Metoda Greedy

Prezentare [1]

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

Un tipar posibil de probleme la care se aplică metoda Greedy este următorul:

> *Se dă o mulțime finită A. Să se determine o submulțime finită B⊆A care satisface o anumită proprietăți propr(**este soluție posibilă**) și îndeplinește un criteriu de optim (**este soluție optimă**), adică minimizează/maximizează o **funcție obiectiv f***

Spre exemplu, să considerăm următoarea problemă: Dată o mulțime de intervale, să se determine o submulțime de cardinal maxim de intervale din mulțimea dată care nu se intersectează. Pentru această problemă A = mulțimea de intervale dată, o soluție S ⊆ A este o submulțime de intervale care trebuie să verifice proprietatea că sunt disjuncte două câte două, iar funcția obiectiv care trebuie maximizată este f(S) = |S|.

Formalizând, un cadru posibil pentru aplicarea metodei Greedy este următorul: Considerăm mulțimea finită $A=\{a_1,\ldots,a_n\}$ şi o proprietate `propr` definită pe mulţimea submulţimilor lui `A` (notată $P(A)$):

$$propr: P(A) \to \{True, False\} \text{ cu } \begin{cases} propr(\emptyset) = True \\ propr(X) \Rightarrow propr(Y), \forall Y \subset X \end{cases}$$

O submulţime $S \subset A$ se numeşte ***soluţie*** (*soluţie posibilă*) dacă $propr(S)=1$.
Dintre soluțiile posibile dorim să alegem una care optimizează (minimizează/maximizează) o funcție $f: P(A) \to \mathbf{R}$ dată.

Metoda Greedy urmăreşte evitarea parcurgerii tuturor submulţimilor (ceea ce ar necesita un timp de calcul exponenţial), **mergându-se "direct" spre soluţia optimă**. Astfel, soluția se construiește element cu element, elementul ales la un pas pentru a se adăuga în soluție fiind cel care pare cel mai “bun” la acel pas, conform criteriului de optim f. 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: cu prelucrare inițială a elementelor din A (de obicei ordonare) sau fără:

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

```
prelucreaza(A)
S←φ
for x in A
   if propr(S ∪ x)=True
      S ← S ∪ {x}
return S
```

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. La fiecare pas trebuie să adăugăm la submulțime un element care să crească suma elementelor din submulțime cu o valoare cât mai mare. Astfel, vom adăuga inițial la submulțime elementul cel mai mare din A. Apoi vom parcurge mulțimea și vom selecta numai elementele pozitive, dacă astfel de elemente există (dacă nu există elemente pozitive, soluția este dată de cel mai mare element din mulțime). Corectitudinea soluției este evidentă.

Exemplul 2 Se consideră mulțimea de valori reale $A=\{a_1,...,a_n\}$ și k<n. Să se determine o submulțime a lui A de cardinal k a cărei sumă a elementelor este maximă.

Soluție. Vom alege în S cele mai mari k elemente.
Fără o prelucrare inițială a elementelor, pseudocodul este următorul:

```
S←ϕ
for i=1,n
   x ← maxim(A)
   A ← A\{x} -> eliminam sau marcam x
   S ← S∪{x}
   if |S| = k
        stop (break)
```

Complexitatea acestui algoritm este O(nk) (alegem de k ori maximul), care oscilează între O(n) și $O(n^2)$ după cum valoarea lui k este mai apropiată de 1 sau de n.

Am putea însă ordona de la început elementele din A descrescător, pentru a alege mai simplu pe cele mai mari k (fiind primele k din ordonare), și în acest caz complexitatea va fi $O(n\log_2(n)+k) = O(n\log_2(n))$ (deoarece $k \le n \le n\log2(n)$):

```
sorteaza_descrescator(A)
S←ϕ
for x in A
   S ← S ∪ {x}
   if |S| = k
        stop (break)
return S
```

Exemplul 3 (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.

Probleme de planificare

Vom prezenta câteva tipuri de probleme de planificare care se pot rezolva cu ajutorul metodei Greedy. Cadrul comun este următorul.
Avem n activități care se desfășoară utilizând o resursă comună (procesor, sală de spectacole, bucătar/evaluator de proiecte care execută/evaluează comenzi/proiecte pe rând etc). La un moment dat **doar o activitate poate folosi resursa**, iar o activitate începută trebuie finalizată (nu poate fi întreruptă)

A. **Planificarea activităților astfel încât să fie minimizat timpul mediu de așteptare** (timpul de așteptare necesar finalizării unei activități este dat de suma dintre timpii de execuție al activităților programate înaintea ei și timpul de execuție al ei, deoarece o activitate trebuie să aștepte se finalizeze activitățile programate înainte pentru a putea începe), dacă se cunosc $t_1,...,t_n$ duratele de desfășurare a activităților

B. **Planificarea unui număr maxim posibil de activități**, dacă se cunosc intervalele în care trebuie să se desfășare aceste activități **– Problema spectacolelor**

C. Determinarea numărului minim de resurse necesare pentru a putea desfășura toate activitățile, dacă se cunosc intervalele în care trebuie să se desfășare aceste activități – **Problema partiționării intervalelor**

A. Planificarea activităților cu minimizarea timpului mediu de așteptare

Avem n activități (comenzi date simultan de n clienți la un restaurant) care se desfășoară utilizând o resursă comună (un bucătar care trebuie să execute comenzile, pe rând, câte una sau un evaluator de proiecte care evaluează pe rând proiecte depuse, un procesor etc)

Știind durata fiecărei activități t_1, t_2,..., t_n (cât îi ia bucătarului să facă fiecare comandă sau evaluatorului să evalueze fiecare proiect), să se determine ordinea în care trebuie executate activitățile astfel încât timpul mediu de așteptare să fie minim: o activitate trebuie să aștepte se finalizeze activitățile programate înainte pentru a putea începe (un client trebuie să aștepte ca bucătarul să termine toate comenzile pe care le face înainte plus comanda lui pentru a fi servit; o persoană care a depus un proiect primește răspunsul evaluării proiectului său după ce au fost evaluate toate proiectele programate înainte plus proiectul său).
Pentru activitatea i notăm cu a_i timpul de așteptare până la finalizarea activității i (în raport cu ordinea aleasă a activităților).

De exemplu, să presupunem că avem n = 4 activități cu duratele $t_1 = 5$, $t_2 = 1$, $t_3 = 7$, $t_4 = 2$. Timpul mediu de așteptare depinde de ordinea în care se execută activitățile. Pentru exemplificarea modului în care se calculează acest timp vom considera câteva ordonări și vom calcula timpul mediu pentru fiecare dintre aceste ordonări.

- dacă activitățile se execută în ordinea 1, 2, 3, 4, timpul de așteptare pentru fiecare activitate este calculat în următorul tabel:

activitatea	durata	Timpul de așteptare
1	$t_1 = 5$	$a_1 = \mathbf{t_1} = 5$
2	$t_2 = 1$	$a_2 = a_1 + t_2 = 5 + 1 = 6$ $= \mathbf{t_1 + t_2}$
3	$t_3 = 7$	$a_3 = a_2 + t_3 = 6 + 7 = 13$ $= \mathbf{(t_1 + t_2) + t_3}$
4	$t_4 = 2$	$a_4 = a_3 + t_4 = 13 + 2 = 15$ $= \mathbf{(t_1 + t_2 + t_3) + t_4}$

Timpul mediu de așteptare pentru ordonarea 1, 2, 3, 4 va fi

$$\frac{a_1 + a_2 + a_3 + a_4}{4} = \frac{5+6+13+15}{4} = \frac{39}{4}$$

- dacă activitățile se execută în ordinea 2, 4, 1, 3, timpul de așteptare pentru fiecare activitate este calculat în următorul tabel:

activitatea	durata	Timpul de așteptare
2	$t_2 = 1$	$a_2 = \mathbf{t_2} = 1$
4	$t_4 = 2$	$a_4 = a_2 + t_4 =$ $= \mathbf{t_2 + t_4} = 1 + 2 = 3$
1	$t_1 = 5$	$a_1 = a_4 + t_1 = 3 + 5 = 8$ $= \mathbf{(t_2 + t_4) + t_1}$
3	$t_3 = 7$	$a_3 = a_1 + t_3 =$ $= \mathbf{(t_2 + t_4 + t_1) + t_3} = 8 + 7 = 15$

Timpul mediu de așteptare pentru ordonarea 1, 2, 3, 4 va fi

$$\frac{a_2 + a_4 + a_1 + a_3}{4} = \frac{1+3+8+15}{4} = \frac{27}{4}$$

Astfel, a determina o soluție este echivalent cu a determina o permutare $\sigma \in S_n$ pentru care se minimizează timpul mediu de așteptare:

$$T(\sigma) = \frac{1}{n}(a_{\sigma(1)} + \ldots + a_{\sigma(n)}) =$$
$$= \frac{1}{n}\left[t_{\sigma(1)} + (t_{\sigma(1)} + t_{\sigma(2)}) + \ldots + (t_{\sigma(1)} + \ldots + t_{\sigma(n)})\right] =$$
$$= \frac{1}{n}\sum_{k=1}^{n}(n-k+1)t_{\sigma(k)}$$

Soluție Greedy: După ce criteriu alegem prima activitate, sau, mai general, activitatea planificată la pasul i? Deoarece după prima activitate așteaptă toate celelalte, vom alege ca prima activitate efectuată să fie cea cu durata cea mai mică. Raționăm similar la fiecare pas și alegem mereu activitatea cu durata cea mai mică dintre cele rămase. Astfel, strategia Greedy pentru această problemă este de a planifica activitățile în ordine crescătoare după durată.

Corectitudine [1]: Vom demonstra că soluția dată de strategia Greedy este optimă. Pentru aceasta să renotăm activitățile astfel încât

$t_1 \le t_2 \le ... \le t_n$.

Atunci soluția Greedy este permutarea identică id.

Fie $\sigma \in S_n$ o așezare optimă (σ minimizează funcţia T).

Presupunem prin absurd că $T(\sigma) < T(id)$, adică soluția Greedy nu este optimă.

Să observăm întâi că dacă σ ar diferi de id doar prin ordinea în care alege activitățile în caz de egalitate, atunci am avea $T(\sigma) = T(id)$.

Evidențiem o diferență între σ și id, adică o inversiune în σ și arătăm că eliminând această diferență (permutând elementele inversiunii) obținem o soluție mai bună decât σ.

Atunci $\exists i<j$ cu $t_{\sigma(i)} > t_{\sigma(j)}$ Considerăm permutarea σ' obținută din σ prin interschimbarea elementelor de pe pozițiile i şi j (este suficient să considerăm $j=i+1$; permutarea σ' are mai puține inversiuni, deci este "mai apropiată" de soluția Greedy *id* decât σ).

Arătăm că $T(\sigma') < T(\sigma)$, ceea ce contrazice optimalitatea lui σ (soluția σ' "mai apropiată" de soluția Greedy are timpul asociat mai mic decât σ).

Folosind formulele pentru $T(\sigma)$ şi $T(\sigma')$ şi reducând termenii în diferența $T(\sigma) - T(\sigma')$, obținem:

$$T(\sigma) - T(\sigma') = \frac{1}{n}\sum_{k=1}^{n}(n-k+1)t_{\sigma(k)} - \frac{1}{n}\sum_{k=1}^{n}(n-k+1)t_{\sigma'(k)} =$$

$$= \frac{1}{n}\left[(n-i+1)t_{\sigma(i)} + (n-j+1)t_{\sigma(j)} - (n-i+1)t_{\sigma(j)} - (n-j+1)t_{\sigma(i)}\right] =$$

$$= \frac{1}{n}\left[(j-i)t_{\sigma(i)} + (i-j)t_{\sigma(j)}\right] = \frac{(j-i)(t_{\sigma(i)} - t_{\sigma(j)})}{n} > 0$$

(deoarece $j > i$ și $t_{\sigma(i)} > t_{\sigma(j)}$)

Rezultă că $T(\sigma') < T(\sigma)$, contradicție.

Variante. Generalizări (v. laborator+seminar)

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

B. Planificarea unui număr maxim posibil de activități – Problema spectacolelor [2]

- **Problema selectării activităților (problema spectacolelor) [2]**

Să presupunem că avem o mulţime $A=\{1, 2,\ldots, n\}$ 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, deci se poate desfășura doar în intervalul $[s_i, t_i)$. Problema selectării activităților (spectacolelor) constă în selectarea unei mulțimi de activități compatibile între ele de cardinal maxim (compatibile = având intervalele de desfășurare disjuncte).

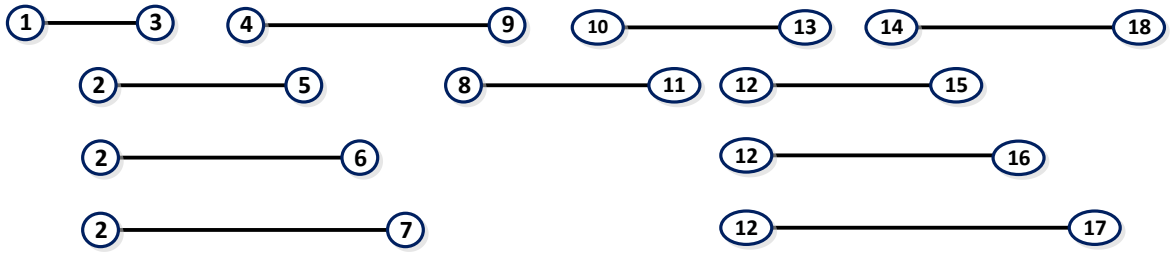
Variantă de enunț: Știind intervalele de desfăşurare a unor conferinţe la care dorim să asistăm în timpul unei zile (sau emisiuni pe care dorim sa le vizionăm), să se determine numărul maxim de conferințe la care putem participa (= cu intervale de desfășurare disjuncte).

Variantă de enunț cu intervale: Se consideră o mulțime A de intervale închise. Să se determine o submulțime de cardinal maxim de intervale disjuncte din A

Soluție. După ce criteriu alegem prima activitate, sau, mai general, activitatea planificată la pasul curent?

Posibile criterii „greedy" de alegere:

- activitatea cu durata cea mai mică, pentru a ocupa cât mai puțin resursa – nu este corect, deoarece acest criteriu nu ține cont de intervalul în care se desfășoară activitatea, care se poate intersecta cu intervalele altor activități. Spre exemplu, pentru activitățile cu intervalele [1,10), [9,12), [13,18) soluția optimă este formată cu activitățile 1 și 3 (nu conține activitatea 2 cu lungimea cea mai mică)
- activitatea care începe cel mai devreme, pentru a ocupa cât mai repede resursa - – nu este corect, această activitate poate avea durată mare: contraexemplu: [1,10), [2,4), [5,8)
- activitatea care este compatibilă cu cât mai multe activități (sau, echivalent, se intersectează cu cât mai puține intervalele de desfășurare ale celorlaltor activități), pentru ca prin programarea ei să fie excluse din lista activităților care se mai pot programa cât mai puține activități – nici acest criteriu nu este corect, pentru că o mare parte dintre activitățile cu care este compatibilă pot sa nu fie compatibile două câte două, ca în exemplul din desenul de mai jos (unde sunt indicate intervalele de desfășurare ale activităților). Intervalul [8,11) se intersectează doar cu două intervale, dar soluția optimă este formată cu activitățile cu intervalele [1,3), [4,9), [10,13), [14,18)



Strategia Greedy (corectă) 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 a lăsa după terminarea ei resursa liberă cât mai mult timp.** Pentru aceasta este util să procesăm inițial datele, mai exact să ordonăm activitățile crescător după timpul de terminare.

Pentru a simplifica scrierea pseudocodului și justificarea corectitudinii, să presupunem că activitățile sunt ordonate crescător după timpul de terminare: $t_1 \leq t_2 \leq ... \leq t_n$

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

```
function selectie_activitati(s, t)
   S ← {1}
   us ← 1  //indicele ultimei activitati selectate
   pentru ac = 2,n executa  //activitatea curenta
        daca s_ac ≥ t_us  atunci  // t_us = max{ t_k , k∈S}
           S ← S ∪ {ac}
           us ← ac
   return S
```

În mulţimea `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 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 $s_{ac} \geq t_{us}$.

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

Varianta 1 [2]:

Vom folosi o strategie întâlnită în multe demonstrații de corectitudine pentru algoritmi de tip Greedy, similară celei din problema anterioară. Pentru a demonstra că soluția Greedy S este optimă considerăm o soluție optimă O care are un număr maxim de elemente inițiale în comun cu Greedy (o soluție optimă care "seamănă" cel mai mult cu soluția Greedy). Evidențiem prima diferență dintre O și S și arătăm că eliminând această diferență (înlocuind elementul din O care nu este în soluția Greedy cu unul din S – exchange argument) obținem o soluție cel puțin la fel de bună ca O, care are însă mai multe elemente în comun cu O (deci **O se poate transforma element cu element printr-o strategie de tip "exchange argument" în S**)

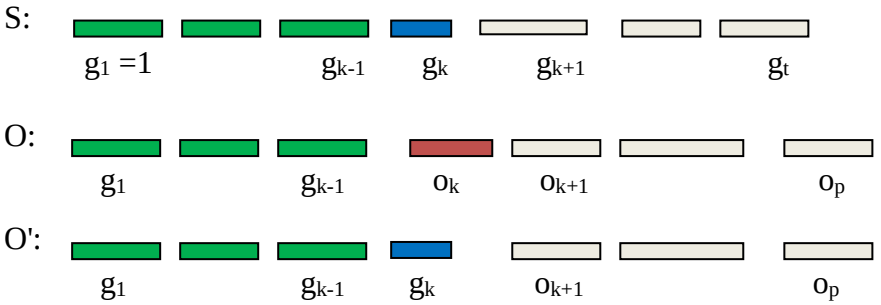
Considerăm o soluție optimă O = $\{o_1, \ldots, o_p\}$ (cu activităţile notate astfel încât $o_1 < \ldots < o_p$, deci $to_1 \leq \ldots \leq to_p$) care are **un număr maxim de elemente (inițiale) în comun cu soluția Greedy** `S`. Avem $m \leq p$. Presupunem prin absurd $|O| > |S|$.

Atunci există un indice $k \leq m$ astfel încât $g_k \neq o_k$ (nu putem avea $S \subset O$ şi $m < p$ deoarece activitatea o_{m+1} este compatibilă cu toate activitățile din $\{o_1, \ldots, o_m\} = \{g_1, \ldots, g_m\}$ și algoritmul Greedy `selectie_activitati` ar mai fi avut activități compatibile cu $\{g_1, \ldots, g_m\}$ din care să selecteze, deci ar fi furnizat o soluție cu mai multe elemente).

Fie $k \leq m$ cel mai mic indice astfel încât $g_k \neq o_k$ (prima poziţie pe care soluţia greedy S 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 $g_1 = o_1, \ldots, g_{k-1} = o_{k-1}$. 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 $o_{k+1}, \ldots, o_m$ (deoarece acestea încep după ce se termină o_k) – v. desenul de mai jos:

$$t_{g_k} \leq t_{o_k} \leq s_{o_{k+1}} \leq t_{o_{k+1}} \leq ... \leq s_{o_p} \leq t_{o_p}$$

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 activități compatibile din A). Avem $|O'| = |O|$, deci O' este tot o soluție optimă, dar care are mai multe elemente în comun cu soluţia greedy S, contradicţie.



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

Pentru n = 0, 1 afirmația este evidentă.

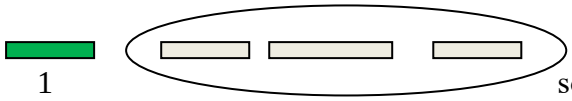
Fie n ≥ 2. Presupunem că algoritmul `selectie_activitati` construiește o soluție optimă pentru orice mulțime de activități A' cu |A'|<n

Fie A o mulțime de n activități.

a) **Există o soluție optimă care conţine prima activitate selectată de algoritmul greedy, anume g_1= 1**

Într-adevăr, fie O = $\{o_1,\ldots,o_p\}$ o soluţie optimă (o submulţime de cardinal maxim de activităţi compatibile) cu activităţile notate astfel încât $o_1<\ldots<o_p$, deci $t_{o1}\leq\ldots\leq t_{op}$. Dacă activitatea 1 aparţine lui O, atunci afirmaţia a) este adevărată. Altfel considerăm O' = O – $\{o_1\}$ ∪ {1}. Deoarece $t_1 \leq t_{o1,}$ activitatea 1 este compatibila cu toate celelalte activităţi din O – $\{o_1\}$, deci O' este o submulțime de activități compatibile cu |O'|=| O|, adică o soluţie optimă care conţine activitatea 1.

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



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

C. Determinarea numărului minim de resurse necesare pentru a putea desfășura toate activitățile – Problema partiționării intervalelor [2]

Să presupunem că avem `n` activităţi (spectacole) care pentru a se desfășura au nevoie de o resursă (sală 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 $t_{i,}$ deci se poate desfășura doar în intervalul $[s_i, t_i)$. Astfel, pe o resursă se pot planifica doar activități cu intervalele de desfășurare disjuncte. Să se determine numărul minim k de resurse (săli de spectacole) de care este nevoie pentru a efectua toate activitățile (spectacolele) și o planificare a acestor activități pe cele k resurse.

Exemplu: pentru n=3 spectacole, care trebuie să se desfășoare în intervalele: [10, 14), [12, 16), respectiv [17, 18), sunt necesare 2 săli, o programare optimă fiind:

- Sala 1: [10, 14) – spectacolul 1, [17, 18) – spectacolul 3
- Sala 2: [12, 16) – spectacolul 2

Variantă de enunț cu intervale – Problema partiționării intervalelor: Se consideră o mulțime A de intervale închise. Să se împartă (partiţioneze) această mulţime de intervale într-un **număr minim de submulţimi** cu proprietatea că oricare două intervale dintr-o submulţime nu se intersectează şi să se afişeze aceste submulţimi

Soluție – Temă. v. seminar+laborator

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 (și valoarea asociată va fi acea fracțiune din câștigul total). Să se determine o modalitate de încărcare de (fracţiuni de) obiecte în rucsac, astfel încât câştigul total să fie maxim.

De exemplu, pentru G = 8, n= 4, obiectul 1 cu greutatea 4 și câștigul 8, obiectul 2 cu greutatea 8 și câștigul 12, obiectul 3 cu greutatea 3 și câștigul 6 și obiectul 4 cu greutatea 10 și câștigul 10 o soluție optimă este dată de vectorul de fracțiuni (1, 1/8, 1, 0) , adică se vor încărca obiectele 1 și 3 întregi si 1/8 din obiectul 2, iar câștigul total va fi 8 + 1/8*12 + 6 = 15,5

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 $g_1+\ldots+g_n>G$.**

Exemplu

Cum alegem obiectul care trebuie încărcat la pasul curent? Alegem dintre obiectele rămase pe cel mai valoros **pe unitatea de greutate** (cu valoarea cea mai mare a raportului c/g) și încărcăm cât putem de mult din el.

Astfel, pentru a implementa strategia Greedy, ar fi util sa ordonăm inițial obiectele descrescător după câștigul la unitatea de greutate. Algoritmul constă în încărcarea în această ordine a obiectelor, atâta timp cât nu se depăşeşte greutatea G, ultimul obiect adăugat putând fi eventual încărcat parţial. Mai exact, dacă presupunem obiectele ordonate astfel încât

$$\frac{c_1}{g_1} \ge \frac{c_2}{g_2} \ge \ldots \ge \frac{c_n}{g_n}$$

algoritmul este următorul:

```
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
             x_j ← 0
        stop
write(x)
```

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

Corectitudine [1]. –Arătăm că soluţia astfel obţinută este optimă.

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

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

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 o soluție y' cu **mai multe elemente inițiale în comun** cu soluția Greedy x, obținută din y înlocuind y_k cu x_k și păstrând o fracțiune mai mică din obiectele k+1,..., n, astfel încât să obținem tot greutatea G: $y'=(y_1,...,y_{k-1},x_k,\alpha y_{k+1},..., \alpha y_n)$ cu $\alpha<1$, astfel încât:

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

Comparăm performanța lui y' cu cea a lui y pentru a arăta că soluția y' este cel puțin la fel de bună ca y, dar cu mai multe elemente în comun cu soluția greedy x, ceea ce contrazice alegerea lui y. Avem

$$\begin{aligned} f(y')-f(y) &= c_k x_k + \alpha c_{k+1}y_{k+1} + ... + \alpha c_n y_n - (c_k y_k + c_{k+1}y_{k+1} + ... + c_n y_n) = \\ &= c_k(x_k-y_k) - (1-\alpha)(c_{k+1}y_{k+1}+...+c_n y_n) = \\ &= c_k/g_k \cdot g_k(x_k-y_k) + (\alpha-1)(c_{k+1}/g_{k+1} \cdot g_{k+1}y_{k+1}+...+ c_n/g_n \cdot g_n y_n) = \end{aligned}$$

Dar $1-\alpha > 0$ şi $c_k/g_k \geq c_s/g_s$, $\forall s>k$. Atunci

$f(y')-f(y) \geq c_k/g_k[g_k(x_k-y_k)+(\alpha-1)(c_{k+1}y_{k+1}+...+ c_n y_n)] = 0$ (din (*)),

deci $f(y') \geq f(y)$. Rezultă că și y' este soluție optimă (f(y) este maxim), contradicţie cu alegerea lui y, deoarece 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: $G=5$, $n=3$ 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 un câștig mai mare 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