Algoritmi Aproximativi (1p oficiu)

deadline - saptamana 6

Knapsack (2p):

1) ***Fie un şir de numere naturale S={s_1, s_2, ..., s_n} şi un număr natural K, cu K≥s_i pentru orice i între 1 şi n.***

   a) Să se scrie un algoritm pseudo-polinomial care găseşte suma maximă, dar care să fie ≤K, ce poate fi formată din elemente din S (numere întregi, pozitive, luate cel mult o singură dată). **(1p)**

   b) Să se găsească un algoritm aproximativ care calculează o sumă cel puţin pe jumătate de mare ca cea optimă dar rulează în timp O(n) şi complexitate spaţiu O(1). Mai exact: aveţi voie să parcurgeţi fiecare element din S cel mult o singură dată, respectiv aveţi memorie alocată doar pentru 3 variabile de tip int (dintre care una este K) + varabile de tip stream **(1p)**

Load Balance **(max 3p):**

1) Fie o iteraţie a problemei Load Balancing (Cursul 2, slide-ul 16) pentru 2 maşini. La seminarul de algoritmi aproximativi unul dintre studenţi propune un algoritm de rezolvare si sustine ca acesta este 1.1 aproximativ. El ruleaza algoritmul pe un set de *n* activitati si obtine o incarcatura de 80 pe una dintre masini, respectiv 120 pe cealalta. Este posibil ca factorul lui de aproximare sa fie corect?
   a) tinand cont ca rezultatul obtinut anterior a fost făcut pe un set de activităţi, fiecare cu timpul de lucru cel mult 100 (0.5p)
   b) tinand cont ca rezultatul obtinut anterior a fost făcut pe un set de activităţi, fiecare cu timpul de lucru cel mult 10 (0,5p)

2) Fie ALG1 si ALG2 - doi algoritmi de rezolvare pentru aceeaşi problema de **minimizare**. ALG1 este un algoritm 2-aproximativ, respectiv ALG2 este un algoritm 4-aproximativ. Stabiliti valoarea de adevar a urmatoarelor propozitii, dand si o scurta justificare.
   a) Exista cu siguranţă un input *I* pentru care *ALG2(I)≥2 ・ ALG1(I) (0,5p)*
   b) Nu exista niciun input *I* pentru care *ALG1(I)>2 ・ ALG2(I) (0,5p)*

3) Fie algoritmul Ordered-Scheduling Algorithm (Cursul 2, slide-ul 42) care implica algoritmul descris anterior (slide 19) la care adaugăm o preprocesare cu care sortăm descrescător activităţile după timpul de desfăşurare. Th. 2 afirmă că acest algoritm este 3/2 aproximativ. Arătaţi ca acest factor de aproximare poate fi îmbunătăţit la 3/2-1/(2m). *(2p)*

TSP (max 2p):

1) Fie varianta TSP unde toate muchiile au ponderea 1 sau 2.
   a) arătaţi ca problema ramane NP-hard pentru aceste instanţe (1p)
   b) arătaţi ca aceste ponderi satisfac în continuare inegalitatea triunghiului. (0p)

c) Algoritmul descris în curs (c3, slides 18-19) oferă o aproximare de ordin 2 pentru forma generala a TSP (cu regula triunghiului). ***Verificati daca in aceasta instanța a problemei, algoritmul din curs este 3/2 aproximativ!*** (1p)

2) Fie P o mulţime de puncte în plan. Din cursurile anterioare ştim să construim un MST pe baza punctelor din P. Numim acest arbore T. Uneori, adaugand şi alte puncte pe lângă cele din P, putem obţine un MST cu cost mai mic. Un asemenea arbore, construit prin adăugare de noduri se numeşte Steiner Tree. Algoritmii pt calcularea de ST-uri sunt (cel mai adesea) NP-hard.
   a) Arătati ca există cazuri în care alegand un punct $q \notin P$ MST-ul pentru mulţimea de puncte $P \cup \{q\}$ are un cost mai mic decat T (1)
   b) Fie Q o mulţime de puncte în plan, disjuncte faţă de P. Arătaţi ca T este de cel mult de 2 ori ca si cost MST-ului pentru punctele P U Q. Altfel spus, odată ce avem un MST pentru P, putem îmbunătăţi rezultatul adăugând alte puncte, dar niciodată cu mai mult de un factor de 2. (1p)

Vertex Cover (max 2p):

Fie $X=\{x_1, x_2, \ldots, x_n\}$ o mulţime de variabile de tip bool. Numim formulă booleană peste mulţimea X o formulă CNF (conjunctive normal form) o expresie de forma $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ unde fiecare predicat (clause) C_i este o disjuncţie a unui număr de variabile (e alcătuit din mai multe variabile cu simbolul $\vee$ - logical or - între ele). Exemplu de astfel de expresie:

$\mathbf{(x_1 \vee x_3 \vee x_4) \wedge (x_2 \vee x_3 \vee x_7) \wedge (x_1 \vee x_5 \vee x_6) \wedge (x_2 \vee x_5 \vee x_7)}$.

Evident că orice expresie de acest tip va fi evaluată cu "true" dacă toate elementele lui X iau valoarea true. Ne interesează în schimb, să aflăm un număr minim de elemente din X care trebuie să aibă valoarea *true* astfel încât toată expresia să fie *true.*

Fie următorul algoritm pentru problema in forma 3CNF

Greedy-3CNF(C, X)

1: $C = \{C_1, \ldots, C_m\}$ mulţimea de predicate, $X = \{x_1, \ldots, x_n\}$ - mulţime 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: $x_i \leftarrow$ true.

   6: Eliminăm din C toate predicatele ce îl conţin pe x_i .

7: return X

a) Analizaţi factorul de aproximare (worst case) al algoritmului (0,5 p)

b) Modificaţi algoritmul de mai sus, astfel încât acesta să fie un algoritm 3-aproximativ pentru problema iniţială (si justificati) (0,5p)

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

d) Daţi o soluţie 3-aproximativă pentru problema de programare liniara (0,5p)