### **Algoritmi Aproximativi (1p oficiu)**

**Knapsack (2p):**

1. *Fie un șir de numere naturale S={s1, s2, …, sn} și un număr natural K, cu K≥si pentru orice i între 1 și n.*

*a) Să se scrie un algoritm pseudo-polinomial care găsește 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 puțin pe jumătate de mare ca cea optimă dar rulează în timp O(n) și complexitate spațiu O(1). Mai exact: aveți voie să parcurgeți fiecare element din S cel mult o singură dată, respectiv aveți memorie alocată doar pentru 3 variabile de tip int (dintre care una este K) + varabile de tip stream* (1p)

**Load Balance (max 3p - probleme 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 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?
   1. tinand cont ca rezultatul obtinut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 100 (0.5p)
   2. tinand cont ca rezultatul obtinut anterior a fost făcut pe un set de activități, fiecare cu timpul de lucru cel mult 10 (0,5p)
2. Fie ALG1 si ALG2 - doi algoritmi de rezolvare pentru aceeași 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.
3. Exista cu siguranță un input *I* pentru care *ALG2(I)≥2・ALG1(I) (0,5p)*
4. Nu exista niciun input *I* pentru care *ALG1(I)>2・ALG2(I) (0,5p)*
5. Fie algoritmul Ordered-Scheduling Algorithm (Cursul 2, slide-ul 42) care implica algoritmul descris anterior (slide 19) la care adăugăm o preprocesare cu care sortăm descrescător activitățile după timpul de desfășurare. Th. 2 afirmă că acest algoritm este 3/2 aproximativ. Arătați ca acest factor de aproximare poate fi îmbunătățit la 3/2-1/(2m). *(2p)*

**TSP (max 2p - de ales una dintre cele 2 probleme):**

1. Fie varianta TSP unde toate muchiile au ponderea 1 sau 2.
   1. arătați ca problema ramane NP-hard pentru aceste instanțe (1p)
   2. arătați ca aceste ponderi satisfac în continuare inegalitatea triunghiului. (0p)
   3. 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 mulțime de puncte în plan. Din cursurile anterioare știm să construim un MST pe baza punctelor din P. Numim acest arbore T. Uneori, adaugand și alte puncte pe lângă cele din P, putem obține un MST cu cost mai mic. Un asemenea arbore, construit prin adăugare de noduri se numește Steiner Tree. Algoritmii pt calcularea de ST-uri sunt (cel mai adesea) NP-hard.
   1. Arătati ca există cazuri în care alegand un punct q ∉P MST-ul pentru mulțimea de puncte PU{q} are un cost mai mic decat T (1)
   2. Fie Q o mulțime de puncte în plan, disjuncte față de P. Arătați 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 îmbunătăți rezultatul adăugând alte puncte, dar niciodată cu mai mult de un factor de 2. (1p)

**Vertex Cover (2p):**

Fie X={x1, x2, …, xn} o mulțime de variabile de tip bool. Numim formulă booleană peste mulțimea X o formulă CNF (conjunctive normal form) o expresie de forma C1∧C2∧...∧Cm unde fiecare predicat (clause) Ci este o disjuncție a unui număr de variabile (e alcătuit din mai multe variabile cu simbolul ∨ - logical or - între ele).

Exemplu de astfel de expresie:

**(x1 ∨ x3 ∨ x4) ∧ (x2 ∨ x3 ∨ x7) ∧ (x1 ∨ x5 ∨ x6)∧ (x2 ∨ x5 ∨ x7).**

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 = {C1, . . . , Cm} mulțimea de predicate, X = {x1, . . . , xn} - mulțime de variabile

2: cât timp C ≠ ∅ execută

3: Alegem aleator Cj ∈ C.

4: Fie xi una dintre variabilele din Cj .

5: xi ← true.

6: Eliminăm din C toate predicatele ce îl conțin pe xi .

7: return X

1. Analizați factorul de aproximare (worst case) al algoritmului (0,5 p)
2. Modificați algoritmul de mai sus, astfel încât acesta să fie un algoritm 3-aproximativ pentru problema inițială (si justificati) (0,5p)
3. Reformulati problema de mai sus sub forma unei probleme de programare liniară (0,5p)
4. Dați o soluție 3-aproximativă pentru problema de programare liniara (0,5p)