Metoda Greedy

Prezentare [1]

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

Considerăm mulţimea finită $A=\{a_1,...,a_n\}$ şi o proprietate p definită pe mulţimea submulţimilor lui A (notată $P(A)$):

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

O submulţime $S \subset A$ se numeşte ***soluţie*** dacă $p(S)=1$.

Dintre soluţii dorim să alegem una care optimizează o funcţie $f: P(A)\to\mathbf{R}$ dată.

Metoda urmăreşte evitarea parcurgerii tuturor submulţimilor (ceea ce ar necesita un timp de calcul exponenţial), **mergându-se "direct" spre soluţia optimă**. Nu este însă garantată obţinerea unei soluţii optime; de aceea aplicarea metodei Greedy trebuie însoţită neapărat de o demonstraţie.

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

```
S←φ
for i=1,n
   x←alege(A);
   A←A\{x}
   if p(S∪{x})=1
      S←S∪{x}
```

```
prel(A)
S←φ
for i=1,n
   if p(S∪{ai})=1
      S←S∪{ai}
```

Observaţii:

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

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

Soluție. Vom parcurge mulţimea şi 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 mulţimea $A=\{a_1,...,a_n\}$ cu elemente pozitive. Să se determine o submulţime 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 $A=\{6,3,4,2\}$ şi $M=7$ obţinem $\{6\}$. Dar soluţia 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 mulţime $S=\{1, 2,\ldots, n\}$ de n activităţi (spectacole) care doresc să folosească aceeaşi 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 şi un timp de terminare t_i. Problema selectării activităţilor (spectacolelor) constă în selectarea unei mulţimi de activităţi compatibile între ele de cardinal maxim (compatibile = având intervalele de desfăşurare disjuncte).

Variantă de enunț: Știind intervalele de desfăşurare a unor conferinţe la care dorim să asistăm în timpul unei zile (sau emisiuni pe care dorim sa le vizionăm), să se determine numărul maxim de conferinţe la care putem participa (= cu intervale de desfăşurare 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 activităţile crescător după timpul de terminare.

Să presupunem că activităţile sunt ordonate crescător după timpul de terminare: $t_1 \le t_2 \le \ldots \le t_n$

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

```
function selectie_activitati(s, t)
   S ← {1}
   us ← 1
   for ac = 2,n
      if s_ac ≥ t_us
          S ← S ∪ {ac}
          us ← ac
   return S
```

În mulţimea S se introduc activităţile care au fost selectate. Variabila uas indică ultima activitate introdusă în S.

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

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

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

Corectitudine. Presupunem (ca şi în algoritm) că activităţile sunt ordonate crescător după timpul de terminare: $t_1 \le t_2 \le \ldots \le t_n$. Fie $AS = \{g_1=1, \ldots, g_t\}$ (cu $g_1 < \ldots < g_t$) soluţia Greedy.

Varianta 1 [2]:

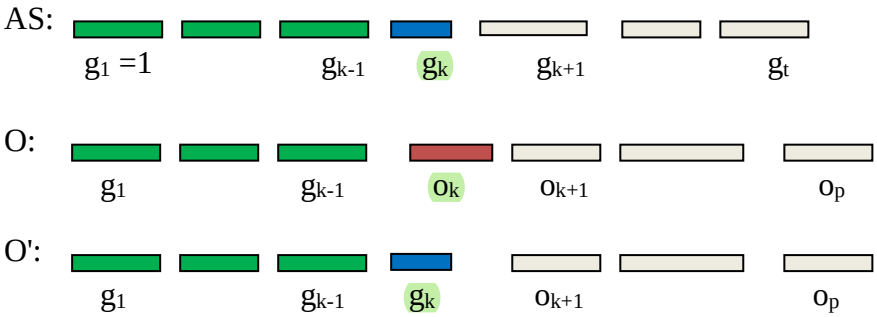
Considerăm o soluție optimă $O = \{o_1, \ldots, o_p\}$ (cu activităţile notate astfel încât $o_1 < \ldots < o_p$, deci $to_1 \le \ldots \le to_p$) care are **un număr maxim de elemente (inițiale) în comun cu soluția Greedy** AS. Avem $t \le p$. Presupunem $O \neq AS$.

Atunci există un indice $k \le t$ astfel încât $g_k \neq o_k$, altfel am avea $g_1 = o_1, \ldots, g_t = o_t$, adică $AS \subseteq O$ şi $t < p$. Dar, deoarece activitatea o_{t+1} este compatibilă cu toate activităţile din $\{o_1, \ldots, o_t\} = \{g_1, \ldots, g_t\}$, algoritmul $selectie\_activitati$ ar mai fi avut activităţi compatibile cu $\{g_1, \ldots, g_t\}$ din care să selecteze, deci ar fi furnizat o soluţie cu mai multe elemente.

Fie $k \leq t$ cel mai mic indice astfel încât $g_k \neq o_k$ (prima poziţie pe care soluţia greedy AS şi soluţia optimă O diferă). La pasul la care `selectie_activitati` a ales activitatea g_k, activitatea o_k era de asemenea neselectată şi compatibilă cu activităţile deja selectate $g_1 = o_1$, … ,$g_{k-1} = o_{k-1}$. 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}, … , o_t (deoarece acestea încep după ce se termină o_k) – v. desenul de mai jos:

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

Atunci putem înlocui în soluţia optimă O activitatea o_k cu g_k şi obţinem tot o soluţie posibilă $O' = O - \{o_k\} \cup \{g_k\}$ (formată din activităţi compatibile din S). Avem $|O'| = |O|$, deci O' este tot o soluţie optimă, dar care are mai multe elemente în comun cu soluţia greedy AS, contradicţie.



Varianta 2 (similar) – Demonstrăm prin inducţie după n că algoritmul greedy `selectie_activitati` construieşte o soluţie optimă.

Pentru n = 0, 1 afirmaţia este evidentă.

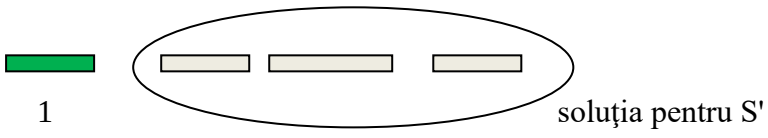
Fie $n \geq 2$. Presupunem că algoritmul `selectie_activitati` construieşte o soluţie optimă pentru orice mulţime de activităţi S' cu $|S'|<n$

Fie S o mulţime de n activităţi.

a) **Există o soluţie optimă pentru S care conţine activitatea 1 (prima activitate adăugată la soluţia AS de algoritmul greedy).**

Într-adevăr, fie $O = \{o_1,\ldots,o_p\}$ o soluţie optimă pentru S (o submulţime de cardinal maxim de activităţi compatibile) cu activităţile notate astfel încât $o_1 < \ldots < o_p$, deci $to_1 \leq \ldots \leq to_p$. Dacă activitatea 1 aparţine lui O, atunci afirmaţia a) este adevărată. Altfel considerăm $O' = O - \{o_1\} \cup \{1\}$. Deoarece $t_1 \leq to_1$, activitatea 1 este compatibila cu toate celelalte activităţi din $O - \{o_1\}$, deci O' este o submulţime de activităţi compatibile cu $|O'|=|O|$, adică o soluţie optimă care conţine activitatea 1.

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



Variante. Generalizări

1. Problema partiționării intervalelor (- programarea tuturor activităţilor folosind un numărului minim de resurse = partiţionarea unei mulţimi de intervale într-un număr minim de submulţimi 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 $L(1),\ldots,L(n)$ urmează a fi aşezate pe o bandă. Pentru a citi textul de pe poziţia k, trebuie citite textele de pe poziţiile $1,2,\ldots,k$ (conform specificului accesului secvenţial pe bandă). Să se determine o modalitate de aşezare a textelor pe bandă astfel încât timpul mediu de acces să fie minimizat.

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

Soluție. O *soluţie* înseamnă o permutare $p\in S_n$.

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

Să observăm că funcţia 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 $L(i)\le L(j),\ \forall i<j$.

Corectitudine [1]:
Fie $p\in S_n$ optimă, adică p minimizează funcţia T. Presupunem că $\exists i<j$ cu $L(p_i)>L(p_j)$. Considerăm permutarea p' în care am interschimbat elementele de pe poziţiile i şi j (este suficient să considerăm $j=i+1$). Atunci, folosind formulele pentru $T(p)$ şi $T(p')$ şi reducând termenii în diferenţa $T(p)-T(p')$, obţinem

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

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

Variante. Generalizări (v. laborator+seminar)

1. Fiecare text are asociată o frecvență de citire
2. Avem la dispoziție p benzi (p>=1)

- **Problema continuă a rucsacului [1]**

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

- greutăţile lor $g_1, \ldots, g_n$;
- câştigurile $c_1, \ldots, c_n$ obţinute la încărcarea lor în totalitate în rucsac.

Din fiecare obiect poate fi încărcată orice fracţiune a sa.

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

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

Soluție [1]. Prin *soluţie* înţelegem un vector $x=(x_1,\ldots,x_n)$ cu $\begin{cases} x_i \in [0,1], \forall i \\ \sum_{i=1}^{n} g_i x_i \le G \end{cases}$

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

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

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

$$\frac{c_1}{g_1} \ge \frac{c_2}{g_2} \ge \ldots \ge \frac{c_n}{g_n} \qquad (*)$$

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

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

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

Corectitudine [1]. – vezi curs. Idee: Arătăm că soluţia astfel obţinută este optimă.

Fie y soluţia optimă **care are o subsecvenţă iniţială maximă în comun cu x**:

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

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

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

Considerăm soluţia: $y'=(y_1,\ldots,y_{k-1},x_k,\alpha y_{k+1},\ldots,\ \alpha y_n)$ cu $\alpha<1$ (primele $k-1$ componente coincid cu cele din x). Păstrăm greutatea totală G, deci:

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

Comparăm performanţa lui y' cu cea a lui y:

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

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 şi folosind (**) rezultă $f(y')-f(y) \geq 0$, deci $f(y') \geq f(y)$. Deoarece y este optimă, rezultă că $f(y') = f(y)$. Contradicţie, y′ are mai multe elemente iniţiale 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 eşuează în acest caz. Într-adevăr, aplicarea ei pentru: $G=5,\ n=3$ şi $g=(4,3,2),\ c=(6,4,2.5)$
are ca rezultat încărcarea primul obiect; câştigul obţinut este 6. Dar încărcarea ultimelor două obiecte conduce la câştigul superior 6.5.

Bibliografie

1. Horia Georgescu. **Tehnici de programare**. Editura Universităţii din Bucureşti 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 îmbogăţită a acestora este accesibilă la adresa
   http://www.cs.princeton.edu/~wayne/kleinberg-tardos