Tema 1 Algoritmi Avansati

Moroianu Theodor

23 martie 2021

1 Exerciţiul Knapsack

Cerinta 1

Consideram functia urmatoare, scrisa in limbajul *C++*:

```
int SumaMax(vector <int> w, int K)
{
    vector <int> viz(K + 1);
    viz[0] = 1;
    for (auto i : w)
        for (int j = K; j >= i; j--)
            viz[j] |= viz[j - i];

    int ans = K;
    while (!viz[ans])
        ans--;
    return ans;
}
```

Din cod se observa imediat ca are o complexitate de $\Theta(N * K)$, unde N este numarul de obiecte.
Tot din cod se vede ca spatiul folosit este $\Theta(K)$.

Lema 1. *Cand am procesat primele i elemente, $viz_x = 1$ daca si numai daca exista o submultime a multimii $\{ W_1, \dots W_i \}$ cu suma x.*

Demonstrație. Demonstram lema prin inductie.
Cazul $i = 0$ este elementar: $W_x = 1$ daca si numai daca $x = 0$.

Cand adaugam un element Q in multime, apar doua cazuri:

- Nu consideram elementul Q pentru a obtine suma x. Observam ca algoritmul nostru trateaza acest caz ne-setand niciodata o valoare de 1 in 0.
- Consideram elementul Q pentru a obtine suma x. Observam de asemenea ca algoritmul nostru trateaza acest caz setand $viz'_i = viz_i \vee viz_{i-Q}$.

Asadar, algoritmul prezentat mai sus ofera o solutie corecta. □

Cerinta 2

Consideram functia urmatoare, scrisa in limbajul *C++*:

```
int K, ans = 0, act;
cin >> K;

while (cin >> act)
{
    ans += act;
    if (ans > K)
        ans -= act;
    else if (ans < act)
        ans = act;
}

cout << "Answer is " << ans << '\n';
```

Din cod se observa ca are o complexitate de $\Theta(N)$, unde N este numarul de obiecte citite. Tot din cod se vede ca spatiul folosit este $\Theta(1)$, mai exact 3 variabile.

Lema 2. *Algoritmul prezentat mai sus este* 2*-aproximativ.*

Demonstrație. Consideram secventa de obiecte citite (exceptandul pe K) S, si OPT solutia optima.
Exista doua cazuri:

- $\exists\ i$ a.i. $S_i > \frac{OPT}{2}$.
  Altfel spus, exista un element mai mare decat solutia optima impartita la 2. Algoritmul nostru ne garanteaza ca va gasi un raspuns mai mare sau egal cu S_i, considerand cazurile cand raspunsul este format dintr-un singur element.

- $\forall\ i\ S_i \leq \frac{OPT}{2}$.
  In acest caz, avem garantia ca atata timp cat raspunsul partial este mai mic decat $\frac{OPT}{2}$ putem sa mai adaugam un element.
  Asadar, cum suma elementelor este cel putin OPT si noi putem mereu adauga elemente cat timp suma noastra este mai mica decat $\frac{OPT}{2}$ stim ca algoritmul o sa aduca raspunsul in intervalul $[\frac{OPT}{2} + 1, OPT]$.

Observam ca in ambele situatii gasim un raspuns satisfacator. □

Codul pentru cele doua exemple se gaseste in *ZIP*-ul atasat.

2 Load Balance

Cerinţa 1

A

Consideram setul de greutati $S = \{\ 20,\ 30,\ 60,\ 90\ \}$. Solutia optima este partitionarea multimii S in felul urmator:

- $S_1 = \{\ 20,\ 90\ \}$
- $S_2 = \{\ 30,\ 60\ \}$

Asadar, notand cu OPT solutia optima, avem $OPT = 110$.

Consideram acum urmatoarea partitionare:

- $S'_1 = \{\ 20,\ 90\ \}$
- $S'_2 = \{\ 30,\ 60\ \}$

Observam ca $ALG = 120$, unde ALG este solutia algoritmului care ne-a generat a doua paritie.

Stiind ca $120 < 110 * 1.1$, exista cel putin un test pentru care incarcarea masinilor cu suma de 80, respectiv 120 este o 2-aproximare. Altfel spus, un algoritm care pentru S ne genereaza impartirea S'_1 si S'_2 este 2-aproximativ pe acel caz.

Putem asadar afirma ca testul dat nu prezinta un contra-exemplu la afirmatia de aproximare al algoritmului, acesta putand fi corect. □

B

Fie OPT solutia optima.

Lema 3. $OPT < 110$.

Demonstraţie. Fie $S_{1,2}$ partitionarea lui S in solutia optima, cu $\sum_{i \in S_1} i \geq \sum_{i \in S_2} i$.
Mai departe vom considera $G_{1,2}$ ca fiind suma elementelor din S_1 respectiv S_2.

Exista doua cazuri:

- $G_1 < 110$. *Qed.*
- $G_1 \geq 110$. Alegem un element x din S_1. Conform restrictiilor, $1 \leq x \leq 10$.
  Il mutam pe x din S_1 in S_2, obtinand astfel o solutie mai buna (G_1 scade, si G_2 ramane mai mic decat 100). Contradictie.

□

Conform lemei de mai sus, $OPT < 110$, sau $OPT \leq 109$ si deci orice algoritm 1.1 aproximativ este mai mic sau egal cu $1.1 * 109 = 119.9$.

Cum $ALG = 120$, factorul sau de aproximare nu este corect. □

Cerinţa 2

A

Consideram urmatoarea problema (triviala) de minimizare:
Se da un numar X, si se cere un numar cat mai mic mai mare sau egal cu X.

Consideram urmatorii algoritmi:

- Algoritmul optim, cu $OPT(x) = x$.
- Algoritmul 1, cu $ALG_1(x) = x$.
- Algoritmul 2, cu $ALG_2(x) = 2 * x$.

Este trivial de arat ca ALG_1 este 2-aproximativ si ALG_2 este 4-aproximativ (ALG_1 este chiar 1 aproximativ si ALG_2 2-aproximativ).

Pentru inputul $x = 1$ obtinem ca $ALG_1(x) = 1$ si $ALG_2(x) = 2$, deci exista cel putin un scenariu pentru care $ALG_2(I) \geq 2 * ALG_1(I)$.

Consideram acum doua alegeri diferite pentru $ALG_{1,2}$:

- Alegem algoritmul 1 ca $ALG'_1(x) = 2 * x$.
- Alegem algoritmul 2 ca $ALG'_2(x) = x$.

Din definitie, $ALG'_1 > ALG'_2$, pentru inputul $x = 1$ obtinem ca $ALG'_1(x) = 2$ si $ALG_2(x) = 1$, deci exista cel putin un scenariu pentru care $ALG_2(I) \ngeq 2 * ALG_1(I)$.

Asadar, propozitia este falsa.
De notat ca nici inegalitatea inversa nu este adevarata. □

B

Notand OPT algoritmul optim, pentru oricare input avem urmatoarele inegalitati:

$$OPT \leq ALG_1 \leq 2 * OPT$$
$$OPT \leq ALG_2 \leq 4 * OPT$$

Obtinem asadar urmatoarea inegalitate:

$$OPT \leq ALG_1 \leq 2 * OPT \leq 2 * ALG_2$$

Asadar, pentru oricare input avem $ALG_1 \leq 2 * ALG_2$, si deci pentru niciun input nu avem $ALG_1 > 2 * ALG_2$. Afirmatia este deci adevarata. □

Cerinţa 3

Consideram algoritmul Ordered-Scheduling, si dorim sa aratam ca factorul sau de aproximare este de $\frac{3}{2} - \frac{1}{2m}$.
De fapt, vom gasi un factor de aproximare si mai bun. Mai precis vom arata ca:

$$\frac{OSA}{OPT} \leq \frac{4}{3} - \frac{1}{3 * m}$$

Prin OSA intelegem raspunsul dat de algoritmul Ordered-Scheduling, prin OPT intelegem raspunsul optim, si prim m se intelege numarul de masini. Load-ul masinii i dat de algoritmul OSA este notat cu L_i si greutatea obiectului j este notata cu G_j.

Lema 4. *Bound-ul de* $\frac{3}{2} - \frac{1}{2*m}$ *este mai slab decat bound-ul de* $\frac{4}{3} - \frac{1}{3*m}$.

Demonstraţie. Verificam inegalitatea prin echivalente:

$$\frac{4}{3} - \frac{1}{3 * m} \leq \frac{3}{2} - \frac{1}{2 * m}$$

$$\frac{4 * 6 * m}{3} - \frac{6 * m}{3 * m} \leq \frac{3 * 6 * m}{2} - \frac{6 * m}{2 * m}$$

$$8 * m - 2 \leq 9 * m - 3$$

$$1 \leq m$$

Cum m reprezinta numarul de masini, este cel putin egal cu 1. $\square$

Asadar, vom demonstra ca un upper-bound pentru $\frac{OSA}{OPT}$ este $\frac{4}{3} - \frac{1}{3*m}$.

Fie X o instanta a problemei de Load Balancing, OPT solutia optima, si OSA solutia oferita de algoritmul de Ordered Scheduling.
Fie K indicele ultimului obiect pus in masina cu load maxim in OSA.

Lema 5. *Daca $K \neq N$, putem sa eliminam toate obiectele cu indice mai mare decat K, obtinand un nou input pe care algoritmul ALG are un factor de aproximare cel mai mare sau egal cel al inputului initial.*

Demonstraţie. Cum K este ultimul obiect pus de OSA in cea mai umpluta masina, eliminarea obiectelor de dupa K nu afecteaza raspunsul OSA.
Pe de alta parte, algoritmul optim OPT poate fi influentat de eliminarea acestor obiecte.
Asadar, OSA nu se modifica, si OPT scade sau nu se modifica, factorul de aproximare al lui OSA fie nu se modifica fie creste. $\square$

In continuare, vom considera ca masina de load maxim in cadrul algoritmului OSA este masina cu obiectul N, orice alt input pudandu-se reduce la forma aceasta, conform lemei de mei sus.

Consideram doua cazuri posibile:

Caz 1: $G_N > \frac{OPT}{3}$. Cum obiectul N este cel mai usor, inseamna ca nu pot fi puse mai mult de doua obiecte in aceeasi masina. Totusi, in cazul in care sunt cel mult doua obiecte per masina, algoritmul OSA este optim, asignand masinii K obiectele nr. K si $2*m-K+1$.

Asadar, obtinem ecuatia 1:

$$G_N > \frac{OPT}{3} \Rightarrow OSA = OPT$$

Caz 2: $G_N \leq \frac{OPT}{3}$. Fie k masina de load maxim. Conform afirmatiilor de mai sus, obiectul N este asignat acesteia.
Asadar, inainte sa ii fie adaugat obiectul N, masina k avea cel mai mic load, si deci mai mic decat media:

$$L_k - G_N \leq \frac{1}{m} * \sum_{i=1}^{n-1} G \iff L_k \leq \frac{1}{m} * \sum_{i=1}^{n} G_i + (1 - \frac{1}{m}) * G_N$$

Stim ca OPT este mai mare sau egal cu media, si conform cazului in care ne aflam G_N este cel putin $\frac{OPT}{3}$, deci avem:

$$L_k \leq \frac{1}{m} * \sum_{i=1}^{n} G_i + (1 - \frac{1}{m}) * G_N$$
$$G_N \leq \frac{OPT}{3}$$
$$OPT \geq \frac{1}{m} * \sum_{i=1}^{n} G_i$$

Combinand cele 3 relatii, obtinem:

$$L_k \leq OPT + (1 - \frac{1}{m}) * \frac{OPT}{3} \iff L_k \leq OPT * (\frac{4}{3} - \frac{1}{3 * m})$$
$$\Rightarrow \frac{OSA}{OPT} \leq \frac{4}{3} - \frac{1}{3 * m} \leq \frac{3}{2} - \frac{1}{2m}$$

□

3 Problema Comis Voiajorului

Cerinţa 1

A

Pentru a rezolva acest exercitiu presupunem ca determinarea existentei unui ciclu hamiltonian este *NP-Complete*.

Lema 6. *Problema Covis Voiajorului este NP-Hard pe grafurile complete cu muchii cu valori in multimea* $\{\ 1,\ 2\ \}$.

Demonstraţie. Vom demonstra lema prin reducerea determinarii existentei unui ciclu hamiltonian la problema comis voiajorului intr-un graf cu muchii de 1 si 2.

Fie $G = (V, E)$ un graf oarecare.
Consideram $G' = (V, E')$, unde $E' = \{(a, b, 1) \mid (a, b) \in E\} \cup \{(a, b, 2) \mid (a, b) \notin E\}$.
In alte cuvinte, G' este un graf complet peste acelasi set de noduri, in care muchiile existente in G sunt inlocuite cu muchii de cost 1, si muchiile inexistente cu muchii de cost 2.

Afirmatie: *Fie C costul minim al comis voiajorului in G', si H valoarea de adevar al existentei unui ciclu hamiltonian in G.*
Atunci, $H \iff (C = |V|)$.
Demonstratia Afirmatiei:

- $H \implies (C = |V|)$ Daca exista un ciclu hamiltonian in G, atunci exista un ciclu hamiltonian cu muchii de cost 1 in G'. Cum nu exista muchii de cost mai mic de 1 si orice drum al comis voiajorului are distanta minim $|V|$, atunci $C = |V|$.
- $(C = |V|) \implies H$ Daca exista o parcurgere a nodurilor din G' de cost $|V|$, cum orice parcurgere are lungimea minim $|V|$ atunci pargurgerea este un ciclu hamiltonian trecand exclusiv prin muchii de cost 1.
  Asadar, acelasi ciclu hamiltonian exista si in G.

Asadar, existenta unui ciclu hamiltonian in G este echivalenta printr-o transformare polinomiala cu rezolvarea problemei comis voiajorului pe un graf cu muchii din multimea $\{\ 1,\ 2\ \}$.
PCV este asadar *NP-Hard* si pentru instanta cu muchii de 1 si 2. □

B

Muchiile avand costuri de 1 si 2, orice combinatie de 3 muchii $a,\ b,\ c$ respecta inegalitatea $a+b+c-max(a,b,c) \geq max(a,b,c)$. $\square$

C

Algoritmul prezentat la curs, cu factor de aproximare 2 este urmatorul:

- Se construieste un APM al grafului.
- Se extrage parcurgerea in-ordine, la care adaugam din nou radacina.
- Se trag muchii intre toate perechile de noduri consecutive.

Algoritmul prezentat la curs asadar plimba comis voiajorul conform parcurgerii in preordine, aducandu-l inapoi la destinatie dupa-aceea.

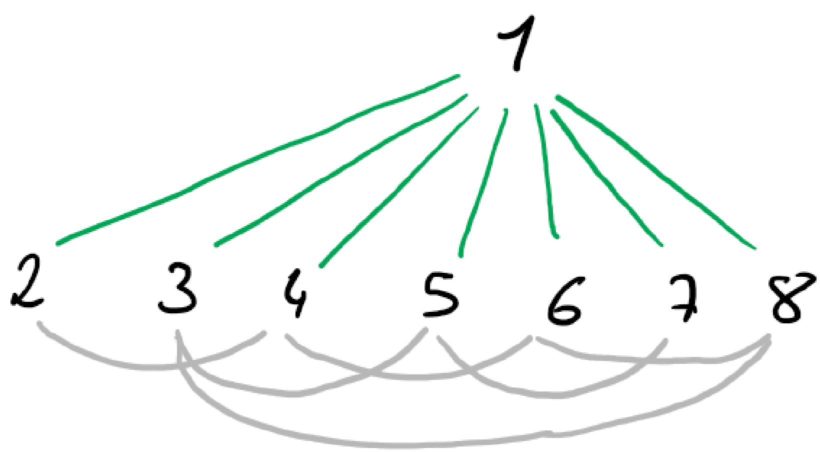

Figura 1: Input cu grad de aproximare mai mare de $\frac{3}{2}$

Consideram graful depictat de imaginea de mai sus $G = (V, E)$, unde:

- $V = \{\ 1,\ 2,\ \ldots,\ 8\ \}$
- E este format din toate muchiile verzi si gri, cu cost 1, si toate muchiile intre noduri neconectate din imagine, cu cost 2.

Graful admite un ciclu hamiltonian exclusiv din muchii de 1:

$$1 \to 2 \to 4 \to 6 \to 8 \to 3 \to 5 \to 7 \to 1$$

Asadar, costul solutiei optime este 8.

Algoritmul descris la curs va alege ciclul $1 \to 2 \to 3 \cdots \to 8 \to 1$, de cost 14.

Asadar, cum $14 > \frac{3}{2} * 8$, algoritmul din curs nu este $\frac{3}{2}$ aproximativ. $\square$

Cerinţa 2

A

Consideram urmatoarele 4 puncte:

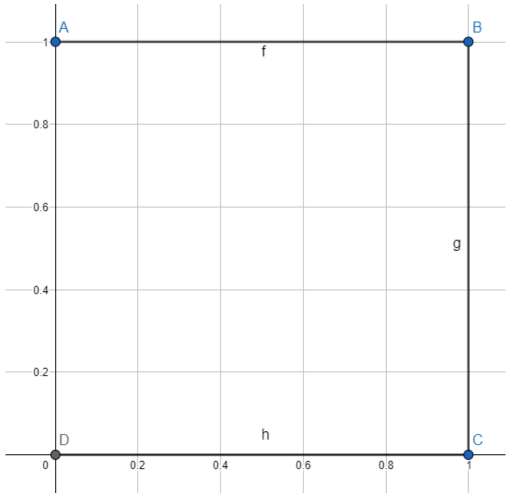

Figura 2: APM intr-un patrat

APM-ul are un cost de 3.
Observam noul APM obtinut adaugand punctul $(\frac{1}{2}, \frac{1}{2})$:

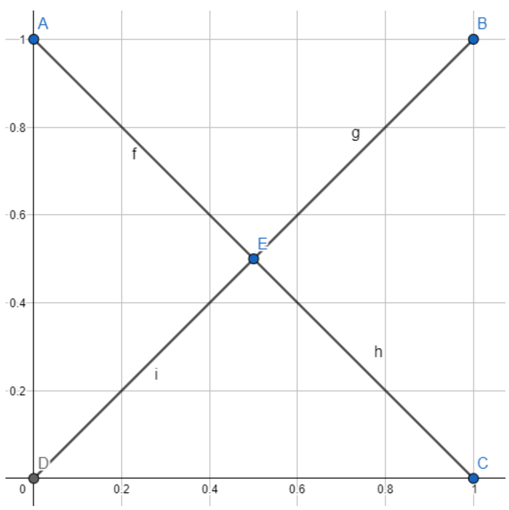

Figura 3: APM intr-un patrat

Noul APM are un cost de $4 * \frac{\sqrt{2}}{2} \simeq 2.8284$.
Asadar, adaugand punctul $(\frac{1}{2}, \frac{1}{2})$ multimii $\{\ (0,0),\ (0,1),\ (1,1),\ (1,0)\ \}$ obtinem un APM de cost mai mic. $\square$

B

Consideram urmatorul algoritm ALG:

- Gasim un MST al lui $P \cup Q$.
- Gasim parcurgerea in pre-ordine (*"Pre-Order Traversal"*) al MST-ului.
- Stergem toate nodurile din Q din parcurgere.
- Unim toate nodurile adiacente in parcurgerea ramasa, obtinand un lant.

Lema 7. *Algoritmul ALG produce un lant W cu $L(W) \leq 2 * L(G)$, unde $L(X)$ reprezinta suma lungimilor muchiilor din graful X.*

Demonstraţie. Din functionarea parcurgerii in pre-ordine, fiecare muchie din MST va fi considerata de cel mult doua ori:

- Cand parcurgerea intra in subarborele acelei muchii.
- Cand parcurgerea iese din subarbore.

Stergerea unor noduri din parcurgere, din inegalitatea triunghiului, nu creste lungimea muchiilor parcurse, asadar $L(W) \leq L(EUL) = 2 * L(APM)$, unde EUL reprezinta parcurgerea euleriantă a APM-ului.

Avem deci $L(W) \leq 2 * L(MST)$. □

Folosind algoritmul ALG, avem un algoritm care sa ne dea un arbore cu costul cel mult $2 * L(APM)$. Asadar, costul oricarui APM al lui P o sa fie cel mult costul arborelui generat, adica $L(T) \leq L(ALG) \leq L(APM)$. □

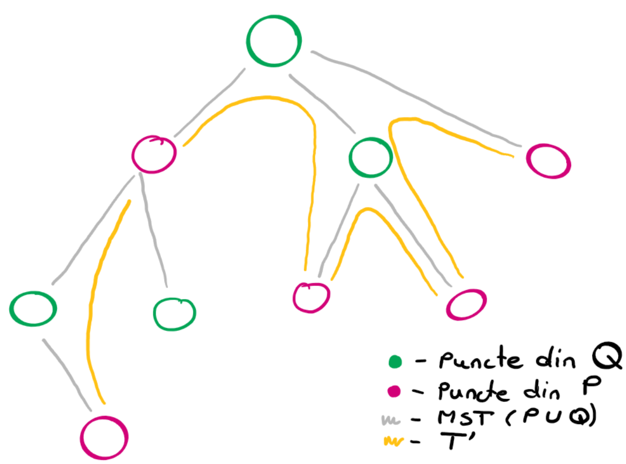

Figura 4: Reducerea unui MST

4 Vertex Cover / SAT

A

Analizam factorul de aproximare al urmatorului algoritm:

```
Greedy_3CNF(C, X):
    C := {C1, ..., Cm}          # mulțimea de predicate
    X := {x1, ..., xn}          # mulțime de variabile

    while C != { }:
        Cj <- C                   # extragem aleator un predicat din C
        Xi <- Cj                  # Extragem o variabila aleator din Cj
        xi := True
        C := C \ xi               # Eliminăm din C toate predicatele
                                  # ce îl conțin pe xi
    return X
```

Fie I_n, cu $n \in \mathbf{N}$ o familie de inputuri definite astfel:

$I_k = (X_1 \vee X_1 \vee X_1) \wedge (X_1 \vee X_2 \vee X_2) \wedge \ldots \wedge (X_1 \vee X_k \vee X_k)$.

Este trivial de aratat ca solutia optima pentru oricare element I_i este $x_i = \delta_{i,1}$, unde prin δ se intelege functia Kronecker.

In cel mai rau scenariu, algoritmul nostru *Greedy-CNF* seteaza variabilele $x_i = 1, \forall i$.

Asadar, observam ca pe multimea de inputuri I algoritmul dat nu respecta nicio constanta de optimizare:

$$lim_{n\to\infty}\frac{ALG(I_n)}{OPT(I_n)} = lim_{n\to\infty}\frac{n}{1} = \infty$$

Se observa usor ca pentru o problema de satisifabilitate formata din N clauze, solutia optima cat si cea generata de greedy sunt mereu in intervalul $[1, N]$: In cel mai bun caz putem satisface toate disjutiile cu o singura variabila, si in cel mai rau caz ne trebuie o variabila per disjunctie.

Cum greedy-ul *Greedy-3CNF* este pe multimea de inputuri $(I_k)_{k\in\mathbf{N}}$ de N ori mai prost decat OPT, si N este cel mai post factor de aproximare putem deduce ca factorul de aproximare al algoritmului *Greedy-3CNF* este N, unde N reprezinta numarul de clauze din input. □

B

Propunem urmatorul algoritm, si afirmam ca reprezinta o 3-aproximare a solutiei optime:

```
Good_Greedy_3CNF(C, X):
    C := {C1, ..., Cm}          # mulțimea de predicate
    X := {x1, ..., xn}          # mulțime de variabile

    while C != { }:
        Cj <- C                   # extragem aleator un predicat din C
        (Xa, Xb, Xc) := Cj        # extragem cele 3 variabile din Cj

        Xa := True                # setam cele 3 variabile ca adevarat
        Xb := True
        Xc := True


        C := C \ Xa \ Xb \ Xc   # Eliminăm din C toate predicatele
                                # ce îl conțin pe Xa, Xb sau Xc
    return X
```

Lema 8. *Algoritmul Good_Greedy_3CNF reprezinta o 3-aproximare a solutiei optime.*

Demonstraţie. Fie o clauza $(X_a \vee X_b \vee X_c)$. Clauza fiind satisfacuta, cel putin una din variabilele X_a, X_b si X_c sunt adevarate. Algoritmul prezentat considera cele 3 variabile adevarate, si deci in cel mai rau caz are 3 variabile adevarate in loc de una. Dupa aceea, algoritmul elimina toate clauzele satisfactute si repeta pasii.

Putem sa privim asignarea variabilelor cu analiza amortizata, mai precis cu ajutorul metodei potentialelor:

- Gasim o solutie optima Y a asignarii variabilelor X.
- Aplicam algoritmul greedy descris mai sus, si pentru fiecare clauza $(X_a \vee X_b \vee X_c)$ tratata stim ca cel putin una din cele 3 variabile este in Y.
- Asociem fiecarei asignare $X_i := True$ costul 1.
- In plus de costul descris mai sus, asociem fiecarei asignare al unei variabile din Y un bonus de 3 (un cost de -3).

Cum pentru fiecare clauza suma bonusurilor minus cea a costurilor este pozitiva, suma totala este pozitiva.

Asadar, $3 * |Y| \;\geq\; |\{X_i \mid X_i = True\}|$ □

C

Fie $X = \{ x_1, x_2, \ldots, x_n \}$ o multime de variabile booleene, si C o formula in forma 3CNF.

Consideram urmatoarea instanta de programare liniara booleana (programare liniara pe intregi cu valori 0-1):

- Pentru fiecare i avem restrictia $x_i \geq 0$.
- Pentru fiecare clauza $(x_a \vee x_b \vee x_c)$ adaugam restrictia $x_a + x_b + x_c >= 1$.
- Formula pe care dorim sa o minimizam este $x_1 + x_2 + \cdots + x_n$.

Exemplu

Ilustram construirea instantei de programare liniara pentru urmatoarea expresie in 3CNF:

$$(X_1 \vee X_2 \vee X_3) \wedge (X_2 \vee X_4 \vee X_5) \wedge (X_1 \vee X_6 \vee X_7) \wedge (X_5 \vee X_7 \vee X_8)$$

Instanta de programare liniara pe care o obtinem este urmatoarea:

$$\begin{aligned} X_1 + X_2 + x_3 &\geq 1 \\ X_2 + X_4 + X_5 &\geq 1 \\ X_1 + X_6 + X_7 &\geq 1 \\ X_5 + X_7 + X_8 &\geq 1 \\ X_1, \ \ldots, \ X_8 &\geq 0 \end{aligned}$$

Ecuatia liniara pe care dorim sa o minimizam este suma variabilelor X:

$$S_{min} = X_1 + X_2 + X_3 + X_4 + X_5 + X_6 + X_7 + X_8$$

Se observa usor ca o solutie optima este de-a alege $X_i = \delta_{i,1} + \delta_{i,5}$. Altfel spus, alegem X_1 si X_5 adevarate si restul false.

Solutia este in mod evident optima, prima si a 3-a clauza fiind disjuncte ca multimi de variabile.

Se poate observa ca solutia instantei de programare liniara este si o solutie a problemei 3CNF.

Lema 9. *Solutiile ale instantei de programare liniara care minimizeaza suma S_{min} sunt aceleasi cu solutiile problemei 3CNF care minimizeaza numarul de variabile adevarate.*

Demonstraţie. Fie C o problema de 3CNF, si W problema corespunzatoare de programare liniara, construita folosind procedeul descris mai sus.

Fie Y o solutie a problemei C. Implicit, din constructia instantei de programare liniara, Y satisface toate integalitatile ale acesteia, si deci formeaza o solutie. Analog, orice solutie a problemei W este si o solutie a problemei C.

Pentru a demonstra ca o solutie minima a lui C este si o solutie minima a lui W observam ca atat W cat si C doresc sa minimizeze numarul de variabile cu valoarea 1 sau *Adevarat*. Cum solutiile celor doua probleme formeaza o bijectie (bijectia fiind chiar functia identitate), este clar ca solutiile minime sunt aceleasi. $\square$

Asadar, prin procedeul descris mai sus putem reduce orice problema de tipul 3CNF cu toti termenii pozitivi (fara negatii) intr-o instanta a programarii lineare pe numere intregi. $\square$

D

Rezolvarea instantelor generale de programare liniara pe numere intregi este cunoscuta ca fiind *NP-Complete*. Totusi, rezolvarea instantelor de programare liniara pe **R** este posibila in timp liniar prin algoritmi de tipul *Simplex*.

Asadar, prezentam urmatorul algoritm 3-aproximativ pentru instanta de programare liniara:

```
Good_Greedy_3Lin_Prog(C, X):
    C := {C1, ..., Cm}          # Mulțimea de inegalitati.
                                # Stim ca totate inegalitatile sunt de
                                # forma x_i + x_j + x_k >= 1
    X := {x1, ..., xn}          # Mulțime de variabile.
                                # Stim de asemenea ca ecuatia pe care
                                # dorim sa o minimizam este
                                # X_1 + ... + X_n

    Y <- Simplex(C, X)          # Rezolvam C pe reale.

    Z := {Z1, ..., Zn}          # Variabilele din raspuns.

    for i in {1, ..., n}:
        if Yi >= 1 / 3:         # Daca o variabila din simplex
            Zi = 1              # are o valoare mai mare de
        else:                   # 1/3, atunci o setam ca
            Zi = 0              # adevarata, daca nu falsa.

    return Z
```

Lema 10. *Algoritmul Good_Greedy_3Lin_Prog reprezinta o 3-aproximare a solutiei optime.*

Demonstrație. Fie o variabila Y_i. Daca valoarea acesteia este mai mica de $\frac{1}{3}$, atunci raspunsul Z_i este 0, si daca valoarea acesteia este mai mare de $\frac{1}{3}$, atunci Z_i este 1. Asadar, se observa imediat ca:

$$Y_i \geq 3 * Z_i$$
$$\sum_{i=1}^{n} Y_i \geq 3 * \sum_{i=1}^{n} Z_i$$

Asadar, algoritmul prezentat este, presupunand ca acesta este corect, o 3-aproximare a solutiei Y.

Pentru a arata ca algoritmul intoarce o solutie valida, observam ca din principiul cutiei in fiecare inegalitate $X_i + X_j + X_k \geq 1$ exista cel putin un element mai mare decat $\frac{1}{3}$, care este transformat in 1.

Pe de alta parte, solutia optima reprezinta o restrangere a problemei calculate de simplex pe numere intregi, si deci este mai mare decat aceasta.

Asadar, avem inegalitatie:

$$SIMPLEX \leq ALG \leq 3 * SIMPLEX$$

$$SIMPLEX \leq OPT$$

$$OPT \leq ALG$$

Primele doua inegalitati provin din explicatiile de mai sus, si a treia inegalitate din minimalitatea solutiei optime.

Asadar, avem inegalitatea urmatoare:

$$ALG \leq 3 * OPT$$

Asadar, algoritmul prezentat este o 3-aproximare a solutiei optime. $\square$