

---

Curs 8-9: Tehnica alegerii local optimale ("greedy")

- Introducere
  - Principiul tehnicii
  - Verificarea corectitudinii si analiza complexității
  - Aplicații
  - Exerciții
- 

## 1 Introducere

Problemele pot fi încadrate în clase abstracte de probleme ce au anumite elemente comune. O clasă frecvent întâlnită în practică este cea a problemelor de *optimizare* de tipul:

*Să se determine  $x \in X$  astfel încât: (i)  $x$  satisface anumite condiții (restricții); (ii)  $x$  optimizează (minimizează sau maximizează) un criteriu.*

O subclasă importantă în informatică o reprezintă cea în care  $X$  este o mulțime finită, problema de optimizare fiind numită în acest caz *discretă* sau *combinatorială*. La o primă vedere problema pare foarte simplă întrucât este suficient să se parcurgă  $X$  și să se aleagă elementul care satisface restricțiile și optimizează criteriul. O astfel de abordare (numită și metoda forței brute) devine total ineficientă (sau chiar impracticabilă) atunci când numărul de elemente din  $X$  este foarte mare.

*Exemplu.* (problema submulțimii de sumă dată și cardinal minim) Se consideră mulțimea  $A = \{a_1, a_2, \dots, a_n\} \subset \mathbb{R}$  și valoarea  $C \leq \sum_{i=1}^n a_i$ . Se cere să se determine (dacă există) o submulțime având cât mai puține elemente și a căror sumă să fie egală cu  $C$ .

În acest caz  $X$  reprezintă ansamblul submulțimilor lui  $A$ . O abordare prin metoda forței brute a unei astfel de probleme necesită generarea tuturor submulțimilor lui  $A$ , reținerea celor care satisfac restricția (suma elementelor egală cu  $C$ ) și alegerea celei (celor) ce au cele mai puține elemente. Un astfel de algoritm ar avea ordinul de complexitate  $O(2^n)$ .

Pentru a rezolva mai eficient probleme de acest tip au fost dezvoltate tehnici specifice de elaborare a algoritmilor. Două dintre acestea sunt tehnica alegerii local optimale ("greedy") și cea a programării dinamice.

## 2 Principiul tehnicii

Să considerăm problema de optimizare formulată într-o variantă ușor diferită:

*Fie  $A = (a_1, a_2, \dots, a_n)$  un set finit de elemente (nu neapărat distincte). Să se determine  $S = (s_1, s_2, \dots, s_k) \subset A$  astfel încât  $S$  să satisfacă anumite restricții și să optimizeze un criteriu.*

În continuare prin set vom înțelege o mulțime în sens general (elementele sale nu sunt neapărat distincte). Principiul metodei constă în a construi succesiv soluția  $S$  pornind de la setul vid și adăugând la fiecare etapă un nou element, și anume acela care pare "optim" la momentul respectiv. Selecția elementului care va fi eventual adăugat urmărește apropierea de atingerea optimului. Însă alegerea care pare optimă la etapa curentă s-ar putea să nu fie cea mai bună și din punct de vedere

global astfel că e posibil să nu conducă la o soluție optimă. Totuși, o dată efectuată o alegere nu se mai revine asupra ei, aceasta fiind irevocabilă. Pe de altă parte, prin strategia greedy se poate obține o singură soluție chiar dacă problema are mai multe.

Structura generală a algoritmului este:

---

```

greedy1(A)
S ← ∅
WHILE "S nu este soluție" ∧ "există elemente neselectate în A" DO
    ||selectează a din A
    ||dacă nu sunt încălcate restricțiile se adaugă a la S
RETURN S

```

---

Etapa cea mai importantă este cea a selectării unui nou element. Pentru anumite probleme este posibil ca setul inițial,  $A$ , să fie ordonat după un criteriu determinat de criteriul de optim astfel că elementele vor fi alese chiar în ordinea în care apar în  $A$ . Criteriul de sortare depinde de problema concretă care se rezolvă. În acest caz algoritmul poate fi descris ceva mai detaliat astfel:

---

```

greedy2(A[1..n])
A[1..n] ← sortare(A[1..n]) // sortare după un criteriu corelat cu cel de optimizat
i ← 0 // contor de parcurgere a lui A
k ← 0 // contor utilizat în construirea lui S
WHILE "S[1..k] nu este soluție" ∧  $i \leq n$  DO
    ||IF "(S[1], ..., S[k], A[i]) satisface restricțiile" THEN
        || ||k ← k + 1; S[k] ← A[i] // se transferă elementul curent din A în S
        ||i ← i + 1 // se trece la următorul element din A
RETURN S[1..k]

```

---

În aplicații contează principiul metodei algoritmi construiți folosindu-l putând fi destul de diferiți de structurile generale *greedy1* respectiv *greedy2*.

*Exemplul 1.* (Problema submulțimii de cardinal dat și sumă maximă.) Se cere determinarea unui subset cu  $m$  elemente,  $S$ , dintr-un set  $A$  având  $n > m$  elemente astfel încât suma elementelor selectate să fie cât mai mare.

În acest caz este suficient să ordonăm descrescător elementele din  $A$  și să reținem primele  $m$  elemente:

---

```

submulțime(A[1..n],m)
A[1..n] ← sortare_descrescătoare(A[1..n])
FOR  $i \leftarrow 1, m$  DO S[i] ← A[i]
RETURN S[1..m]

```

---

Pentru această problemă se va demonstra în secțiunea următoare că strategia greedy produce întotdeauna o soluție optimă.

*Exemplul 2.* (Problema monedelor.) Presupunem că o sumă de bani  $C$  trebuie acoperită folosind cât mai puține monede. Dacă valorile monedelor sunt  $\{v_1, v_2, \dots, v_n\}$  atunci a rezolva problema înseamnă a determina  $(k_1, k_2, \dots, k_n)$  ( $k_i$  reprezintă numărul de monede de valoare  $v_i$  și poate fi egal cu 0) astfel încât să fie satisfăcută restricția  $\sum_{i=1}^n k_i v_i = C$  iar  $\sum_{i=1}^n k_i$  să fie cât mai mică. Se observă că această problemă este similară cu problema submulțimii de sumă dată și cardinal minim cu observația că setul  $A$  este infinit:  $A = \{v_1, v_1, \dots, v_2, v_2, \dots, \dots, v_n, v_n, \dots\}$  (pentru fiecare valoare  $v_i$  există un număr nelimitat de elemente).

O abordare de tip greedy ar conduce (în cazul în care numărul de monede de fiecare tip este nelimitat) la un algoritm de forma:

---

```

monede( $v[1..n], C$ )
 $v[1..n] \leftarrow \text{sortare\_descrescătoare}(v[1..n])$ 
FOR  $i \leftarrow 1, n$  DO  $S[i] \leftarrow 0$ 
 $i \leftarrow 1$ 
WHILE  $C > 0 \wedge i \leq n$  DO
    ||  $S[i] \leftarrow C \text{ DIV } v[i]$ 
    ||  $C \leftarrow C \text{ MOD } v[i]$ 
    ||  $i \leftarrow i + 1$ 
IF  $C = 0$  THEN RETURN  $S[1..n]$ 
ELSE "nu s-a gasit solutie"

```

---

*Observație.* Tehnica greedy nu conduce pentru orice instanță a problemei monedelor la soluția optimă. De exemplu pentru cazul monedelor cu valorile (25, 20, 10, 5, 1) și a valorii  $C = 40$  strategia greedy ar conduce la soluția (1, 0, 1, 1, 0) în timp ce varianta (0, 2, 0, 0, 0) este optimă. Pe de altă parte există situații în care problema nu are soluții. De exemplu pentru monede având valorile (20, 10, 5) și  $C = 17$  problema nu are soluție. Se observă ușor că dacă există monedă cu valoarea 1 atunci pentru orice sumă  $C$  se poate determina o soluție. Aceasta nu este însă neapărat optimală. Tehnica greedy conduce la o soluție optimă pentru problema monedelor doar dacă valorile acestora satisfac anumite restricții. Un exemplu de valori pentru care se poate demonstra optimalitatea soluției este:  $v_{i-1}$  este multiplu al lui  $v_i$  pentru fiecare  $i \geq 2$  (în condițiile în care  $v[1..n]$  este ordonat crescător).

### 3 Verificarea corectitudinii si analiza complexitatii

Tehnica alegerii local optimale este o tehnică intuitivă și eficientă, dar nu garantează obținerea soluției optime decât pentru anumite cazuri particulare de probleme.

**Verificarea corectitudinii.** Întrucât tehnica greedy nu asigură implicit optimalitatea soluției, aceasta trebuie demonstrată pentru cazul particular al problemei tratate. În general, problemele pentru care tehnica greedy poate fi aplicată cu succes au următoarele proprietăți:

- *Proprietatea de alegere greedy.* Înseamnă că se poate ajunge la soluția optimă global efectuând alegeri local optimale. Pentru a demonstra că o problemă are această proprietate se pornește de la o soluție optimă global și se arată că poate fi modificată astfel încât prima componentă să fie obținută efectuând o alegere local optimală. Se analizează în continuare restul soluției (problema se reduce la una similară dar de dimensiune mai mică). Aplicând principiul inducției matematice se arată că la fiecare pas poate fi efectuată o alegere de tip greedy. Această etapă se reduce de fapt la a arăta că problema are proprietatea de *substructură optimă*.
- *Proprietatea de substructură optimă.* Se consideră că o problemă are proprietatea de substructură optimă dacă o soluție optimă a problemei conține soluții optime ale subproblemelor. Pentru a demonstra acest lucru se poate folosi tehnica reducerii la absurd. Pornind de la o soluție optimă  $S(n) = (s_1, s_2, \dots, s_n)$  se arată că  $S(n-1) = (s_2, \dots, s_n)$  este soluție optimă pentru subproblema de dimensiune  $n-1$ . Se presupune că  $S(n-1)$  nu este optimă. Atunci ar exista  $S'(n-1) = (s'_2, \dots, s'_n)$  o soluție mai bună. Aceasta conduce de regulă la faptul că  $S'(n) = (s_1, s'_2, \dots, s'_n)$  este mai bună decât  $S(n)$  ceea ce contrazice ipoteza că  $S(n)$  este optimă.

*Exemplul 1.* Demonstrăm că algoritmul *submulțime* generează soluția optimă.

*Proprietatea de alegere greedy.* Fie  $O = (o_1, o_2, \dots, o_m)$  o soluție optimă a problemei. Cum nu contează ordinea elementelor în  $O$  putem presupune că  $o_1 \geq o_2 \geq \dots \geq o_m$ . Cum elementele lui  $A$  sunt ordonate descrescător ( $a_1 \geq a_2 \geq \dots \geq a_n$ ) prin alegere de tip greedy prima componentă ar trebui să fie  $a_1$ . Vom arăta că  $o_1 = a_1$  prin metoda reducerii la absurd. Presupunem că  $o_1 \neq a_1$ . Rezultă că  $a_1 > o_1$  iar soluția  $O' = (a_1, o_2, \dots, o_m)$  are proprietatea că  $\sum_{o \in O'} o - \sum_{o \in O} o = a_1 - o_1 > 0$  adică este mai bună decât  $O$ . Aceasta însă contrazice ipoteza că  $O$  este soluție optimă. Prin urmare  $o_1 = a_1$  adică primul element al soluției este ales conform strategiei greedy. Este suficient acum să demonstrăm că problema posedă proprietatea de substructură optimă.

*Proprietatea substructurii optime.* Fie  $O = (o_1, o_2, \dots, o_m)$  o soluție optimă a problemei inițiale. Presupunem că elementele lui  $A$  sunt ordonate descrescător ( $a_1 \geq a_2 \dots \geq a_n$ ). Arătăm că  $O_{(1)} = (o_2, \dots, o_m)$  este soluție optimă a problemei similare pentru  $A_{(2)} = (a_2, \dots, a_n)$  prin reducere la absurd. Dacă nu ar fi soluție optimă înseamnă că ar exista o altă soluție mai bună:  $(o'_2, \dots, o'_m)$  care completată pe prima poziție cu  $o_1$  ar conduce la o soluție globală mai bună decât  $O$ . S-a ajuns la o contradicție, deci problema posedă proprietatea substructurii optime.

*Exemplul 2.* Presupunem că valorile din *problema monedelor* satisfac proprietățile:  $v_1 > v_2 > \dots > v_n = 1$  și  $v_{i-1} = d_{i-1}v_i$  ( $v_{i-1}$  este multiplu al lui  $v_i$ ) pentru  $i = \overline{2, m}$ .

*Proprietatea alegerii greedy.* Fie  $O = (o_1, \dots, o_2)$  o soluție optimă ( $\sum_{i=1}^n o_i v_i = C$  și  $\sum_{i=1}^n o_i = K$  este minimă). Arătăm că prima componentă a lui  $O$  satisface proprietatea alegerii greedy, adică  $o_1 = \lfloor C/v_1 \rfloor$ . Fie  $C_2 = C - o_1 v_1$  suma ce rămâne după ce s-au dat  $o_1$  monede de valoare  $v_1$ . Analizăm două situații:

(i)  $C_2 < v_1$ . În acest caz  $C_2 = C \bmod v_1 = C - \lfloor C/v_1 \rfloor v_1$  deci  $o_1 = \lfloor C/v_1 \rfloor$ .

(ii)  $C_2 \geq v_1$ . Pornind de la  $O$  construim o nouă soluție  $O'$  caracterizată prin  $o'_1 = o_1 + \lfloor C_2/v_1 \rfloor$ . Suma ce rămâne de acoperit după ce s-au dat monedele de valoare  $v_1$  este de această dată  $C'_2 = C - o'_1 v_1 = C_2 - \lfloor C_2/v_1 \rfloor v_1 < C_2$ . Diferența  $C_2 - C'_2$  era acoperită în soluția  $O$  din monedele de valoare mai mică. Să considerăm cazul particular în care  $\lfloor C_2/v_1 \rfloor v_1 \leq o_2 v_2$  (diferența era acoperită doar cu monede de valoare  $v_2$ ). Noua soluție va fi atunci:  $O' = (o'_1, o'_2, o_3, \dots, o_n)$  cu  $o'_1 = o_1 + \lfloor C_2/v_1 \rfloor$  și  $o'_2 = o_2 - \lfloor C_2/v_1 \rfloor d_1$ . Aceasta acoperă valoarea:

$$o'_1 + o'_2 + o_3 + \dots + o_n = (o_1 + \lfloor C_2/v_1 \rfloor)v_1 + (o_2 - \lfloor C_2/v_1 \rfloor d_1)v_2 + \sum_{i=3}^n o_i v_i.$$

Cum  $v_1 = d_1 v_2$  rezultă că suma este chiar  $C$ . În ceea ce privește numărul de monede folosite, acesta este:

$$o'_1 + o'_2 + o_3 + \dots + o_n = o_1 + \lfloor C_2/v_1 \rfloor + o_2 - \lfloor C_2/v_1 \rfloor d_1 + \sum_{i=3}^n o_i = \sum_{i=1}^n o_i + \lfloor C_2/v_1 \rfloor (1 - d_1) < K$$

adică alegerea de tip greedy conduce la utilizarea a mai puține monede. Aceasta contrazice ipoteza că  $O$  este soluție optimă. Prin urmare primul element se alege după tehnica greedy. Același raționament se aplică și dacă  $\lfloor C_2/v_1 \rfloor v_1 > o_2 v_2$  doar că suma se acoperă din contribuția mai multor monede de valoare mai mică - posibil datorită relației de divizibilitate existente între valori).

*Proprietatea de substructură optimă.* Fie  $O = (o_1, \dots, o_n)$  o soluție optimă. Arătăm că  $O_{(2)} = (o_2, \dots, o_n)$  e soluție optimă pentru problema acoperirii sumei  $C_2 = C - o_1 v_1$  cu monede de valori  $v_2, v_3, \dots, v_n$ . Presupunem că  $O_{(2)}$  nu este optimă, adică există  $O'_{(2)} = (o'_2, \dots, o'_n)$  cu proprietățile  $\sum_{i=2}^n o'_i v_i = \sum_{i=2}^n o_i v_i = C_2$  și  $\sum_{i=2}^n o'_i < \sum_{i=2}^n o_i$ . Atunci  $O' = (o_1, o'_2, \dots, o'_n)$  verifică  $o_1 v_1 + \sum_{i=2}^n o'_i v_i = C$  și  $o_1 + \sum_{i=2}^n o'_i < o_1 + \sum_{i=2}^n o_i$ , adică  $O'$  este mai bună decât  $O$ , ceea ce contrazice ipoteza că  $O$  este optimă.

**Analiza complexității.** Strategia greedy conduce în general la algoritmi eficienți. Din structura generală (*greedy1*) se observă că prelucrarea care determină ordinul de complexitate este cea de selecție. Dacă aceasta este simplă se pot obține chiar algoritmi de complexitate liniară. În general însă aplicarea strategiei necesită o sortare prealabilă a elementelor lui  $A$  astfel că se ajunge la complexitate de tipul  $O(n^2)$  sau  $O(n \lg n)$  (dacă se folosește un algoritm eficient de sortare).

În cazul algoritmului *submulțime* dacă  $m$  este mic în raport cu  $n$  atunci nu este justificat să se ordoneze întregul set  $A$  ci să se aleagă la fiecare etapă valoarea maximă din cele ce nu au fost încă selectate. Se ajunge astfel la un ordin de complexitate  $O(mn)$ . În algoritmul *monede* sortarea domină celelalte prelucrări astfel că aceasta impune ordinul de complexitate ( $O(n^2)$  sau  $O(n \lg n)$ ).

## 4 Aplicații

**Problema rucsacului.** Este o problemă clasică ce are multe aplicații și diverse variante. Se consideră un set de obiecte caracterizate prin profitul ( $p$ ) și dimensiunea lor ( $d$ ):  $A = ((p_1, d_1), \dots, (p_n, d_n))$ . Se mai consideră un rucsac de capacitate maximă  $C$  și se pune problema selectării unui subset de obiecte,  $((p_{i_1}, d_{i_1}), \dots, (p_{i_k}, d_{i_k}))$  care să nu depășească capacitatea rucsacului ( $\sum_{j=1}^k d_{i_j} \leq C$ ) și să asigure un profit maxim ( $\sum_{j=1}^k p_{i_j}$  este maximă). Două dintre cele mai cunoscute variante ale problemei sunt:

- *Varianta discretă (0-1).* Obiectele nu pot fi divizate: un obiect este fie preluat în întregime fie nu este preluat.
- *Varianta continuă (fracționară).* Este posibil să fie transferate și fracțiuni din obiecte, profitul asigurat fiind proporțional cu fracțiunea.

Doar pentru varianta fracționară se poate obține o soluție optimă aplicând strategia greedy. În acest caz alegerea local optimă este cea care corespunde *profitului relativ* maxim (profitul relativ al obiectului  $i$  este  $p_i/d_i$ ). Aplicând strategia greedy se ajunge la a efectua următoarele prelucrări:

- se ordonează  $A$  descrescător după profitul relativ;
- se transferă obiectele în ordinea în care apar în  $A$ , în întregime cu excepția ultimului obiect din care se ia doar o fracțiune, atât cât să se umple rucsacul.

Pentru a descrie algoritmul facem următoarele convenții:  $A[i].p$ ,  $A[i].d$  reprezintă profitul respectiv dimensiunea obiectului  $i$ . Soluția va fi de forma:  $S = (S[1], \dots, S[n])$  cu  $S[i] \in [0, 1]$ , iar elementele sale vor fi interpretate astfel:  $S[i] = 0$  - obiectul  $i$  nu este selectat,  $S[i] = 1$  - obiectul  $i$  este selectat în întregime,  $S[i] \in (0, 1)$  - este selectată doar o fracțiune din obiectul  $i$ . Cu aceste convenții algoritmul poate fi descris astfel:

---

```

rucsac_fracționar( $A[1..n], C$ )
 $A[1..n] \leftarrow$  sortare_descrescătoare( $A[1..n]$ ) // sortare după profitul relativ
FOR  $i \leftarrow 1, n$  DO  $S[i] \leftarrow 0$ 
 $i \leftarrow 1$ ;
WHILE  $C > 0 \wedge i \leq n$  DO
    ||IF  $S[i].d \leq C$ 
    || THEN  $S[i] \leftarrow 1$ ;  $C \leftarrow C - A[i].d$ 
    || ELSE  $S[i] \leftarrow C/A[i].d$ ;  $C \leftarrow 0$ 
    || $i \leftarrow i + 1$ 
RETURN  $S[1..n]$ 

```

---

Soluția obținută aplicând acest algoritm va fi de forma:  $S = (1, 1, \dots, 1, f, 0, \dots, 0)$ . În plus  $\sum_{i=1}^n s_i d_i = C$ , astfel că în continuare considerăm că restricția problemei este de tip egalitate.

Demonstrăm că *rucsac\_fracționar* generează o soluție optimă arătând că orice soluție optimă trebuie să respecte criteriul alegerii greedy.

Considerăm că obiectele sunt ordonate descrescător după valoarea profitului relativ:  $p_1/d_1 > p_2/d_2 > \dots > p_n/d_n$  (în cazul când inegalitatea nu este strictă alegerea de tip greedy nu este unică și complică puțin demonstrația). Considerăm  $O = (o_1, o_2, \dots, o_n)$  o soluție optimă și demonstrăm că este de tip greedy prin reducere la absurd. Presupunem că  $O$  nu este de tip greedy și considerăm că  $O' = (o'_1, \dots, o'_n)$  este generată aplicând tehnica greedy. Fie  $B_+ = \{i | o'_i \geq o_i\}$ ,  $B_- = \{i | o'_i < o_i\}$  iar  $k = \min B_-$  (cel mai mic indice pentru care  $o'_i < o_i$ ). Din structura unei soluții de tip greedy rezultă că dacă  $i \in B_+$  și  $j \in B_-$  atunci  $i < j$ . Din restricția problemei rezultă  $\sum_{i=1}^n o_i d_i = \sum_{i=1}^n o'_i d_i$  adică  $\sum_{i \in B_+} (o'_i - o_i) d_i = \sum_{i \in B_-} (o_i - o'_i) d_i$ . Calculăm profiturile corespunzătoare celor două soluții:  $P = \sum_{i=1}^n o_i p_i$  și  $P' = \sum_{i=1}^n o'_i p_i$ . Diferența lor este:

$$P' - P = \sum_{i=1}^n (o'_i - o_i) p_i = \sum_{i \in B_+} (o'_i - o_i) d_i \frac{p_i}{d_i} - \sum_{i \in B_-} (o_i - o'_i) d_i \frac{p_i}{d_i}.$$

Se observă că  $p_i/d_i > p_k/d_k$  pentru orice  $i \in B_+$  iar  $p_i/d_i \leq p_k/d_k$  pentru orice  $i \in B_-$ . Prin urmare

$$P' - P > \frac{p_k}{d_k} \sum_{i \in B_+} (o'_i - o_i) d_i - \frac{p_k}{d_k} \sum_{i \in B_-} (o_i - o'_i) d_i = 0$$

Deci  $P' > P$  ceea ce contrazice ipoteza că  $O$  este optimă. Deci  $O$  are o structură de tip greedy. Proprietatea de substructură optimă rezultă imediat prin reducere la absurd.

*Observație.* Pentru a ilustra faptul că pentru varianta discretă această strategie nu conduce la soluția optimă considerăm exemplul:  $A = ((60, 10), (100, 20), (120, 30))$ ,  $C = 50$ . Profiturile relative sunt: 6, 5, 4, iar  $A$  este deja ordonat descrescător după acest criteriu. Aplicând strategia greedy s-ar transfera primele două obiecte ducând la un profit egal cu 160. Dacă s-ar transfera al doilea și al treilea obiect s-ar ajunge la un profit mai mare, 220.

Deși nu conduce la soluția optimă, strategia greedy poate fi aplicată și pentru varianta discretă conducând la o soluție sub-optimală (se poate demonstra că soluția obținută aplicând tehnica greedy satisface:  $\sum_{i=1}^n s_i p_i \geq P_{opt} - \max_i p_i$ ,  $P_{opt}$  fiind profitul corespunzător soluției optime). În schimb se poate obține o soluție optimă aplicând tehnica programării dinamice.

**Problema selectării activităților.** Se consideră un set de activități care au nevoie de o anumită resursă și la un moment dat o singură activitate poate beneficia de resursa respectivă (de exemplu activitatea poate fi un examen, iar resursa o sală de examen). Presupunem că pentru fiecare activitate,  $A_i$ , se cunoaște momentul de începere,  $p_i$  și cel de finalizare  $t_i$  ( $t_i > p_i$ ). Presupunem că activitatea se desfășoară în intervalul  $[p_i, t_i)$ . Două activități  $A_i$  și  $A_j$  se consideră *compatibile* dacă intervalele asociate sunt disjuncte. Se cere să se selecteze un număr cât mai mare de activități compatibile.

O soluție a acestei probleme constă într-un subset de activități  $S = (a_{i_1}, \dots, a_{i_m})$  care satisfac  $[p_{i_j}, t_{i_j}) \cap [p_{i_k}, t_{i_k}) = \emptyset$  pentru orice  $j \neq k$ .

Criteriul de selecție ar putea fi: (i) cea mai mică durată; (ii) cel mai mic moment de începere; (iii) cel mai mic moment de sfârșit; (iv) intervalul care se intersectează cu cele mai puține alte intervale.

Dintre aceste variante cea pentru care se poate demonstra că asigură obținerea soluției optime este a treia: la fiecare etapă se alege activitatea care se termină cel mai devreme.

Notând cu  $A[i].p$ ,  $A[i].t$  momentul de începere respectiv cel de final al activității  $A[i]$  algoritmul bazat pe strategia greedy poate fi descris astfel:

---

```

selecție_activități( $A[1..n]$ )
 $A[1..n] \leftarrow$  sortare_crescătoare( $A[1..n]$ ) // sortare după timpul de finalizare
 $S[1] \leftarrow A[1]$ 
 $i \leftarrow 2; k \leftarrow 1$ 
WHILE  $i \leq n$  DO
    ||IF  $S[k].t \leq A[i].p$ 
    || THEN  $k \leftarrow k + 1; S[k] \leftarrow A[i];$ 
    || $i \leftarrow i + 1$ 
RETURN  $S[1..k]$ 

```

---

Demonstrăm că algoritmul de mai sus conduce la soluția optimă. După ordonarea crescătoare după timpul de final avem  $t_1 \leq t_2 \leq \dots \leq t_n$ . Considerăm o soluție optimă  $O = ((p_{i_1}, t_{i_1}), \dots, (p_{i_m}, t_{i_m}))$ . Evident  $t_{i_1} \geq t_1$  prin urmare înlocuind  $A_{i_1}$  cu  $A_1$  în  $O$ , se obține o soluție în care există același număr de activități compatibile dar prima activitate este aleasă conform strategiei greedy. Deci problema satisface proprietate alegerii local optimale. O dată aleasă prima activitate, problema se reduce la planificarea optimă a activităților compatibile cu prima. Dacă  $O$  este soluția optimă a problemei inițiale atunci  $O' = ((p_{i_2}, t_{i_2}), \dots, (p_{i_m}, t_{i_m}))$  este soluție optimă a subproblemei la care se ajunge după stabilirea primei activități. Astfel problema satisface și proprietatea substructurii optime.

Să considerăm cazul  $A = ((1, 8), (2, 5), (6, 8), (5, 6))$ . Sortând  $A$  în ordinea crescătoare a duratei activităților se obține ca soluție:  $S = ((5, 6), (6, 8), (2, 5))$ . Prin selecție după momentul de începere a activităților se obține  $((1, 8))$ , în schimb prin selecție după momentul de finalizare se obține o soluție,  $S = ((2, 5), (5, 6), (6, 8))$ , în care activitățile sunt deja ordonate în ordinea în care vor fi efectuate.

**Problema planificării activităților.** Se consideră un set de  $n$  prelucrări ce trebuie executate de către un procesor. Durata execuției prelucrărilor este aceeași (se consideră egală cu 1). Fiecare prelucrare  $i$  are asociat un termen final de execuție,  $t_i \leq n$  și un profit,  $p_i$ . Profitul unei prelucrări intervine în calculul profitului total doar dacă prelucrarea este executată (dacă prelucrarea nu poate fi planificată înainte de termenul final de execuție profitul este nul). Se cere să se planifice activitățile astfel încât să fie maximizat profitul total (acesta este corelat cu numărul de prelucrări planificate).

O soluție constă în stabilirea unui "orar" de execuție a prelucrărilor  $S = (s_1, s_2, \dots, s_n)$ ,  $s_i \in \{1, \dots, n\}$  fiind indicele prelucrării planificate la momentul  $i$ . Pentru rezolvarea problemei se poate aplica o tehnică de tip greedy caracterizată prin:

- se sortează activitățile în ordinea descrescătoare a profitului;
- fiecare activitate se planifică într-un interval liber cât mai apropiat de termenul final de execuție.

---

```

planificare( $A[1..n]$ )
 $A[1..n] \leftarrow$  sortare_descrescătoare( $A[1..n]$ ) // sortare după profit ( $A[i].p$ )
FOR  $i \leftarrow 1, n$  DO  $S[i] \leftarrow 0$ 
FOR  $i \leftarrow 1, n$  DO
    || $poz \leftarrow A[i].t$  // termenul final de execuție
    ||WHILE  $S[poz] \neq 0 \wedge poz \geq 1$  DO  $poz \leftarrow poz - 1$ 
    ||IF  $poz \neq 0$  THEN  $s[poz] \leftarrow i$  // se planifica activitatea
    ||// altfel activitatea  $i$  nu poate fi planificată
RETURN  $S[1..n]$ 

```

---

Se observă că dacă pentru fiecare  $i \in \{1, \dots, n\}$  numărul activităților care au termenul final cel mult  $i$  este cel mult egal cu  $i$  ( $\text{card}\{j | t_j \leq i\} \leq i$ ) atunci toate activitățile vor fi planificate, altfel vor exista activități ce nu pot fi planificate.

Să considerăm  $n = 4$  activități având termenele finale:  $(2, 3, 4, 2)$  și profiturile corespunzătoare  $(4, 3, 2, 1)$ . Atunci aplicând tehnica greedy se obține soluția  $(4, 1, 2, 3)$ . Dacă însă termenele de execuție sunt:  $(2, 4, 1, 2)$  și aceleași profituri se obține soluția  $(3, 1, 0, 2)$  iar activitatea 4 nu este planificată.

## 5 Exerciții

1. *Problema împachetării.* Se consideră un set de numere  $a_1, a_2, \dots, a_n$  cu proprietatea că  $a_i \in (0, 1]$ . Se cere să se grupeze în cât mai puține subseturi ( $k$ ) astfel încât suma elementelor din fiecare subset să nu depășească valoarea 1.

*Indicație.* Elementele setului inițial pot fi în orice ordine. *Varianta 1.* Se construiesc succesiv submulțimile: se transferă (în ordinea în care se află în set) în primul subset atâtea elemente cât este posibil, din elementele rămase se transferă elemente în al doilea subset etc. (nu e garantată optimalitatea soluției însă s-a demonstrat că  $k < 2k_{opt}$ ,  $k$  fiind numărul de subseturi generat prin strategia greedy de mai sus, iar  $k_{opt}$  este numărul optim de subseturi). *Varianta 2.* Se initializează  $n$  subseturi vide. Se parcurge setul de numere și fiecare număr se transferă în primul subset în care "încap". Numărul de subseturi nevide astfel constituite  $k$ , are proprietatea  $k < 1.7k_{opt}$ . *Varianta 3.* Dacă se ordonează descrescător setul inițial și se aplică varianta 2 se obține o îmbunătățire:  $k < 11/9k_{opt} + 4$ .

2. Să se aplice strategia greedy pentru a rezolva problema discretă a rucsacului.

*Indicație.* Se ordonează obiectele descrescător după profitul relativ și se selectează în această ordine. Obiectele care nu încap în rucsac în etapa selectării lor sunt ignorate.

3. Se consideră o variantă a problemei rucsacului în care ordonarea obiectelor după profit coincide cu ordonarea obiectelor după dimensiune (un caz particular este acela în care sunt egale). Să se elaboreze un algoritm bazat pe strategia greedy în care selecția se face în ordinea profiturilor absolute. Conduce algoritmul la soluția optimă ?
4. Se consideră o variantă a problemei rucsacului în care toate obiectele au aceeași dimensiune (dar profituri diferite). Să se elaboreze un algoritm bazat pe strategia greedy care să rezolve problema. Se obține soluția optimă ? Aceeași problemă pentru cazul în care toate profiturile sunt identice (dar dimensiunile sunt diferite).