Tema Algoritmi Avansati

Eduard-Valentin Dumitrescul, grupa 232

March 19, 2024

1 Algoritmi Aproximativi

1.1 Knapsack

Fie S un șir de numere naturale $s_1, s_2, ..., s_n$ și K un număr natural, cu $K \geq s_i$ pentru orice i între 1 și n.

a) Scrieți un algoritm pseudo-polinomial care găsește suma maximă, dar care să fie $\leq K$, ce poate fi formată din elementele din S (numere întregi, pozitive, luate cel mult o singură dată). 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)

Rezolvare: a)

Complexitate: O(N*K) timp si O(K) spatiu

```
sum_reached[0] = 1    #suma 0 se obtine in mod implicit
for i=1 to k:
  sum_reached[i] = 0 #marcam faptul ca inca nu am obtinut suma i

#luam, pe rand, fiecare valoare din sirul S si incercam sa obtinem noi sume pe
    ↪ baza celor deja obtinute
for i=1 to n:
  for sum=k to 0:
    if sum_reached[sum] = 1 and i + sum <= k:
      sum_reached[i+sum] = 1

maximum_sum = arg(max(i | i∈{0,1,...,k}, sum_reached[i] = 1))
```

Stim ca suma 0 se poate obtine mereu (nealegand niciun numar din sir).

Presupunem ca am gasit toate sumele $\leq K$ ce pot fi obtinute adunand numere din $s_1, s_2, ..., s_{i-1}$. Acum, orice suma ce il contine pe s_i si este formata cu numerele $s_1, s_2, ..., s_i$ este de forma $s_i + S_{i-1}$, unde S este o suma obtinuta din numerele $s_1, s_2, ..., s_{i-1}$. Stiind ca am gasit toate astfel de S_{i-1} rezulta ca vom gasi si toate sumele de forma S_i. Inductiv, vom gasi toate sumele $S_n \leq K$ posibile formate din $s_1, s_2, ..., s_n$.

Rezolvare: b)

Complexitate: O(N) timp si O(1) spatiu

```
Fie S lista de numere

sum = 0
maxValue = 0
for each s in S:
  if sum + s <= K:
    sum += s
  if max < s:
    maxValue = s

if maxValue >= K/2:
  ALG() = maxValue
else:
  ALG() = sum
```

OPT = valoarea solutiei optime
vmax = valoarea maxima din S
Daca $vmax \geq OPT/2$, atunci $ALG = vmax \geq OPT/2$, deci $ALG \geq 1/2 * OPT$.
Daca $vmax < OPT/2$, atunci fiecare numar din S este $\leq OPT/2$. Daca algoritmul va insuma toate numerele, atunci obtine solutia optima. Altfel, fie primul numar s_i pe care nu il aduna si sum, suma obtinuta pana in acel punct.
$sum + s_i > K \geq OPT \Rightarrow sum > OPT - s_i$ (1)
$s_i < OPT/2 \Rightarrow OPT - s_i > OPT/2$ (2)
$(1) si (2) \Rightarrow sum > OPT/2$
$ALG \geq sum \Rightarrow ALG > OPT/2$
Asadar, ALG ete 1/2-aproximativ.

Exemplu in care $ALG \approx OPT/2$:

$S = \{10, 1, 2, 3, 4\}, K = 20$
$ALG = 10$
$OPT = 20$
$ALG/OPT = 2$

1.2 Load-Balancing - 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. Stabiliți valoarea de adevăr a următoarelor propoziții, dând și o scurtă justificare:

a) Există cu siguranță un input I pentru care $ALG_2(I) \geq 2 \cdot ALG_1(I)$ (0,5p)
b) Nu există niciun input I pentru care $ALG_1(I) \geq 2 \cdot ALG_2(I)$ (0,5p)

Rezolvare: a)

Nu este adevarat. Daca pentru un anume I, amandoi algoritmii dau solutia optima, relatia nu este adevarata.

Rezolvare: b)

Nu este adevarat.
ALG_1 este 2-aproximativ $\Rightarrow ALG_1 \leq 2 \cdot OPT$
Presupunem ca exista I pentru care $ALG_1 \geq 2 \cdot ALG_2$, deci
$2 \cdot OPT \geq ALG_1 \geq 2 \cdot ALG_2$, de unde
$OPT \geq ALG_2$.
Asadar, pentru $ALG_2 = OPT$ si $ALG_1(I) = 2 \cdot OPT(I)$ inegalitatea are loc.

1.3 Load-Balancing - 3

Fie algoritmul Ordered-Scheduling Algorithm (cursul 2, 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)

Rezolvare

Fie k indicele masinii cu loadul maxim in urma executarii algoritmului.
$ALG = load(k)$
Fie q ultima activitate adaugata pe masina k
Fie $load'(i)$ load-ul masinii i fix inainte ca activitatea q sa fie asociata masinii k.
$ALG = load(K) = load'(k) + t_q$
Conform justificarii de la curs:

$$load'(k) + t_q \leq \frac{1}{m} \sum_{i=1}^{n} t_i + t_q$$

Dar, faptul ca adaugam activitate q la masina k ne spune ca $load`(k)$ este min(load`). Astfel, putem imbunatati inegalitatea:

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

Daca $q \leq m$ atunci q va fi pusa peste o masina goala, deci $ALG = OPT$ Daca $q > m$ atunci:

$$\begin{aligned} load'(k) + t_q &\leq \frac{1}{m} \sum_{i=1}^{q-1} t_i + t_q = \frac{1}{m} \sum_{i=1}^{q} t_i + (1 - \frac{1}{m}) t_q \\ &\leq \frac{1}{m} \sum_{i=1}^{n} t_i + (1 - \frac{1}{m}) \cdot \frac{1}{2}(t_m + t_{m+1}) \\ &\leq OPT + \frac{1}{2} OPT - \frac{1}{2m} = (\frac{3}{2} - \frac{1}{2m}) OPT \end{aligned}$$

1.4 Traveling Salesman Problem - 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. (2p)

Rezolvare: a)

Fie $P = \{p_1 = (0,0), p_2(0,2), p_3 = (2,0), p_4 = (2,2)\}$ si $q = (1,1)$.
Costul lui T este

$$dist(p1, p2) + dist(p1, p3) + dist(p2, p4) = 2 + 2 + 2 = 6$$

Costul arborelui partial de cost minim pe baza punctelor $P \cup \{q\}$ este

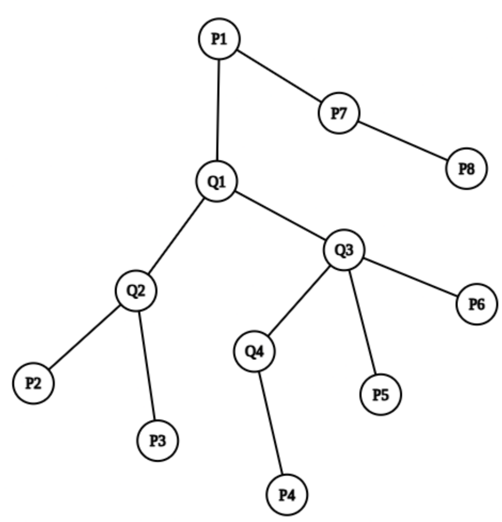
$$\begin{gathered} dist(p1, q) + dist(p2, q) + dist(p3, q) + dist(p4, q) = \\ = \sqrt{2} + \sqrt{2} + \sqrt{2} + \sqrt{2} = \\ = 4 * sqrt2 \approx 5.656 \end{gathered}$$

Asadar, acesta este un caz in care alegând un punct $q \notin P$ obținem un MST pentru mulțimea de puncte $P \cup \{q\}$ cu un cost mai mic decât T.

Rezolvare: b)

Idee: Pornind de la MST-ul asociat punctelor $P \cup Q$ de cost C, construim un arbore partial asociat punctelor P cu costul cel mult $2 \cdot C$. Asadar, MST-ul asociat punctelor P va avea cost cel mult dublu, ceea ce trebuie demonstrat.
Observatie 1: Deoarece P si Q sunt multimi de puncte in plan, regula triunghiului este valabila.
Fie A MST-ul asociat punctelor $P \cup Q$, de cost C.
Fie B arborele partial asociat punctelor P pe care urmeaza sa-l construim.
Observatie 2: Nu are sens ca un punct $q \in Q$ sa fie frunza in A, deoarece, prin eliminarea lui, obtinem un nou MST, de cost mai mic, asociat punctelor $P \cup Q \setminus \{q\}$. Asadar, vom presupune, fara a pierde generalitatea, ca nu exista astfel de puncte.
De asemenea, consideram cu $p \in P$ este radacina arborelui A.

Scriem parcurgerea Euleriana a arborelui A.



Pentru arborele din imagine, parcurgerea Euleriana este:
P1, Q1, Q2, P2, Q2, P3, Q2, Q1, Q3, Q4, P4, Q4, Q3, P5, Q3, P6, Q3, Q1, P1, P7, P8, P7, P1
Arborele nostru partial B va fi construit in asa fel incat parcurgerea lui Euleriana sa fie identica cu cea a arborelui A, dupa ce eliminam punctele din Q.
Pentru acest exemplu, parcurgerea Euleriana a arborelui B va fi: P1 P2 P3 P4 P5 P6 P7 P8 P7 P1
Pentru orice lant (cu eventualele renumerotari) $P1 - Q1 - Q2 - ... - Qn - P2$ avem:

$$dist(P1, P2) \leq dist(P1, Q1) + dist(Q1, Q2) + ... + dist(Qn - 1, Qn) + dist(Qn, P2)$$

Deci, suma distantelor SB dintre toate perechile de puncte consecutive din parcurgerea Euleriana a lui B este mai mica sau egala cu suma dinstantelor SA dintre toate perechile de puncte consecutive din parcurgerea Euleriana a lui A. Cum SA este egal cu lungimea conturului arborelui A, avem:

$$SB \leq SA = 2 \cdot C$$

Construim graful G in care exista muchie intre Px si Py, daca Px si Py se afla pe pozitii consecutive in parcurgerea Euleriana a lui B. Evident, obtinem chiar arborele B.

$$costMST(B) \leq \text{suma costurilor muchiilor lui B} = SB \leq 2 \cdot C$$

Asadar, am demonstrat ca, prin adaugarea de noi puncte, putem imbunatati MST-ul lui P cu un facor cel mult 2.

1.5 Vertex Cover **(2p)**

Fie $X = \{x_1, x_2, ..., x_n\}$ o mulțime de variabile boolene. Numim formulă booleană (peste mulțimea X) în Conjunctive Normal Form (CNF) o expresie de forma $C1 \wedge C2 \wedge ... \wedge Cm$ 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: $(x1 \vee x3 \vee x4) \wedge (x2 \vee x3 \vee x7) \wedge (x1 \vee x5 \vee x6) \wedge (x2 \vee x5 \vee x7)$

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 $C = \{C_1, ..., C_m\}$ mulțimea de predicate, $X = \{x_1, ..., x_n\}$ mulțimea de variabile.
2. Cât timp $C \neq \phi$ execută:
(a) Alegem aleator $Cj \in C$.
(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

Cerinte

a) Analizați factorul de aproximare (worst case) al algoritmului. (0,5p)
b) Modificați algoritmul de mai sus, astfel încât acesta să fie un algoritm 3-aproximativ pentru problema inițială (și justificați). (0,5p)
c) Reformulați problema de mai sus sub forma unei probleme de programare liniară (cu numere reale). (0,5p)
d) Dați o soluție 3-aproximativă pentru problema de programare liniară formulată la subpunctul anterior. (0,5p)

Rezolvare: a

Fie expresia:

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_4) \wedge .. \wedge (x_1 \vee x_2 \vee x_n)$$

Pe scurt, doua variabile apar in fiecare predicat, iar a treia variabila este diferita pentru fiecare predicat.
Algoritmul Greedy-3CNF are sansa ca de fiecare data sa aleaga din fiecare predicat exact variabila care apare o singura data in toata formula. Astfel, solutia data de algoritm va fi $\{x_3, x_4, ..., x_n\}$, pe cand solutia optima este $\{x_1\}$.
Asadar, algoritmul Greedy-3CNF ne poate oferi o solutie de (n-2) ori mai slaba decat cea optima.

Rezolvare: b

Vom modifica pasul (b) astfel incat sa alegem toate cele 3 variable din predicatul C_j
Vom spune ca doua predicate sunt complet distincte daca nu exista nicio variabila care sa apara in amandoua.
Notam C^* o multime in care toate predicatele sunt complet distincte doua cate doua.
Aratam ca $OPT \geq |C^*|$:
Fiecare variabila aleasa de solutia OPT poate face parte din cel mult un predicat din C^*, deci $OPT \geq |C^*|$.
Aratam ca $ALG = 3 \cdot |C^*|$:
Observam ca algoritmul nostru alege mereu un nou predicat C_j complet distinct de cele alese pana atunci. Astfel, multimea acestor predicate poate fi notata C^*, avand proprietatea ca oricare doua predicate din ea sunt complet distincte. Din faptul ca ALG, pentru fiecare predicat din C^*, alege cate 3 variabile, ajungem la concluzia ca $ALG = 3 \cdot |C^*|$
Deci $3 \cdot OPT \geq 3 \cdot |C^*| = ALG$

Rezolvare: c

Fie $X = \{x_1, x_2, ..., x_n\}$, cu $0 \leq x_i \leq 1$.
Vrem sa minimizam $S = x_1 + x_2 + ... + x_n$.
Pentru fiecare $C_i = x_m \vee x_n \vee x_p$ avem constrangerea $x_m + x_n + x_p \geq 1$.

Rezolvare: d

Folosindu-ne de problema de programare liniara de la punctul c, facem urmatoarea modificare. La final, daca $x_i \geq 1/3$, spunem ca variabila este adevarata, altfel este falsa.
Astfel, S devine:

$$S_{intreg} = \sum_{i=1}^{n} v_i, \text{ unde } v_i = \begin{cases} 1 & \text{pentru } x_i \geq 1/3 \\ 0 & \text{altfel} \end{cases} \tag{1}$$

$$S_{intreg} \leq \textstyle\sum_{i=1}^{n} 3 \cdot v_i = 3 \cdot S_{real}$$

Deci solutia este 3-aproximativa.