Algoritmi Aproximativi

(1p oficiu)

Knapsack (2p)

1. Fie S un șir de numere naturale $\{s_1, s_2, \dots, s_n\}$ și K un număr natural, cu $K \geq s_i$ pentru orice i între 1 și n. Vrem să determinăm suma **maximă**, dar care să fie $\leq K$, ce poate fi formată din elementele din S (numerele pot fi luate cel mult o singură dată).

a) Scrieți un algoritm pseudo-polinomial care găsește suma optimă. Indicați complexitatea de timp/spațiu a algoritmului propus de voi și justificați de ce acesta este corect (de ce soluția găsită este optimă). (1p)

b) Scrieți un algoritm aproximativ care calculează o sumă cel puțin pe jumătate de mare ca cea optimă, dar rulează în timp $O(n)$ și complexitate spațiu $O(1)$. (1p)

Load Balance (3p)

Puteți rezolva, la alegere, cel mult 2 probleme dintre cele 3.

1. Fie o iterație a problemei *Load Balancing* (cursul 2, slide-ul 16) pentru 2 mașini. La seminarul de algoritmi aproximativi unul dintre studenți propune un algoritm de rezolvare și susține că acesta este 1.1 aproximativ. El rulează algoritmul pe un set de n activități și obține o încărcătură de 80 pe una dintre mașini, respectiv 120 pe cealaltă. Este posibil ca factorul lui de aproximare să fie corect...

a) ...ținând cont că rezultatul obținut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 100? (0,5p)

b) ...ținând cont că rezultatul obținut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 10? (0,5p)

2. Fie ALG_1 și ALG_2 doi algoritmi de rezolvare pentru aceeași problemă de **minimizare**. ALG_1 este un algoritm 2-aproximativ, respectiv ALG_2 este un algoritm 4-aproximativ (presupunem că ambii factori de aproximare sunt *tight bound*). Stabiliți valoarea de adevăr a următoarelor propoziții, dând și o scurtă justificare:

a) Există cu siguranță un input I pentru care

$$\mathrm{ALG}_2(I) \geq 2 \cdot \mathrm{ALG}_1(I)$$

(0,5p)

b) Nu există niciun input I pentru care

$$\mathrm{ALG}_1(I) \geq 2 \cdot \mathrm{ALG}_2(I)$$

(0,5p)

3. Fie algoritmul *Ordered-Scheduling Algorithm* (cursul 8, slide-ul 42), care implică algoritmul descris anterior (slide-ul 19) la care adăugăm o preprocesare cu care sortăm descrescător activitățile după timpul de desfășurare. Th. 2 afirmă că acest algoritm este $\frac{3}{2}$-aproximativ. Arătați că acest factor de aproximare poate fi îmbunătățit la $\frac{3}{2} - \frac{1}{2m}$ (unde m este numărul de calculatoare pe care se pot executa activități). (2p)

Travelling Salesman Problem (2p)

Rezolvați, la alegere, una dintre cele două probleme.

1. Considerăm o variantă a TSP, în care toate muchiile din graf au ponderea 1 sau 2.

a) Arătați că problema rămâne NP-hard pentru aceste instanțe. (1p)

b) Arătați că aceste ponderi satisfac în continuare inegalitatea triunghiului. (0p)

c) Algoritmul descris în curs (cursul 9, slide-urile 18-19) este un algoritm 2-aproximativ pentru forma generală a TSP (pentru instanțele care respectă inegalitatea triunghiului). Verificați dacă în această variantă a problemei, algoritmul din curs este chiar $\frac{3}{2}$-aproximativ. (1p)

2. Fie P o mulțime de puncte în plan. Din cursurile anterioare știm să construim un *Minimum Spanning Tree* pe baza punctelor din P. Numim acest arbore T. Uneori, adăugând și alte puncte pe lângă cele din P, putem obține un MST cu cost mai mic. Un asemenea arbore, construit prin adăugarea de noduri se numește *Steiner Tree*. Algoritmii pentru calcularea de ST-uri sunt de obicei NP-hard.

a) Arătați că există cazuri în care alegând un punct $q \notin P$ obținem un MST pentru mulțimea de puncte $P \cup \{ q \}$ cu un cost mai mic decât T. (1p)

b) Fie Q o mulțime de puncte în plan, disjunctă față de P. Arătați că T este de cel mult două ori mai mare ca și cost față de MST-ul pentru $P \cup Q$. Altfel spus, odată ce avem un MST pentru P, putem îmbunătăți rezultatul adăugând alte puncte, dar niciodată cu mai mult de un factor de 2. (1p)

Vertex Cover (2p)

Fie $X = \{ x_1, x_2, \dots, x_n \}$ o mulțime de variabile boolene. Numim formulă booleană (peste mulțimea X) în *Conjunctive Normal Form* (CNF) o expresie de forma $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ unde fiecare predicat (clauză) C_i este o disjuncție a unui număr de variabile (este alcătuit din mai multe variabile cu simbolul „$\vee$" - *logical or* - între ele).
Exemplu de astfel de expresie:

$$(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_7) \wedge (x_1 \vee x_5 \vee x_6) \wedge (x_2 \vee x_5 \vee x_7)$$

Evident că orice astfel de expresie va fi evaluată ca `true` dacă toate elementele lui X iau valoarea `true`. Ne interesează în schimb să aflăm numărul minim de elemente din X care trebuie să aibă valoarea `true` astfel încât toată expresia să fie `true`.

Fie următorul algoritm pentru problema de mai sus în varianta în care fiecare clauză are exact trei variabile (numită *3CNF*):

Greedy-3CNF

1. Fie $C = \{ C_1, \dots, C_m \}$ mulțimea de predicate, $X = \{ x_1, \dots, x_n \}$ mulțimea de variabile.
2. Cât timp $C \neq \emptyset$ execută:
   (a) Alegem aleator $C_j \in C$.
   (b) Fie x_i una dintre variabilele din C_j.
   (c) $x_i \leftarrow$ `true`
   (d) Eliminăm din C toate predicatele care îl conțin pe x_i
3. Soluția constă din variabilele pe care le-am setat ca `true` pe parcursul execuției algoritmului

Cerințe

a) Este algoritmul descris mai sus un algoritm aproximativ? În cazul afirmativ, determinați factorul de aproximare (*worst case*) al algoritmului. Altfel, justificați de ce nu este aproximativ. (0,5p)

b) Modificați algoritmul de mai sus astfel încât acesta să fie un algoritm 3-aproximativ pentru problema inițială (și justificați de ce se obține acest factor în urma modificărilor voastre). (0,5p)

c) Reformulați problema de mai sus sub forma unei probleme de programare liniară. (0,5p)

d) Dați o soluție 3-aproximativă care să rezolve problema de programare liniară formulată la subpunctul anterior. (0,5p)