

Metoda Greedy

Prezentare [1]

Metoda Greedy (**greedy**=lacom) este aplicabilă problemelor de optim.

Considerăm mulțimea finită $A = \{a_1, \dots, a_n\}$ și o proprietate p definită pe mulțimea submulțimilor lui A (notată $P(A)$):

$$p: P(A) \rightarrow \{0, 1\} \text{ cu } \begin{cases} p(\emptyset) = 1 \\ p(X) \Rightarrow p(Y), \forall Y \subset X \end{cases}$$

O submulțime $S \subset A$ se numește **soluție** dacă $p(S) = 1$.

Dintre soluții dorim să alegem una care optimizează o funcție $f: P(A) \rightarrow \mathbb{R}$ dată.

Metoda urmărește evitarea parcurgerii tuturor submulțimilor (ceea ce ar necesita un timp de calcul exponențial), **mergându-se "direct" spre soluția optimă**. Nu este însă garantată obținerea unei soluții optime; de aceea aplicarea metodei Greedy trebuie însoțită neapărat de o demonstrație.

Distingem două variante generale de aplicare a metodei Greedy: fără/cu prelucrare inițială:

$S \leftarrow \emptyset$ for $i=1, n$ $x \leftarrow \text{alege}(A)$; $A \leftarrow A \setminus \{x\}$ if $p(S \cup \{x\}) = 1$ $S \leftarrow S \cup \{x\}$	$\text{prel}(A)$ $S \leftarrow \emptyset$ for $i=1, n$ if $p(S \cup \{a_i\}) = 1$ $S \leftarrow S \cup \{a_i\}$
---	---

Observații:

- în algoritm nu apare funcția f
- dificultatea constă în a concepe procedurile alege , respectiv prel , în care este "ascunsă" funcția f .

Exemplul 1 Se consideră mulțimea de valori reale $A = \{a_1, \dots, a_n\}$. Să se determine o submulțime a lui A a cărei sumă a elementelor este maximă.

Soluție. Vom parcurge mulțimea și vom selecta numai elementele pozitive. Dacă nu există elemente pozitive, soluția este dată de cel mai mare element din mulțime. Corectitudinea soluției este evidentă.

Exemplul 2 (Contraexemplu) Se consideră mulțimea $A = \{a_1, \dots, a_n\}$ cu elemente pozitive. Să se determine o submulțime a lui A de sumă maximă, dar cel mult egală cu o valoare M dată.

Dacă alegem la fiecare pas cel mai mare element care poate fi adăugat la soluție fără a depăși M , pentru $A = \{6, 3, 4, 2\}$ și $M=7$ obținem $\{6\}$. Dar soluția optimă este $\{3, 4\}$ cu suma egală cu 7.

Continuăm cu prezentarea unor exemple clasice.

• Problema selectării activităților (problema spectacolelor) [2]

Să presupunem că avem o mulțime $S = \{1, 2, \dots, n\}$ de n activități (spectacole) care doresc să folosească aceeași resursă (sala de spectacole). Această resursă poate fi folosită de o singură activitate la un moment dat. Fiecare activitate i are un timp de start s_i și un timp de terminare t_i . Problema selectării activităților (spectacolelor) constă în selectarea unei mulțimi de activități compatibile între ele de cardinal maxim (compatibile = având intervalele de desfășurare disjuncte).

Variantă de enunț: Știind intervalele de desfășurare a unor conferințe la care dorim să asistăm în timpul unei zile (sau emisiuni pe care dorim să le vizionăm), să se determine numărul maxim de conferințe la care putem participa (= cu intervale de desfășurare disjuncte).

Soluție. Strategia Greedy pentru rezolvarea acestei probleme este următoarea: la fiecare pas este selectată o activitate nouă care se termină cel mai **devreme (cu timpul de terminare cel mai mic) compatibilă cu activitățile deja selectate**. Pentru aceasta vom ordona activitățile crescător după timpul de terminare.

Să presupunem că activitățile sunt ordonate crescător după timpul de terminare: $t_1 \leq t_2 \leq \dots \leq t_n$

Algoritmul Greedy este descris de următoarea funcție, scrisă în pseudocod.

```
function selectie_activitati(s, t)
    S ← {1}
    us ← 1
    for ac = 2, n
        if  $s_{ac} \geq t_{us}$ 
            S ← S ∪ {ac}
            us ← ac
    return S
```

În mulțimea S se introduc activitățile care au fost selectate. Variabila us indică ultima activitate introdusă în S .

Deoarece activitățile sunt considerate în ordinea crescătoare a timpilor lor de terminare, t_{us} va reprezenta timpul maxim de terminare a oricărei activități din S :

$$t_{us} = \max\{t_k, k \in S\}$$

De aceea, pentru a vedea dacă activitatea curentă ac este compatibilă cu toate celelalte activități existente la momentul curent în S este suficient ca $s_{ac} \geq t_{us}$.

Corectitudine. Presupunem (ca și în algoritm) că activitățile sunt ordonate crescător după timpul de terminare: $t_1 \leq t_2 \leq \dots \leq t_n$. Fie $AS = \{g_1=1, \dots, g_t\}$ (cu $g_1 < \dots < g_t$) soluția Greedy.

Varianta 1 [2]:

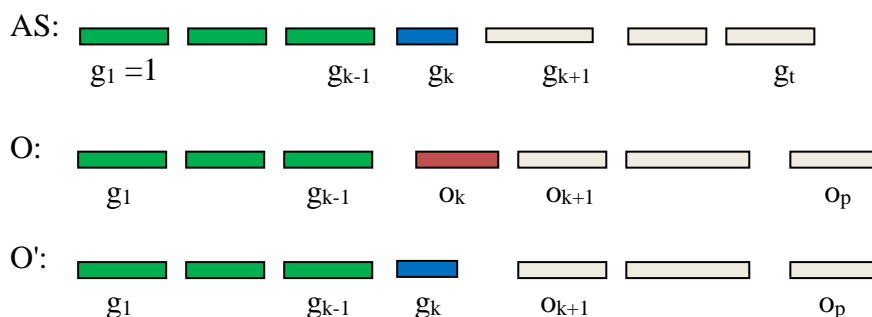
Considerăm o soluție optimă $O = \{o_1, \dots, o_p\}$ (cu activitățile notate astfel încât $o_1 < \dots < o_p$, deci $t_{o_1} \leq \dots \leq t_{o_p}$) care are **un număr maxim de elemente (inițiale) în comun cu soluția Greedy AS** . Avem $t \leq p$. Presupunem $O \neq AS$.

Atunci există un indice $k \leq t$ astfel încât $g_k \neq o_k$, altfel am avea $g_1 = o_1, \dots, g_t = o_t$, adică $AS \subseteq O$ și $t < p$. Dar, deoarece activitatea o_{t+1} este compatibilă cu toate activitățile din $\{o_1, \dots, o_t\} = \{g_1, \dots, g_t\}$, algoritmul `selectie_activitati` ar mai fi avut activități compatibile cu $\{g_1, \dots, g_t\}$ din care să selecteze, deci ar fi furnizat o soluție cu mai multe elemente.

Fie $k \leq t$ cel mai mic indice astfel încât $g_k \neq o_k$ (prima poziție pe care soluția greedy AS și soluția optimă O diferă). La pasul la care `selectie_activitati` a ales activitatea g_k , activitatea o_k era de asemenea neselectată și compatibilă cu activitățile deja selectate $g_1 = o_1, \dots, g_{k-1} = o_{k-1}$. Deoarece activitatea g_k a fost cea selectată, rezultă că $tg_k \leq to_k$ (g_k a fost selectată deoarece avea timpul de terminare mai mic), deci g_k este compatibilă cu o_{k+1}, \dots, o_t (deoarece acestea încep după ce se termină o_k) – v. desenul de mai jos:

$$tg_k \leq to_k \leq so_{k+1} \leq to_{k+1} \leq \dots \leq so_p \leq to_p$$

Atunci putem înlocui în soluția optimă O activitatea o_k cu g_k și obținem tot o soluție posibilă $O' = O - \{o_k\} \cup \{g_k\}$ (formată din activități compatibile din S). Avem $|O'| = |O|$, deci O' este tot o soluție optimă, dar care are mai multe elemente în comun cu soluția greedy AS, contradicție.



Varianta 2 (similar) – Demonstrăm prin inducție după n că algoritmul greedy `selectie_activitati` construiește o soluție optimă.

Pentru $n = 0, 1$ afirmația este evidentă.

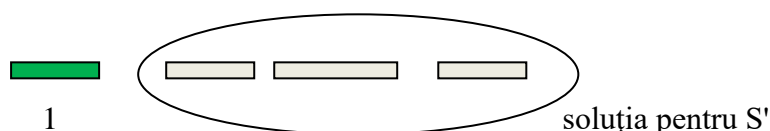
Fie $n \geq 2$. Presupunem că algoritmul `selectie_activitati` construiește o soluție optimă pentru orice mulțime de activități S' cu $|S'| < n$

Fie S o mulțime de n activități.

a) **Există o soluție optimă pentru S care conține activitatea 1 (prima activitate adăugată la soluția AS de algoritmul greedy).**

Într-adevăr, fie $O = \{o_1, \dots, o_p\}$ o soluție optimă pentru S (o submulțime de cardinal maxim de activități compatibile) cu activitățile notate astfel încât $o_1 < \dots < o_p$, deci $to_1 \leq \dots \leq to_p$. Dacă activitatea 1 aparține lui O, atunci afirmația a) este adevărată. Altfel considerăm $O' = O - \{o_1\} \cup \{1\}$. Deoarece $t_1 \leq to_1$, activitatea 1 este compatibilă cu toate celelalte activități din $O - \{o_1\}$, deci O' este o submulțime de activități compatibile cu $|O'| = |O|$, adică o soluție optimă care conține activitatea 1.

b) Fie $S' = \{k \in S, s_k \geq t_1\}$ mulțimea activităților care încep după ce se termină activitatea 1 (compatibile cu activitatea 1). Conform ipotezei de inducție soluția construită de algoritmul `selectie_activitati` pentru S', anume $AS' = AS - \{1\}$, este soluție optimă pentru S'. Rezultă că $AS = AS' \cup \{1\}$ este soluție optimă pentru S (altfel, conform punctului a) ar exista o soluție optimă O pentru S care conține activitatea 1 cu cardinal mai mare decât AS: $|O| > |AS|$; dar atunci $O - \{1\}$ este soluție posibilă pentru S' cu $|O - \{1\}| > |AS - \{1\}| = |AS'|$, ceea ce contrazice optimalitatea lui AS')



Variante. Generalizări

1. Problema partiționării intervalelor (- programarea tuturor activităților folosind un numărului minim de resurse = partiționarea unei mulțimi de intervale într-un număr minim de submulțimi de intervale disjuncte două câte două -v. laborator, seminar, [2])
2. Cazul în care fiecare interval are asociată o pondere (un profit asociat activității) și se cere determinarea unei submulțimi de intervale disjuncte de pondere maximă – la programare dinamică

• Memorarea textelor pe bandă

n texte cu lungimile $L(1), \dots, L(n)$ urmează a fi așezate pe o bandă. Pentru a citi textul de pe poziția k , trebuie citite textele de pe pozițiile $1, 2, \dots, k$ (conform specificului accesului secvențial pe bandă). Să se determine o modalitate de așezare a textelor pe bandă astfel încât timpul mediu de acces să fie minimizat.

Variantă de enunț: Care este ordinea de execuție a unor comenzi cu durate de execuție cunoscute astfel încât să se minimizeze timpul total de așteptare al clienților pentru finalizarea execuției tuturor comenzilor (= suma timpilor de așteptare pentru finalizarea fiecărui proiect/activitate)

Soluție. O soluție înseamnă o permutare $p \in S_n$.

Pentru o astfel de permutare (ordine de așezare a textelor pe bandă), timpul pentru a citi textul de pe poziție k este: $T_p(k) = L(p_1) + \dots + L(p_k)$. Presupunând textele egal probabile, trebuie minimizată valoarea $T(p) = \frac{1}{n} \sum_{k=1}^n T_p(k)$.

Să observăm că funcția T se mai poate scrie:

$$T(p) = \frac{1}{n} \sum_{k=1}^n (n - k + 1) L(p_k)$$

Conform strategiei Greedy, începem prin a ordona crescător vectorul L . Rezultă că în continuare $L(i) \leq L(j)$, $\forall i < j$.

Corectitudine [1]:

Fie $p \in S_n$ optimă, adică p minimizează funcția T . Presupunem că $\exists i < j$ cu $L(p_i) > L(p_j)$. Considerăm permutarea p' în care am interschimbat elementele de pe pozițiile i și j (este suficient să considerăm $j = i + 1$). Atunci, folosind formulele pentru $T(p)$ și $T(p')$ și reducând termenii în diferența $T(p) - T(p')$, obținem

$$\begin{aligned} n[T(p) - T(p')] &= (n-i+1)L(p_i) + (n-j+1)L(p_j) - \\ &\quad - (n-i+1)L(p_j) - (n-j+1)L(p_i) = \\ &= (j-i)L(p_i) + (i-j)L(p_j) = \\ &= (j-i)[L(p_i) - L(p_j)] > 0 \end{aligned}$$

Rezultă că $T(p') < T(p)$. Contradicție.

Variante. Generalizări (v. laborator+seminar)

1. Fiecare text are asociată o frecvență de citire
2. Avem la dispoziție p benzi ($p \geq 1$)

• Problema continuă a rucsacului [1]

Se consideră un rucsac de capacitate (greutate) maximă G și n obiecte caracterizate prin:

- greutatea lor g_1, \dots, g_n ;
- câștigurile c_1, \dots, c_n obținute la încărcarea lor în totalitate în rucsac.

Din fiecare obiect poate fi încărcată orice fracțiune a sa.

Să se determine o modalitate de încărcare de (fracțiuni de) obiecte în rucsac, astfel încât câștigul total să fie maxim.

Variantă de enunț: T =timp total de funcționare a unei resurse, n activități cu durata d_i și profitul p_i . O activitate începută poate fi întreruptă obținându-se un profit parțial.

Soluție [1]. Prin *soluție* înțelegem un vector $x = (x_1, \dots, x_n)$ cu

$$\begin{cases} x_i \in [0, 1], \forall i \\ \sum_{i=1}^n g_i x_i \leq G \end{cases}$$

O *soluție optimă* este soluție care maximizează funcția $f(x) = \sum_{i=1}^n c_i x_i$.

Dacă suma greutatea obiectelor este mai mică decât G , atunci încărcăm toate obiectele: $x = (1, \dots, 1)$. De aceea presupunem în continuare că $g_1 + \dots + g_n > G$.

Conform strategiei Greedy, ordonăm obiectele descrescător după câștigul la unitatea de greutate, deci lucrăm în situația:

$$\frac{c_1}{g_1} \geq \frac{c_2}{g_2} \geq \dots \geq \frac{c_n}{g_n} \quad (*)$$

Algoritmul constă în încărcarea în această ordine a obiectelor, atâta timp cât nu se depășește greutatea G (ultimul obiect poate fi eventual încărcat parțial):

```
G1 ← G { G1 reprezintă greutatea disponibilă }
for i=1,n
  if g_i ≤ G1
    x_i ← 1; G1 ← G1 - g_i
  else
    x_i ← G1 / g_i;
    for j=i+1,n
      x_j ← 0
    stop
write(x)
```

Am obținut deci $x = (1, \dots, 1, x_j, 0, \dots, 0)$ cu $x_j \in [0, 1)$.

Corectitudine [1]. – vezi curs. Idee: Arătăm că soluția astfel obținută este optimă.

Fie y soluția optimă care are o subsecvență inițială maximă în comun cu x :

$$y = (\dots, y_k, \dots) \text{ cu } \begin{cases} \sum_{i=1}^n g_i y_i = G \\ \sum_{i=1}^n c_i y_i \text{ maxim} \end{cases}$$

Dacă $y \neq x$, fie k **prima poziție** pe care $y_k \neq x_k$. Atunci avem

- $k \leq j$: pentru $k > j$ se depășește G .
- $y_k < x_k$

Considerăm soluția: $y' = (y_1, \dots, y_{k-1}, x_k, \alpha y_{k+1}, \dots, \alpha y_n)$ cu $\alpha < 1$

(primele $k-1$ componente coincid cu cele din x). Păstrăm greutatea totală G , deci:

$$g_k (x_k - y_k) = (1 - \alpha) (g_{k+1} y_{k+1} + \dots + g_n y_n) \quad (**)$$

Comparăm performanța lui y' cu cea a lui y :

$$f(y') - f(y) = c_k / g_k [g_k (x_k - y_k) + (\alpha - 1) (g_k / c_k c_{k+1} y_{k+1} + \dots + g_k / c_k c_n y_n)]$$

Din alegerea Greedy $\frac{g_k}{c_k} \leq \frac{g_j}{c_j}$ pentru $j > k$. Înlocuind $\frac{g_k}{c_k}$ cu $\frac{g_j}{c_j}$ pentru $j > k$ și folosind (**)

rezultă $f(y') - f(y) \geq 0$, deci $f(y') \geq f(y)$. Deoarece y este optimă, rezultă că $f(y') = f(y)$. Contradicție, y' are mai multe elemente inițiale în comun cu x decât y .

Variante. Generalizări

Problema discretă a rucsacului diferă de cea continuă prin faptul că fiecare obiect poate fi încărcat numai în întregime în rucsac.

Să observăm că aplicarea metodei Greedy eșuează în acest caz. Într-adevăr, aplicarea ei pentru: $G=5$, $n=3$ și $g=(4, 3, 2)$, $c=(6, 4, 2.5)$ are ca rezultat încărcarea primului obiect; câștigul obținut este 6. Dar încărcarea ultimelor două obiecte conduce la câștigul superior 6.5.

Bibliografie

1. Horia Georgescu. **Tehnici de programare**. Editura Universității din București 2005
2. **Jon Kleinberg, Éva Tardos** - Algorithm Design, 2005 Addison-Wesley Professional
Slideu-rile oficiale pentru această carte se pot găsi la
<http://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/>
o versiune îmbogățită a acestora este accesibilă la adresa
<http://www.cs.princeton.edu/~wayne/kleinberg-tardos>