contrar} \end{cases}$

A doua ramură din recurentă are ca semnificație faptul că nu lăm elementul s_j în submulțime.

Solutia este acel i din mulțimea $\{0, 1, 2, \dots, k\}$ a.t. $dp[i][n] = \text{true}$ și i maxim posibil.

Algoritmul este corect, deoarece generează toate sumele posibile ce pot fi alcătuite din mulțimea $\{0, 1, 2, \dots, k\}$. Acest lucru se realizează considerând fiecare element s_j ca facând sau nefăcând parte din submulțimea elementelor luate.

Algoritmul ia astfel în considerare toate sumele posibile și garantează că o identifică pe cea optimă.

Algoritmul este pseudo-polynomial, deoarece complexitatea sa depinde de 2 parametrii independenți: K și m .

Complexitate timp: $O(K \cdot m)$

Complexitate spațiu: $O(K \cdot m)$

Complexitatea în spațiu se poate reduce la $O(k)$, deoarece se poate observa faptul că dinamica are nevoie doar de linia anterioră în calculul recurenței și atunci putem calcula întreaga dinamică pe o singură linie, suprascriind-o de la dreapta la stânga.

```
    int solutie = 0;  
    for (int j=1; j<=m; ++j) {  
        if (solutie + s_j <= K) {  
            solutie += s_j;  
        } else {  
            solutie = max(solutie, s_j);  
            break;  
        }  
    }
```

// Variabila "solutie" are acum răspunsul.

Algoritmul dat rulează în timp $O(n)$ (realizează un nr. constant de instrucțiuni pt. fiecare s_j în parte), iar complexitatea în spațiu este $O(1)$. (avem doar variabila "soluție", iar fiecare s_j este folosit în instrucțiuni a singură dată, ceea ce ne permite să tinem în memorie doar s_j -ul curent).

Dacă algoritmul nu intră pe "else" înseamnă că $\sum_{j=1}^m s_j \leq K$, de unde rezultă că soluția optimă este chiar $\sum_{j=1}^m s_j$, pe care algoritmul o va și întoarcă.

Dacă se intră pe "else" înseamnă că există un $j \in \{1, 2, \dots, m\}$ a.t. $(s_1 + s_2 + \dots + s_{j-1}) + s_j > K \Rightarrow (s_1 + s_2 + \dots + s_{j-1}) + s_j \geq K+1 \Rightarrow$

$$\Rightarrow \underbrace{(s_1 + s_2 + \dots + s_{j-1})}_{2} + s_j \geq \frac{K}{2} \quad (\text{media aritmetică dintre } s_1 + \dots + s_{j-1} \text{ și } s_j \text{ este } \geq \frac{K}{2}) \Rightarrow$$

\Rightarrow Există 3 cazuri: $\left(s_1 + s_2 + \dots + s_{j-1} > \frac{K}{2} \text{ și } s_j \leq \frac{K}{2} \right)$ sau $\left(s_1 + s_2 + \dots + s_{j-1} \leq \frac{K}{2} \text{ și } s_j > \frac{K}{2} \right)$ sau ambele egale cu $\frac{K}{2}$.

cum în algoritm avem linia "soluție = max(soluție, s_j)" \Rightarrow
 \Rightarrow $\text{soluție} \geq \frac{K}{2}$.

Dacă $\frac{\text{soluție}}{\text{OPT}} > \frac{1}{2}$ operația va fi puțin pe jumătate suma optimă.

$\Rightarrow \frac{\text{OPT}}{\text{soluție}} \leq 2 \Rightarrow$ Algoritmul aproximativ dat

Load Balance

3. Ordered-Scheduling Algorithm

Fie K indicele maximii cu load-ul maxim în urma executării algoritmului.

$$ALG = \text{load}(K)$$

Fie q ultima activitate adăugată pe masina K .

Fie $\text{load}'(i) = \text{load}-ul$ maximii i fix înainte ca activitatea q să fie asociată masinii K . (load-ul maximelor după atribuirea primelor $q-1$ activități).

$$\Rightarrow \text{load}'(i) = \begin{cases} \text{load}(i) & \text{pt. } i \neq K ; i \in \{1, 2, \dots, m\} \\ \text{load}(i) - t_q & \text{pt. } i = K \end{cases}$$

De asemenea, din modul de funcționare al algoritmului \Rightarrow

$$\Rightarrow \min \left\{ \text{load}'(i) \mid i \in \{1, 2, \dots, m\} \right\} = \text{load}'(K)$$

$$ALG = \text{load}(K) = \text{load}'(K) + t_q$$

$\text{load}'(K) \leq \frac{1}{m} \left[\left(\sum_{1 \leq j \leq m} t_j \right) - t_q \right]$, deoarece activitatea q nu a fost încă atribuită, iar

$\text{load}'(K) = \min \left\{ \text{load}'(i) \mid i \in \{1, 2, \dots, m\} \right\} \Rightarrow \text{load}'(K)$ este cel mult media aritmetică a activităților deja atribuite.

Pentru cazul când $q \leq m \Rightarrow \text{load}'(k) = 0 \Rightarrow$

$$\Rightarrow \text{load}(k) = ALG = \text{load}'(k) + t_q = t_q \leq \max\{t_i \mid i \in \{1, 2, \dots, m\}\} \leq \text{OPT}$$

$$ALG \leq OPT$$

Dacă cum OPT e optimul $\Rightarrow ALG = OPT$, în acest caz algoritmul găsește soluția optimă.

Pentru cazul $q > m$:

$$\text{load}'(k) + t_q \leq \frac{1}{m} \left[\sum_{1 \leq i \leq m} t_i \right] - t_q + t_q = \frac{1}{m} \sum_{1 \leq i \leq m} t_i + t_q \left(1 - \frac{1}{m}\right) \leq \frac{1}{m} \sum_{1 \leq i \leq m} t_i + \frac{1}{2} (t_m + t_{m+1}) \left(1 - \frac{1}{m}\right) \leq OPT + OPT \cdot \frac{1}{2} \left(1 - \frac{1}{m}\right) =$$

$$= OPT \left(1 + \frac{1}{2} - \frac{1}{2m}\right) = OPT \left(\frac{3}{2} - \frac{1}{2m}\right) \Rightarrow \text{Factorul de aproximare este}$$

$$\frac{3}{2} - \frac{1}{2m} \Rightarrow$$

\Rightarrow Algoritmul este $\left(\frac{3}{2} - \frac{1}{2m}\right)$ aproximativ.

Aproximarea $\frac{3}{2} - \frac{1}{2m}$ nu este tight-bounded, deoarece există Teorema lui Graham care afirma că algoritmul din enunt este $\frac{4}{3}$ -aproximativ.

Load Balance

1.

a) Fie multimea $T = \{40, 40, 50, 70\}$ de activități. Toți timpii sunt < 100 .

Optimul se obține grupând 50 cu 40; 70 cu 40.

Optim:

$$\begin{array}{|c|c|} \hline 70 & 40 \\ \hline \end{array} (=110) \quad \Rightarrow \text{Timpul optim este } 110.$$

$$\begin{array}{|c|c|} \hline 50 & 40 \\ \hline \end{array} (=90)$$

Presupunem că algoritmul studentului grupează activitățile în felul următor:

$$\begin{array}{|c|c|} \hline 50 & 40 \\ \hline \end{array} (=80) \quad \Rightarrow \text{Timpul este } 120.$$

$$\begin{array}{|c|c|} \hline 70 & 50 \\ \hline \end{array} (=120)$$

$$\frac{120}{110} = \frac{12}{11} \approx 1,09 < 1,1 \Rightarrow \text{Algoritmul studentului poate fi } 1,1\text{-approximativ.}$$

b) Dacă toți timpii sunt $\leq 10 \Rightarrow$ În cadrul algoritmului optim nu ar exista 2 masini care să aibă diferență în modul dintre timpuri încărcărilor lor ≥ 10 .

Presupunem că în cadrul algoritmului optim există 2 masini cu indicele i și j ($i \neq j$) astfel încât $load(i) \geq load(j) + 10$. Cum toate activitățile au $t_x \leq 10$ și cum $load(i) \geq 10$ din inegalitatea anterioră ($load(j) \geq 0$) \Rightarrow Sigur putem alege o

activitate p pe care s-o mutăm de pe masina i pe masina j ($t_p \leq 10$).

Noii tempi vor fi:

$$\text{load}'(i) = \text{load}(i) - t_p \leq \text{load}(i) \quad \Rightarrow$$

$$\text{load}'(j) = \text{load}(j) + t_p \leq \text{load}(j) + 10 \leq \text{load}(i)$$

\Rightarrow Observăm că acum ambele masini au load-ul sub load-ul initial al masinii i .

De asemenea, putem mereu să-l alegem pe i ca fiind masina ce dă și timpul algoritmului optim (i.e. $\text{load}(i) = \text{OPT} = \max \{ \text{load}(k) \mid k = \overline{1, m} \}$).

În acest caz, tocmai am găsit un mod de a reduce timpul algoritmului OPT . \Rightarrow Contradicție \Rightarrow În cadrul alg.

Dacă OPT era optimul. \Rightarrow ① optim nu putem avea $i; j$ a.s.t. $\text{load}(i) \geq \text{load}(j) + 10$
și $\text{load}(i) = \text{OPT} = \max \{ \text{load}(k) \mid k = \overline{1, m} \}$

Renunțând la problema, observăm că suma load-urilor este 200 și conform lui ① \Rightarrow Algoritmul optim are ca timp worst-case 105 (dată masina având load-ul 95 în acest caz).

Dacă $\frac{120}{105} \geq 1,14 > 1,1 \Rightarrow$ Algoritmul studientului nu mai este $1,1$ -approximativ.

Travelling Salesman Problem

1.

a) Presupunem că TSP nu rămâne NP-hard.

Fie un graf neorientat $G(V, E)$.

Construim graful neorientat ponderat și complet $G'(V'; E')$, unde $V' = V$ și muchile $e \in E'$ au cost 1 dacă $e \in E$, altfel au cost 2.

Graful $G'(V'; E')$ respectă condițiile necesare aplicării algoritmului TSP pe el.

În urma aplicării algoritmului pe $G'(V'; E')$ vom obține un cost C .

Dacă $C = n = |V| = |V'|$, atunci graful initial G admite ciclu Hamiltonian, deoarece TSP-ul a găsit o configurație care folosește doar muchii din graful original.

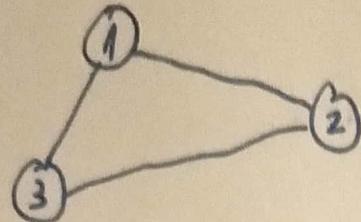
Dacă $C > n$, atunci graful initial G nu admite ciclu Hamiltonian, deoarece s-a folosit cel puțin o muchie de cost 2 (care nu este în G).

Astfel, algoritmul TSP este acum în stare să rezolve problema existenței unui ciclu Hamiltonian într-un graf neorientat carecăre.

Dacă problema ciclului Hamiltonian este NP-hard. \Rightarrow Contradicție
TSP nu este NP-hard. \times

TSP rămâne astfel NP-hard.

b) Fie 3 noduri care să formeze o călă.



Oricum am atribui costuri din mulțimea $\{1; 2\}$ celor 3 muchii să se păstreze inegalitatea triunghiului.

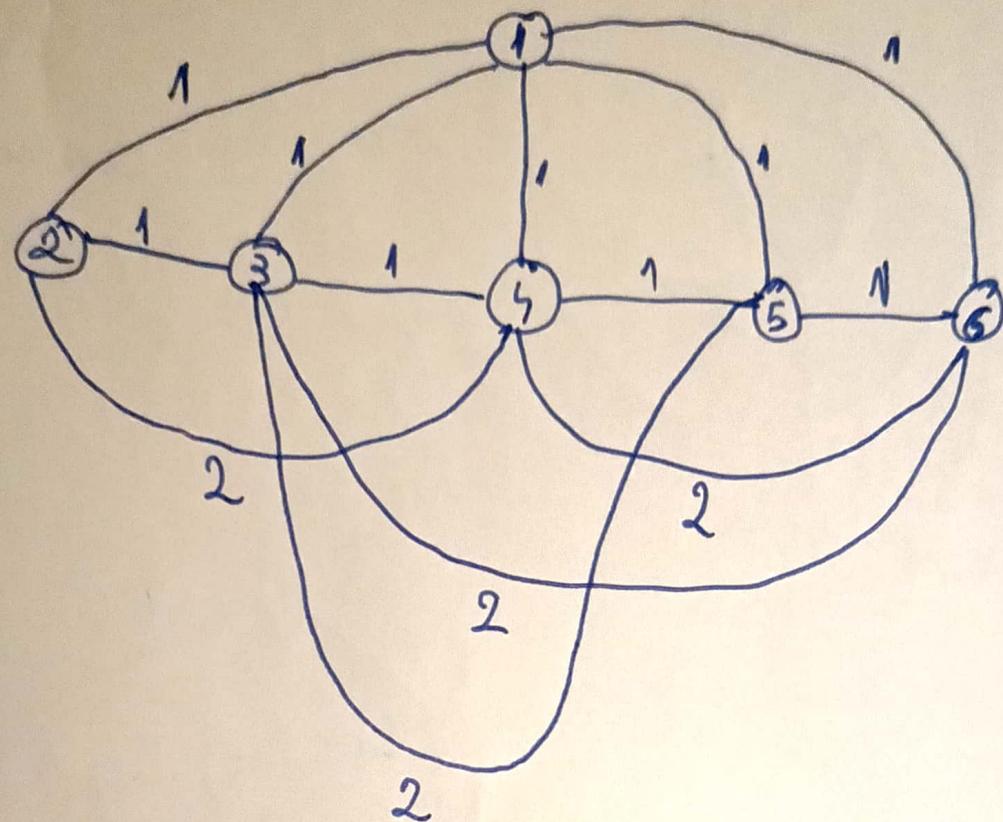
Cost 1-2	Cost 1-3	Cost 2-3	
1	1	1	$\Rightarrow 1+1 \geq 1$
1	1	2	$\Rightarrow 1+1 \geq 2$
1	2	1	$\Rightarrow 1+1 \geq 2$
1	2	2	$\Rightarrow 1+2 \geq 2$
2	1	1	$\Rightarrow 1+1 \geq 2$
2	1	2	$\Rightarrow 1+2 \geq 2$
2	2	1	$\Rightarrow 1+2 \geq 2$
2	2	2	$\Rightarrow 2+2 \geq 2$

Fie $d(x, y)$ distanța minimă de la x la y .

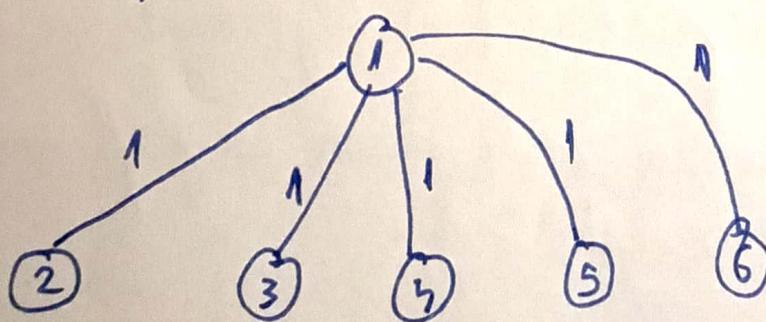
Pt. u și v noduri vecini avem $d(u, v) = \text{costul muchiei } uv$.
 $d(u, v) \leq 2$ dacă u și v noduri vecini.

Fie w un nod intermediar pe un lant de la u la v.
 $d(u, w) + d(w, v) = d(u, v) \geq 1+1=2 \geq d(u, v)$ (u și v noduri vecini)
 \Rightarrow Se păstrează inegalitatea triunghiului.

c) Fixează următorul graf:

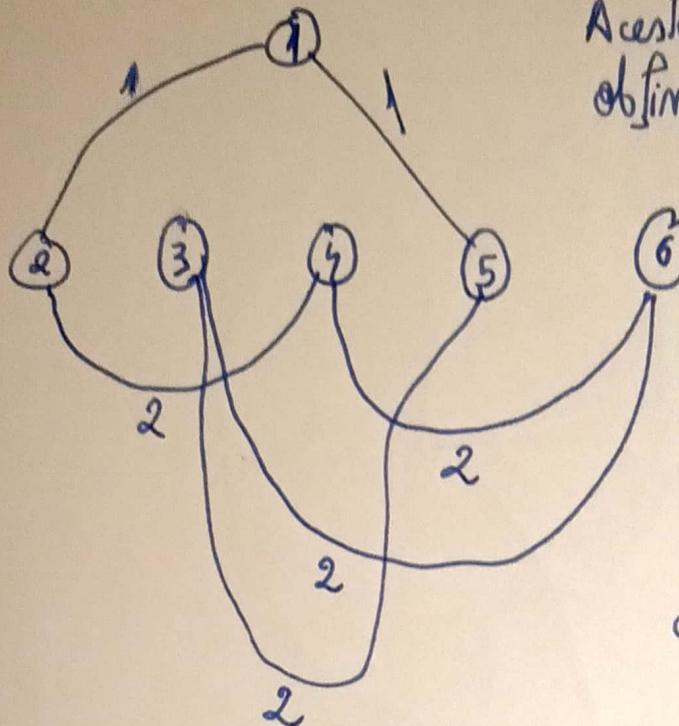


Un arbore parțial de cost minim este:



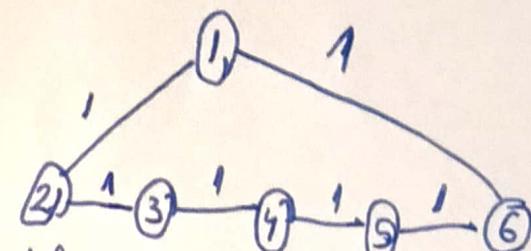
Această are costul 5 și este minimul posibil.
(numărul de noduri - 1).

Fixăm rădăcina arborelui parțial de cost minim drept nodul 1 și considerăm că avem următoarea preordine generată de acest arbore: 1, 2, 4, 6, 3, 5. Concatenăm 1 la final și obținem ciclul Hamiltonian căutat de algoritmul din curs: 1, 2, 4, 6, 3, 5, 1.



Acesta este ciclul Hamiltonian obținut de algoritm.

Costul său este 10,
dar ciclul Hamiltonian
de cost minim pentru
graful dat este
1, 2, 3, 4, 5, 6, 1, ce are
costul 6.



(ciclul Hamiltonian de cost minim)

$$\frac{\text{ALG}}{\text{OPT}} = \frac{10}{6} = \frac{5}{3} \approx 1,6 > 1,5 = \frac{3}{2} \Rightarrow \text{Algoritmul nu este}$$

$\frac{3}{2}$ -aproximativ, deoarece există măcar un graf unde
factorul de aproximare este $> \frac{3}{2}$.

Vertex Cover

a) Fie un exemplu de forma:

$$(x_1 \vee x_{11} \vee x_{12}) \wedge (x_1 \vee x_{21} \vee x_{22}) \wedge \dots \wedge (x_1 \vee x_{m1} \vee x_{m2})$$

În acest caz, optim este să setăm doar $x_1 = \text{true}$, deoarece
 x_1 este singura variabilă prezentă în toate predicatelor, restul
variabilelor apărând fiecare în exact un predicat.

Algoritmul descris riscă să nu îl aleagă niciodată pe x_1 , ceea ce înseamnă că va fi nevoie să marcheze în alte variabile (unde m este numărul de predicate).

Algoritmul este astfel m -aproximativ. ($\frac{\text{ALG}}{\text{OPT}} = \frac{m}{1} = m$)

Mai rău decât m nu se poate, deoarece multimea C din codul algoritmului scade în dimensiune cu cel puțin 1 pentru fiecare variabilă marcată cu true.

b) În loc să marcăm un singur x_i aleator din C_j putem marca toate variabilele din C_j . Algoritmul devine următorul:

1. Fie $C = \{C_1, \dots, C_m\}$ multimea de predicate, $X = \{x_1, \dots, x_m\}$ multimea de variabile.

2. Cât timp $C \neq \emptyset$ execuță:

(a) Alegem aleator $C_j \in C$.

(b) $x_i \leftarrow \text{true}$ pentru $(\forall)x_i \in C_j$

(c) Eliminăm din C toate predicatele ce conțin măcar un x din C_j (inclusiv și C_j)

3. Soluția constă din variabilele pe care le-am setat ca true pe parcursul execuției algoritmului.

Demonstrație 3-aproximativ:

Presupunem că avem un alt altă algoritm care are exact implementarea din cerința problemei, cu excepția că acest algoritm și exact la pasul 2.(c) ce variabile din C_j să marcheze astfel încât să obțină în final

• soluție optimă. Vom numi acest algoritm OPT.

Fie M_i = multimea predicatelor satisfăcute în urma celei de a i -a extragere a lui C_j (realizată la 2.(a) în cadrul din cerință).

Observăm faptul că $C_i \cap M_i = \emptyset$, unde C_i = multimea C după a i -a extragere și eliminare de predicate din C .

De asemenea, $M_{i, \text{ALG}} \supseteq M_{i, \text{OPT}}$, unde $M_{i, \text{ALG}}$ este M_i în timpul aplicării algoritmului dat la punctul b), iar $M_{i, \text{OPT}}$ este M_i în timpul aplicării algoritmului OPT.

Cum $M_{i, \text{OPT}} \subseteq M_{i, \text{ALG}} \Rightarrow$ vom ajunge la soluție în cel mult la fel de multe extrageri ca și algoritmul OPT.

Dar în cel mai rău caz, numărul de extrageri va fi același, situație în care am folosit de cel mult 3 ori mai multe variabile decât OPT (pot exista totuși predicate unde și OPT a marcat cu true 2 sau 3 variabile).

\Rightarrow În cel mai rău caz folosim de 3 ori mai multe variabile \Rightarrow Algoritmul descris la punctul c) este 3-approximativ.

c) Fie $C = \{C_1; C_2; \dots; C_m\}$ multimea de predicate și $X = \{x_1; x_2; \dots; x_m\}$ multimea de variabile.
Avem restricții:

$0 \leq x_i \leq 1 \quad (\forall) i = \overline{1, m} \text{ și}$

$x_{j_1} + x_{j_2} + x_{j_3} \geq 1; x_{j_1}, x_{j_2}, x_{j_3} \in C_j \text{ pf. } (\forall) C_j \in C$

Vrem să minimizăm $\sum_{1 \leq i \leq m} x_i$.

d) După ce am găsit valorile reale din $[0; 1]$ pf. fiecare variabilă, vom lua doar acele variabile x_i pf. care $x_i \geq \frac{1}{3}$, cu $i = \overline{1, m}$. Multimea variabilelor luate constituie soluția problemei. Deoarece fiecare predicat are doar 3 variabile și am atribuit valori variabilelor astfel încât $x_{j_1} + x_{j_2} + x_{j_3} \geq 1 \Rightarrow$
 \Rightarrow Fiecare predicat va conține mereu cel puțin o variabilă x_i a.t. $x_i \geq \frac{1}{3} \Rightarrow$ Algoritmul generează mereu o soluție corectă, dar nu neapărat de dimensiune minimă.

Fie funcția $f(x_i) = \begin{cases} 1 & \text{dacă } x_i \geq \frac{1}{3} \\ 0 & \text{dacă } x_i < \frac{1}{3} \end{cases} \quad (\forall) i = \overline{1, m}$.

$\sum_{1 \leq i \leq m} f(x_i) =$ cardinalul multimii soluție generată de algoritmul de mai sus

$\sum_{1 \leq i \leq m} f(x_i) \leq \sum_{1 \leq i \leq m} 3 \cdot x_i = 3 \sum_{1 \leq i \leq m} x_i$, deoarece pf. $(\forall) x_i \geq \frac{1}{3}$

avem $3x_i \geq 3 \cdot \frac{1}{3} = 1 = f(x_i)$, iar pentru $(\forall) x_i < \frac{1}{3}$

avem $3x_i \geq 0 = f(x_i) \Rightarrow$ Inegalitatea este aderanță.

$$\sum_{1 \leq i \leq m} f(x_i) \leq 3 \cdot \sum_{1 \leq i \leq m} x_i$$

$\sum_{1 \leq i \leq m} x_i$ este suma minimizată de programarea liniară, suma finală înțeleasă ≤ decât orice soluție cu valori discrete pentru problema de prog. liniară, printre aceste soluții numărându-se și OPT-ul pentru problema 3CNF.

$$\sum_{1 \leq i \leq m} f(x_i) \leq 3 \cdot \sum_{1 \leq i \leq m} x_i \leq 3 \cdot \text{OPT} \Rightarrow \text{Algoritmul este } 3\text{-approximativ.}$$