

## Algoritmi Aproximativi (1p oficiu)

deadline - saptamana 6

Knapsack (2p):

- 1) Fie un șir de numere naturale  $S=\{s_1, s_2, \dots, s_n\}$  și un număr natural  $K$ , cu  $K \geq s_i$  pentru orice  $i$  între 1 și  $n$ .
  - a) Să se scrie un algoritm pseudo-polinomial care găsește suma maximă, dar care să fie  $\leq K$ , 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  $K$ ) + variabile de tip stream (1p)

Load Balance (max 3p):

- 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 ALG1 și ALG2 - doi algoritmi de rezolvare pentru aceeași problema de **minimizare**. ALG1 este un algoritm 2-aproximativ, respectiv ALG2 este un algoritm 4-aproximativ. Stabiliți valoarea de adevăr a următoarelor propoziții, dând și o scurtă justificare.
  - a) Există cu siguranță un input  $I$  pentru care  $ALG2(I) \geq 2 \cdot ALG1(I)$  (0,5p)
  - b) Nu există niciun input  $I$  pentru care  $ALG1(I) > 2 \cdot ALG2(I)$  (0,5p)
- 3) Fie algoritmul Ordered-Scheduling Algorithm (Cursul 2, slide-ul 42) care implică 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 că acest factor de aproximare poate fi îmbunătățit la  $3/2 - 1/(2m)$ . (2p)

TSP (max 2p):

- 1) Fie varianta TSP unde toate muchiile 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 (c3, slides 18-19) oferă o aproximare de ordin 2 pentru forma generală a TSP (cu regula triunghiului). **Verificați dacă în această instanță 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, adăugând 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.
- Arătați că există cazuri în care alegând un punct  $q \notin P$  MST-ul pentru mulțimea de puncte  $P \cup \{q\}$  are un cost mai mic decât  $T$  (1)
  - Fie  $Q$  o mulțime de puncte în plan, disjuncte față de  $P$ . Arătați că  $T$  este de cel mult de 2 ori ca și cost MST-ului pentru punctele  $P \cup Q$ . 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 (max 2p):

Fie  $X = \{x_1, x_2, \dots, x_n\}$  o mulțime de variabile de tip bool. Numim formulă booleană peste mulțimea  $X$  o formulă CNF (conjunctive normal form) o expresie de forma  $C_1 \wedge C_2 \wedge \dots \wedge C_m$  unde fiecare predicat (clause)  $C_i$  este o disjuncție a unui număr de variabile (e 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 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 în forma 3CNF

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  $X$

- Analizați factorul de aproximare (worst case) al algoritmului (0,5 p)
- Modificați algoritmul de mai sus, astfel încât acesta să fie un algoritm 3-aproximativ pentru problema inițială (și justificați) (0,5p)
- Reformulați problema de mai sus sub forma unei probleme de programare liniară (0,5p)
- Dați o soluție 3-aproximativă pentru problema de programare liniară (0,5p)