TAP
Curs 2: METODA GREEDY

Tehnici avansate de programare

Lect.dr. Iulia Banu
Departamentul de Informatică,
Universitatea din București

semestrul 1, 2019

Rezumat curs Greedy

1. Prezentare generală
2. Algortimul general
3. Variante de demonstrare a corectitudinii
4. Exemple
   - Memorarea textelor pe bandă
   - Probleme de planificare
   - Problema rucsacului
   - Probleme de packing
   - Probleme de clustering

Prezentare generală

- Aplicabilă problemelor de optim
- Alege soluția optimă sau o aproximare a optimului $\Rightarrow$ **necesită demonstrarea corectitudinii**
- La fiecare pas se face o alegere optimă pentru o subproblemă a probelmei inițiale $\Rightarrow$ **optim local**.
- Evită generarea tuturor soluțiilor posibile (timp de calcul exponențial) $\Rightarrow$ **algoritmi rapizi**.
- Aplicabilă unor probleme pentru care nu sunt cunoscuți algoritmi polinomiali.

Algoritmul general

Considerăm mulțimea finită $A = \{a_1, \ldots, a_n\}$
și o proprietate *check* : $\mathcal{P}(A) \to \{0, 1\}$ astfel încât:

$check(\emptyset) = 1$

$\forall Y \subset X \; check(X) \Rightarrow check(Y)$

Trebuie aleasă ca soluție o submulțime X pentru care $check(X) = 1$ astfel încât să se optimizeze o funcție de cost

$f : \mathcal{P}(A) \to \mathbb{R}$.

Optimizarea funcției f este "ascunsă" în implementarea funcțiilor *choose* și *prel*

Algoritmul general

$S \leftarrow \emptyset$
for i=1,n
    $x \leftarrow choose(A)$;
    $A \leftarrow A \setminus \{x\}$;
    if $check(S \cup \{x\}) = 1$
        $S \leftarrow S \cup \{x\}$

$prel(A)$
$S \leftarrow \emptyset$
for i=1,n
    if $check(S \cup \{a_i\}) = 1$
        $S \leftarrow S \cup \{a_i\}$

Variante de demonstrare a corectitudinii

- Se analizează proprietățile, particularitățile soluțiilor optime.

Exemplu

Submulțime de sumă maximă dintr-un vector de numere întregi.

- Prin reducere la absurd, se presupune că ar exista o soluție optimă care diferă de soluția Greedy (se alege o permutare a soluției Greedy) și se ajunge la o contradicție.

Exemple

Memorarea textelor pe bandă, problema rucsacului, probleme de planificare.

Variante de demonstrare a corectitudinii

- Inductiv:
  pasul 1: se arată că există o soluție optimă care conține primul element selectat de algoritmul Greedy;
  pasul 2: se demonstrează optimalitatea submulțimii selectate de algoritmul Greedy pentru subproblema obținută prin eliminarea primul element selectat de algoritmul Greedy $\Rightarrow$ se poate aplica pentru această submulțime pasul 1.

- Pentru probleme de numărare, se alege o funcție (bijectivă) între mulțimea elementelor unei soluții optime și mulțimea elementelor alese de algoritmul Greedy.
  (funcție 1:1, 2:1 etc.)

Exemplu

Probleme de planificare (problema spectacolelor)

Memorarea textelor pe bandă

Se dau n texte cu lungimile L(1),...,L(n) ce urmează sa fie așezate pe o bandă. Pentru a citi textul de pe poziția k, trebuie citite textele de pe pozițiile $1, 2, ..., k$ (conform specificului accesului secvențial pe bandă).

Problema: Să se determine o modalitate de așezare a textelor pe bandă astfel încât timpul mediu de acces să fie minimizat.

Dacă σ este o permutare a celor n texte atunci timpul mediu de acces este:

$$T(\sigma) = 1/n \sum_{i=1}^{n} \{L(\sigma(1)) + \ldots L(\sigma(i))\}$$

Soluție: Se sortează textele crescător după lungime.

Soluție: Se alege o permutare g în care textele sunt sortate după lungime:

$$i < j \Rightarrow L(g(i)) < L(g(j)).$$

Demonstrarea corectitudinii: Se alege, prin reducere la absurd, o permutare σ optima, diferită de permutarea Greedy cu număr maxim de inversiuni. Se obține, printr-o inversiune convenabilă aplicată lui σ, o permutare τ astfel încat:

$$T(\sigma) > T(\tau).$$

τ va fi o permutare optimă având mai puține inversiuni decât are σ. Contradicție cu alegerea lui σ optimă.

Probleme de planificare

Problema P1: Se presupune ca A este o mulțime de n activitati. Fiecare activitate i are un timp de start s_i și un timp de terminare f_i.
Să se selecteze o mulțime de activități compatibile (intervalele de desfășurare disjuncte) de cardinal maxim.

Problema P2: Fiecare activitate i are un timp de start s_i, un timp de terminare f_i și aparține unei clase de activități c_i.
Să se selecteze o mulțime de activități compatibile (intervalele de desfășurare disjuncte două câte două, oricare două activități aparțin unor clase distincte) de cardinal maxim.

Probleme de planificare

Soluție: se sortează activitățile după timpul de final, apoi la fiecare pas $i = 1, n$ este selectată activitatea i dacă este compatibilă cu activitățile deja selectate, i.e dacă timpul de început al activității i este mai mare decât timpul de final al ultimei activități planificate până la pasul i.

Problema 1: Este demonstrată corectitudinea.

Problema 2: Este demonstrat că numărul de clase selectat este în cel mai rău caz, de două ori mai mic decât soluția optimă. (aproximare 2:1)

Problema rucsacului, varianta fracționară(continuă)

Se cunosc G, greutatea unui rucsac și $g = (g_1, \dots g_n)$, $c = (c_1, \dots c_n)$ greutățile, respectiv costurile a n obiecte. Fiecare obiect poate fi încărcat parțial în rucsac. c_i este costul obținut dacă obiectul i este încărcat în întregime în rucsac.

Cerință:
Încărcare optimă, cost maxim, greutatea totală încărcată nu depășeste G.

Fie o soluție $o = (o_1, \dots o_n)$ unde o_i este cantitatea încărcată din obiectul i, $o_i \in [0, 1]$ $\forall i = 1, n$

$$C_o = \sum_{i=1}^{n} o_i * (c_i/g_i)$$

.

Problema rucsacului, varianta fracționară(continuă)

Soluție: Se ordonează obiectele după profitul obținut pe unitate

$$c_1/g_1 >= c_2/g_2 \cdots >= c_n/g_n.$$

Se alege $a = (g_1, g_2 \ldots, g_{u-1}, a_u, 0, \ldots 0)$ cu $a_u \in [0, g_u]$ astfel încât rucsacul va fi complet ocupat.

Demonstrarea corectitudinii: Prin reducere la absurd se alege $o = (o_1, \ldots o_n)$ optimă astfel încât să difere pe cel puțin o poziție de soluția Greedy. Modificând unele componente ale lui o obținem o' astfel încât

$$C_o < C_{o'}$$

Set cover

Problema P1: Fie S o mulțime cu n elemente și $S_1, S_2, \ldots S_m$, submulțimi ale lui U. Să se aleagă un număr minim de submulțimi astfel încât

$$\cup_{i \in P} S_i = U \ .$$

Soluție: Se alege la pasul i submulțimea cu cel mai mare număr de elemente care nu se regăsesc în mulțimile selectate până la pasul $i - 1$.

Aproximare O(log n).

Probleme de clustering

Sunt date:

- O mulțime O de n obiecte
- O distanță între obiecte $D : O \times O \to \mathbb{R}$
- Un număr $k \in \mathbb{N}$.

Cerință: Să se împartă cele n obiecte în k clustere (partiționare) astfel încât să fie îndeplinite criteriile de optimalitate.

Optimalitate (variante):

- Separare maximă: Distanța dintre clustere să fie cât mai mare.
- k-medii: Distanțele de la obiecte la centrele clusterelor să fie cât mai mică.

Clustering - Separare maximă

Fie $\mathcal{C} = \{\mathcal{C}_1 \ldots \mathcal{C}_k\}$ o împărțire în k-clustere, i.e o partiție a mulțimii $\{1 \ldots n\}$.

$$cost(\mathcal{C}) = \min_{p \neq q} \{D(p,q) | p \in \mathcal{C}_i \wedge q \in \mathcal{C}_j, i \neq j\}$$

$$\max_{\mathcal{C}} cost(\mathcal{C}) = \max_{\mathcal{C}} \min_{p \neq q} \{D(p,q) | p \in \mathcal{C}_i \wedge q \in \mathcal{C}_j, i \neq j\}$$

Se alege partiționarea care maximizează *cost*: distanțele dintre puncte situate în cele mai apropiate clustere să fie cât mai mare.

Soluție: Algoritmul lui Kruskal oprit după formarea a k componente conexe.

Clustering - k medii

Fie $\mathcal{C} = \{\mathcal{C}_1 \ldots \mathcal{C}_k\}$ o împărțire în k-clustere, i.e o partiție a mulțimii $\{1 \ldots n\}$ și $\{\mu_1 \ldots \mu_n\}$ centrele clusterelor.

$$cost(\mathcal{C}) = \sum_{i=1}^{k} \sum_{o \in \mathcal{C}_i} D(o, \mu_i)^2$$

Se alege partiționarea care minimizează *cost* (suma pătratelor distanțelor de la puncte la centrele clusterelor).

Soluție: Algoritmul Lloyd: 1) se inițializează k centre. 2) Se împart punctele în clustere în funcție de distanțele față de centre. 3) se recalculează centrele. Se repetă pașii 2 și 3 până se stabilizează centrele.