

Tema 1 Algoritmi Avansati

Sociu Daniel

March 23, 2021

Contents

Knapsack

1 Problema 1

1.a

E nevoie doar de un simplu algoritm similar cu knapsack pseudo-polinomial, doar ca valoarea si greutatea sunt egale.

```
#include <bits/stdc++.h>
using namespace std;
#define FOR(i,a,b) for(int i=(a); i<=(b); ++i)
#define FORS(i,a,b) for(int i=(a); i<(b); ++i)

int n, K;

int main()
{
    cin.tie(0);
    ios_base::sync_with_stdio(0);
    cin >> n >> K;
    vector<int> S(n);
    vector<int> dp(K + 1, 0);
    FORS (i, 0, n) {
        cin >> S[i];
    }
    FORS (i, 0, n) {
        for (int j = K; j >= S[i]; j--) {
            dp[j] = max(dp[j], S[i] + dp[j - S[i]]);
        }
    }
    cout << dp[K] << '\n';
}
```

1.b

Acest algoritm este cel puțin $2 \cdot \text{OPT}$ în $O(n)$ deoarece dacă avem o sumă care trece de K , suma pe care o avem până acum sau elementul care încercăm să îl adunăm, este mai mare de $K/2$ și doar selectăm elementul mai mare.

```

#include <bits/stdc++.h>
using namespace std;
#define FOR(i,a,b)   for(int i=(a);i<=(b);++i)
#define FORS(i,a,b)  for(int i=(a);i<(b);++i)

int K, x, answer = 0;

int main()
{
    cin.tie(0);
    ios_base::sync_with_stdio(0);
    freopen("data.txt","r",stdin);
    cin >> K >> K;
    while(cin >> x) {
        answer = answer + x;
        if (answer > K) {
            answer = answer - x;
            if (x > answer) { // the secret behind 1/2 optimal
                answer = x; // if the sum is > K then one
                            // value is >= than half
            }
        }
    }
    cout << answer << '\n';
}

```

Load Balance

1 Problema 1

Fie M_1 si M_2 lista job-urilor incarcate pe masina 1, respectiv masina 2

1.a

Pentru a demonstra ca afirmatia e posibila sa fie adevarata, trebuie sa gasim minim un caz pe care e adevarata:

Sa presupunem masinile cu job-urile:

$$M_1 = \{80\}$$

$$M_2 = \{80, 40\}$$

Observam ca $ALG = OPT \implies ALG \leq 1.1 * OPT$ deci afirmatia lui poate fi adevarata

1.b

Pentru a demonstra ca afirmatia e false, aratam ca nu exista un caz pentru care ar putea fi adevarata

Considerand multimea job-urile J si stiind $\forall x_i \in J, x_i \leq 10$, observam ca diferenta maxima in OPT va fi 10, altfel o masina ar fi avut prea multa incarcatura, si ar exista o aranjare a job-urilor mai optima. Deci:

$$OPT \leq \max\{95, 105\} \leq 105 \text{ (105 e val maxima pt } OPT)$$

Dar in cazul nostru:

$$ALG = \max\{80, 120\} = 120 \implies 1.1 * OPT = 1.1 * 105 \approx 116 \leq ALG = 120$$

Ceea ce e fals, adica e imposibil ca algoritmul sa fie $1.1 * OPT$

2 Problema 3

Din curs stim:

Lema 1.

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

Si stim ca $ALG \leq \frac{3}{2}OPT$

Fie K indicele masinii cu load-ul maxim la finalul algoritmului.

Fie q ultimul job adaugat masinii K

Este evident ca algoritmul e optim in cazul in care sunt mai putine job-uri decat numarul masinilor, deci consideram cazul cand avem mai mult de m job-uri

Fie $load'(M)$ load-ul masinii dupa ce am adaugat primele $q - 1$ job-uri dar nu si job-ul q

O observatie importanta este ca $load'(K)$ este minimul dintre toate masinile si $load(K) = load'(K) + t(q)$, deci:

$$ALG = load(K) = load'(K) + t_q \leq \frac{1}{m} \sum_{i=1}^m load'(i) + t_q = \frac{1}{m} \sum_{i=1}^{q-1} t_i + t_q$$

Observam ca $t_q \leq \frac{1}{2}(t_m + t_{m+1}) \leq OPT$ deoarece job-urile sunt sortate descrescator, deci ultimul job este mai mic sau egal cu media a 2 job-uri precedente. Deci:

$$\frac{1}{m} \sum_{i=1}^{q-1} t_i + t_q \leq \frac{1}{m} \left(\sum_{i=1}^n t_i - t_q \right) + \frac{1}{2}(t_m + t_{m+1}) \leq \frac{1}{m} \sum_{i=1}^n t_i - \frac{1}{2 \cdot m}(t_m + t_{m+1}) + \frac{1}{2}(t_m + t_{m+1}) \leq$$

Inlocuim cu OPT

$$\implies \leq OPT - \frac{1}{2 \cdot m} OPT + \frac{1}{2} OPT = \frac{3}{2} OPT - \frac{1}{2 \cdot m} OPT = \left(\frac{3}{2} - \frac{1}{2 \cdot m} \right) OPT$$

Deci avem ca:

$$ALG \leq \left(\frac{3}{2} - \frac{1}{2 \cdot m} \right) OPT$$

TSP

1 Problema 1

TSP unde muchiile au ponderea 1 sau 2.

Sa pp. ca \exists un algoritm polinomial si aproximativ $ALG \leq cOPT$ unde OPT e o rezolvare optima pentru TSP.

1.a

Fie G graful cu n noduri si ponderi 1 si 2 \rightarrow observam ca OPT e maxim $2n$.

Stim ca problema determinarii unui HC in G este NPC.

Construim G' complet, muchiile comune intre G si G' isi pastreaza costul iar celelalte muchii vor avea costul $c \cdot 2n$. Acum avem 2 cazuri:

1. daca G are un HC $\implies ALG$ va oferi un traseu de cost cel mult $c \cdot 2n$
2. daca G nu are un HC $\implies ALG$ va oferi cel mai bun traseu posibil cu cel mult $n - 1$ noduri de cost 1 si 2, iar restul de cost $c \cdot 2n$. Adica cel mai bun traseu va fi $ALG \geq (n - 1) + c \cdot 2n$

Adica putem determina in timp polinomial daca G este sau nu Hamiltonian \implies HC se poate rezolva in timp polinomial. Contradictie (HC e NPC)!

Deci aceasta varianta a TSP-ului este tot NP-hard.

1.b

Regula triunghiului $L_3 \geq L_2 \geq L_1 \implies L_3 \leq L_2 + L_1$

Deci pentru a demonstra ca regula triunghiului tine trebuie sa selectam L_3 maximul posibil si elementele L_2 si L_1 minime

Adica vom considera $L_3 = 2$ si $L_2 = 1, L_1 = 1 \implies 2 \leq 1 \leq 1$ si $2 \leq 1 + 1$ adevarate deci regula triunghiului tine in aceasta instanta.

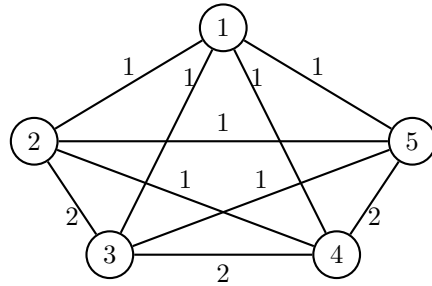
1.c

Am vazut din curs ca daca $len((u, v)) \leq len((u, w)) + len((w, v))$ atunci si relatia urmatoare are loc:

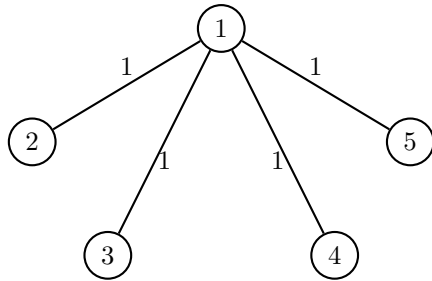
$$len((v_1, v_k)) \leq len(v_1, v_2 \dots v_k) \text{ unde } v_1, v_2 \dots v_k \text{ e un lant.}$$

Mai stim din curs ca $OPT \geq MST$

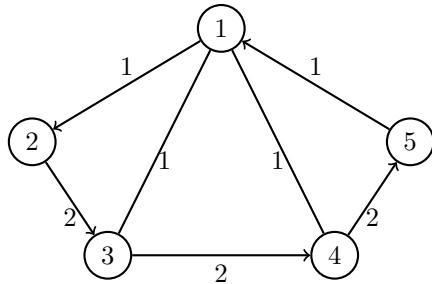
Considerand acelasi algoritm descris in curs, care se bazeaza pe un MST, sa presupunem ca avem $ALG \leq \frac{3}{2}MST \leq \frac{3}{2}OPT$. Observam ca din cauza ca MST e un lower bound pentru OPT trebuie sa aratam direct ca $ALG > \frac{3}{2}OPT$



Presupunem ca algoritmul nostru va considera urmatorul MST

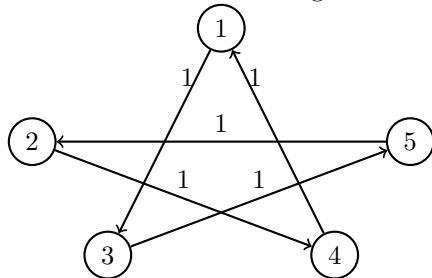


Considerand ca algoritmul din curs nu face nicio optimizare in ordinea in care selecteaza nodurile din MST , algoritmul nostru poate considera ca solutie pentru TSP urmatorul ciclu:



Deci $ALG = len(1, 2, 3, 4, 5, 1) = 1 + 2 + 2 + 2 + 1 = 8$

Dar observam ca in cazul grafului G , solutia OPT este:



Deci $OPT = len(1, 3, 5, 2, 4, 1) = 1 + 1 + 1 + 1 + 1 = 5$

Deci avem ca:

$$ALG \leq \frac{3}{2}OPT \Leftrightarrow ALG = 8 \leq \frac{3}{2}OPT = \frac{3}{2}5 = 7.5 \Leftrightarrow 8 \leq 7.5$$

Deci avem ca $ALG > \frac{3}{2}OPT$, adica ALG nu este $\frac{3}{2}$ aproximativ.

Vertex Cover

1 Problema 1

Deci avem $X = \{x_1, x_2 \dots x_n\}$ mulțimea variabilelor de tip bool si $C = \{C_1, C_2 \dots C_m\}$ cele m clauze si algoritmul:

Greedy-3CNF(C, X)

- 1: $C = \{C_1 \dots C_m\}$ mulțimea de predicate, $X = \{x_1 \dots x_n\}$ - mulțime de variabile
- 2: cât timp $C \neq \emptyset$ execută
 - 3: Alegem aleator $C_j \in C$.
 - 4: Fie x_i una dintre variabilele din C_j .
 - 5: $x_i \leftarrow \text{true}$.
 - 6: Eliminăm din C toate predicatele ce îl conțin pe x_i .
- 7: return X

1.a

În cazul algoritmului nostru considerăm ca \exists o var x_1 care se afla în toate $C_i, i = \overline{1, m}$ și în rest valori unice $x_{C_{i1}}$ și $x_{C_{i2}}$ pentru fiecare $C_i, i = \overline{1, m}$.

În acest caz algoritmul nostru poate să selecteze toate variabilele ignorând x_1 , deci problema care are $OPT = 1$, algoritmul nostru poate fi $ALG = 2m$, adică nu are o aproximare fixă, și poate lua valori foarte mari.

Deci factorul de aproximare worst case este $ALG = 2mOPT$

1.b

Pentru a obține un algoritm 3-aproximativ trebuie să modificăm linia 4, să facă toate valorile $x_i = 1, x_i \in C_j$

Greedy-3CNF(C, X)

- 1: $C = \{C_1 \dots C_m\}$ mulțimea de predicate, $X = \{x_1 \dots x_n\}$ - mulțime de variabile
- 2: cât timp $C \neq \emptyset$ execută
 - 3: Alegem aleator $C_j \in C$.
 - 4: Pentru fiecare $x_i \in C_j$.
 - 5: $x_i \leftarrow \text{true}$.
 - 6: Eliminăm din C toate predicatele ce îl conțin pe x_i .
- 7: return X

Acum trebuie sa demonstram ca $ALG \leq 3OPT$

La fiecare pas al algoritmului retinem in P_i clauzele scoase din C . Sa presupunem ca sunt k pasi.

Deci $P_i = \{C_j \in C \mid C_j \text{ a fost scos la pasul } i\}$

Notam $P = \{P_i \mid i = \overline{1, k}\}$

Este evident ca P este o partitie peste C , adica \nexists 2 clauze comune intre oricare P_i, P_j si $\bigcup_{i=1}^k P_i = C$.

Deci observam ca intre oricare 2 multimi P_i, P_j ; $\exists C_a \in P_i$ si $C_b \in P_j$ astfel incat C_a si C_b nu au niciun x in comun.

Asadar, optimul $OPT \geq |P|$ deoarece in cel mai bun caz $\forall P_i \in P$ necesita doar un $x_i = 1$.

Evdent $ALG = 3 \cdot |P|$ (adica 3^* numarul de pasi al algoritmului in care facem cate 3 variabile 1).

Asadar

$$ALG = 3 \cdot |P| \leq 3OPT \implies ALG \leq 3OPT$$

1.c

Ca sa traducem in programare liniara considerm urmatoarea situatie in programare liniara 0/1, care e chivalenta cu problema 3CNF:

minimizam $\sum_{i=1}^n x_i$

a.i. $x_a + x_b + x_c \geq 1$ unde $x_a, x_b, x_c \in C_i, i = \overline{1, m}$
 $x_i \in \{0, 1\}$

Notam $OPT_{lin_{0/1}}$ rezultatul acestei probleme de programare liniara 0/1 si fie OPT optimul pentru 3CNF. Observam ca:

$$OPT_{lin_{0/1}} = OPT$$

Dar problema de programare liniara 0/1 prezentata mai sus este NPC. Deci vom face o relaxare a problemei pentru a o duce in programare liniara cu numere reale. Notam cu x'_i valoarea lui x_i dusa in intervalul $[0, 1]$, adica $x'_i \in [0, 1]$

Deci acum avem:

minimizam $\sum_{i=1}^n x'_i$

a.i. $x'_a + x'_b + x'_c \geq 1$ unde $x_a, x_b, x_c \in C_i, i = \overline{1, m}$
 $0 \leq x'_i \leq 1$

Fie OPT_{lin} valoarea sumei $\sum_{i=1}^n x'_i$ in urma rezolvarii problemei de programare liniara prezentata mai sus. Observam ca $OPT_{lin} \leq OPT_{lin_{0/1}}$ deoarece noi am facut o relaxare a constrangerilor variantei 0/1, adica este cel putin la fel de buna ca ea.

Adica cum $OPT_{lin_{0/1}} = OPT$, obtinem:

$$OPT_{lin} \leq OPT$$

Acum avem algoritmul pentru problema 3CNF:

Prog-liniara-3CNF(C, X)

- 1: $X = \{x_1 \dots x_n\}$ - mulțime de variabile, $X' = \{x'_1 \dots x'_n\}$ - variabilele $\in [0, 1]$
- 2: Rezolva următoarea problema de programare liniară:
- 3: minimizăm $\sum_{i=1}^n x'_i$
 a.i. $x'_a + x'_b + x'_c \geq 1$ unde $x_a, x_b, x_c \in C_i, i = \overline{1, m}$
 $0 \leq x'_i \leq 1$
- 4: $X = \{x_i = 1 \mid x'_i \geq \frac{1}{3}\}$
- 5: return X

1.d

În primul rând trebuie să demonstrăm că toate clauzele $C_i \in C$ sunt satisfăcute.

Acest lucru reiese din constrângerea $x'_a + x'_b + x'_c \geq 1$, adică cel puțin o variabilă este $\geq \frac{1}{3}$ adică ia o valoare finală 1, asta pentru fiecare $C_i \in C$. Deci algoritmul rezolvă corect problema 3CNF.

Acum trebuie să demonstrăm că $ALG \leq 3OPT$

Este evident că $ALG = \sum_{i=1}^n x_i$

Deci avem că

$$OPT_{lin} = \sum_{i=1}^n x'_i$$

Și știm că $OPT_{lin} \leq OPT$ și $x'_i \geq \frac{1}{3}$ pentru orice x_i care are valoarea 1 la final. Deci:

$$ALG = \sum_{i=1}^n x_i \leq 3 \cdot \sum_{i=1}^n x'_i \leq 3 \cdot OPT_{lin} \leq 3 \cdot OPT$$

Deci avem că algoritmul nostru este 3-aproximativ:

$$ALG \leq 3OPT$$