ALGORITMICĂ. Seminar 12: Aplicaţii ale tehnicii "greedy". Aplicaţii ale programării dinamice.

1.*Problema împachetării.* Se consideră un set de numere $a_1, a_2, \ldots, a_n$ cu proprietatea că $a_i \in (0, C]$. Se cere să se grupeze în cât mai puţine subseturi (k) astfel încât suma elementelor din fiecare subset să nu depăşească valoarea C.
Rezolvare. Există mai multe variante ce folosesc principiul alegerii greedy însă toate sunt suboptimale (nu garantează obţinerea optimului ci doar a unei soluţii suficient de apropiate de cea optimă).

Varianta 1. Se construiesc succesiv submulţimile: 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 soluţiei însă s-a demonstrat că $k < 2k_{opt}$, k fiind numărul se subseturi generat prin strategia greedy de mai sus, iar k_{opt} este numărul optim de subseturi).

Varianta 2. Se initializeaza n subseturi vide. Se parcurge setul de numere şi fiecare număr se transferă în primul subset în care "incape". Numărul de subseturi nevide astfel constituite k, are proprietatea $k < 1.7k_{opt}$.

Varianta 3. Dacă se ordonează descrescător setul iniţial şi se aplică varianta 2 se obţine o îmbunătăţire: $k < 11/9k_{opt} + 4$.

Considerăm că soluţia va fi reprezentată printr-o matrice $R[1..n, 1..n]$ în care linia i corespunde subsetului i iar $R[i, j]$ conţine fie 0 fie indicele unui element din a care trebuie plasat în subsetul i. Pentru a controla modul de completare a subseturilor se pot folosi tablourile $K[1..n]$ ($K[i]$ specifică indicele ultimului element completat în linia i) şi $S[1..n]$ care conţine suma curentă a elementelor de pe linia i.

```
impachetare(integer a[1..n])
R[1..n, 1..n] ← 0; K[1..n] ← 0; S[1..n] ← 0; // initializarea cu 0 a elementelor tuturor tablourilor
a[1..n] ← sortare_descrescatoare(a[1..n])
for i ← 1, n do
    j ← 1
    while S[j] + a[i] > C do j ← j + 1
    K[j] ← K[j] + 1; R[j, K[j]] ← a[i]; S[j] ← S[j] + a[i]
endfor
return R[1..n, 1..n]
```

2.*Problema subşirului strict crescător.* Fie $a_1, a_2, \ldots, a_n$ un şir de valori reale pozitive. Să se determine un subşir strict crescător pentru care suma elementelor este maximă.

Rezolvare. Se parcurg etapele aplicării metodei programării dinamice:
(a) *Analiza structurii unei soluţii optime.* Fie $a_{j_1} < \ldots < a_{j_{k-1}} < a_{j_k}$ un subşir strict crescător pentru care suma elementelor este maximă. Presupunând că $a_{j_{k-1}} = a_i$ rezultă că $a_{j_1} < \ldots < a_{j_{k-1}}$ este un subşir strict crescător de lungime maximă care se termină în a_i (în caz contrar ar exista un subşir strict crescător al şirului iniţial pentru care suma elementelor este mai mare decât $\sum_{l=1}^{k} a_{j_l}$, contrazicând ipoteza că $a_{j_1}, \ldots, a_{j_{k-1}}, a_{j_k}$ este soluţie optimă). Deci o soluţie optimă a problemei generale $P(i)$ a determinării unui subşir strict crescător care se termină în a_i şi care are suma elementelor maximă conţine o soluţie optimă a uneia dintre subproblemele P_j pentru care $j < i$ şi $a_j < a_i$.

(b) *Deducerea unei relaţii de recurenţă.* Valoarea care trebuie maximizată este suma elementelor subşirului, astfel că relaţia de recurenţă ce specifică legătura între soluţia problemei şi soluţiile subproblemelor:

$$S_i = \begin{cases} a_i & i = 1 \\ a_i + max\{S_j | a_j < a_i, j = \overline{1, i-1}\} & i > 1 \end{cases}$$

(c) *Dezvoltarea relaţiei de recurenţă.* Se completează tabelul $S[1..n]$, unde $S[i]$ este suma maximă corespunzătoare subşirurilor strict crescătoare care se termină în $a[i]$. În tabloul $P[1..n]$ se reţin indicii pentru care se atinge valoarea maximă.

```
calcul(real a[1..n], S[1..n], P[1..n])
S[1] ← a[1]; P[1] ← 0
for i ← 2, n do
    imax ← 0; max ← 0
    for j ← 1, i − 1 do
        if a[j] < a[i] AND max < S[j] then
            imax ← j; max ← S[j]
        endif
    endfor
    P[i] ← imax
endfor
```

(d) *Construirea soluţiei.* Subşirul se construieşte pornind de la ultimul element. Acesta corespunde valorii maxime din tabloul $S[1..n]$. După completarea unui element în subşir se identifică elementul imediat anterior folosind tabloul $P[1..n]$. La final elementele tabloului în care este reţinută soluţia se inversează.

```
solutie(real a[1..n])
calcul(a[1..n], S[1..m], P[1..m])
imax ← 1
for i ← 2, n do
    if S[imax] < S[i] then imax ← i endif
endfor
k ← 1; R[k] ← a[imax]; i ← P[imax]
while i > 0 do
    k ← k + 1
    R[k] ← a[i]
    i ← P[i]
endwhile
R[1..k] ←inversare(R[1..k])
```

3.*Problema celui mai lung subşir comun a două şiruri.* Fie $A = (a_1, a_2, \ldots, a_m)$ şi $B = (b_1, b_2, \ldots, b_n)$ două şiruri. Să se determine cel mai lung subşir comun al celor două şiruri, adică $(c_1, c_2, \ldots, c_l)$ cu proprietatea că există $i_1 < i_2 < \ldots < i_l$ şi $j_1 < j_2 < \ldots < j_l$ astfel încât $c_k = a_{i_k} = b_{j_k}$ pentru $k = \overline{1, l}$.

Rezolvare. De exemplu pentru şirurile $a = (2, 1, 4, 3, 2)$ şi $b = (1, 3, 4, 2)$ există două subşiruri comune de lungim maximă (3) şi anume: $(1, 4, 2)$ şi $(1, 3, 2)$.

(a) *Analiza structurii unei soluţii optime.* Fie $c = (c_1, c_2, \ldots, c_{l-1}, c_l)$ o soluţie optimă. Presupunem că $c_l = a_i = b_j$. c poate fi considerată soluţie optimă a problemei $P(i, j)$ ce constă în determinarea

celui mai lung subşir comun al şirurilor $a_{1..i} = (a_1, \ldots, a_i)$ şi $b_{1..j} = (b_1, \ldots, b_j)$. Se poate demonstra prin reducere la absurd că $(c_1, c_2, \ldots, c_{l-1}$ este soluţie optimă a subproblemei $P(i-1, j-1)$.
(b) *Deducerea unei relaţii de recurenţă.* Se notează cu $L(i, j)$ numărul de elemente al celui mai lung subşir comun al lui $(a_1, a_2, \ldots, a_i)$ şi $(b_1, b_2, \ldots, b_j)$. Relaţia de recurenţă pentru calculul lui $L(i, j)$ este:

$$L(i,j) = \begin{cases} 0 & \text{dacă } i = 0 \text{ sau } j = 0 \\ L(i-1, j-1) + 1 & \text{dacă } a_i = b_j \\ \max\{L(i-1, j), L(i, j-1)\} & \text{în celelalte cazuri} \end{cases}$$

În cazul exemplului de mai sus matricea L are următoarea structură:

0	0	0	0	0
0	0	0	0	1
0	1	1	1	1
0	1	1	2	2
0	1	2	2	2
0	1	2	2	3

(c) *Dezvoltarea relaţiei de recurenţă.*

```
construire (integer a[1..m], b[1..n])
for i ← 0, m do L[i, 0] ← 0 endfor
for j ← 0, n do L[0, j] ← 0 endfor
for i ← 1, m do
   for j ← 1, n do
      if a[i] = b[j] then L[i, j] ← L[i − 1, j − 1] + 1
      else if L[i − 1, j] > L[i, j − 1] then L[i, j] ← L[i − 1, j]
           else L[i, j] ← L[i, j − 1]
         endif
      endif
   endfor
endfor
return L[0..m, 0..n]
```

(d) *Construirea soluţiei.* Dacă $a_i = b_j$ atunci soluţia problemei $P(i, j)$ se obţine din soluţia subproblemei $P(i-1, j-1)$ prin adăugarea valorii comune celor două şiruri $a_i = b_j$. Dacă $a_i \neq b_j$ atunci soluţia lui $P(i, j)$ coincide cu soluţia lui $P(i-1, j)$ (dacă $L[i-1, j] \geq L[i, j-1]$) respectiv cu soluţia lui $P(i, j-1)$ (dacă $L[i-1, j] < L[i, j-1]$). Presupunând că tabloul în care se colectează soluţia (x) şi variabila ce conţine numărul de elemente ale subşirului comun (k) sunt variabile globale (iar k este iniţializată cu 0), algoritmul pentru construirea soluţiei poate fi descris recursiv după cum urmează:

```
solutie(i, j)
if L[i, j] ≠ 0 then
    if a[i] = b[j] then
        solutie(i-1,j-1)
        k ← k + 1
        x[k] ← a[i]
    else
        if L[i − 1, j] ≥ L[i, j − 1] then solutie(i − 1, j)
              elsesolutie(i, j − 1)
        endif
    endif
endif
```

4. *Problema monedelor.* Se consideră monede de valori $d_n > d_{n-1} > \ldots > d_1 = 1$. Să se găsească o acoperire minimală (ce foloseşte cât mai puţine monede) a unei sume date S. *Rezolvare.* Întrucât există monedă de valoare 1 orice sumă poate fi acoperită exact. In cazul general, tehnica greedy (bazată pe ideea de a acoperi cât mai mult posibil din suma cu moneda de valoarea cea mai mare) nu conduce întotdeauna la soluţia optimă. De exemplu dacă $S = 12$ şi se dispune de monede cu valorile 10, 6, 1 atunci aplicând tehnica greedy s-ar folosi o monedă de valoare 10 şi două monede de valoare 1. Soluţia optimă este însă cea care foloseşte două monede de valoare 6.
(a) *Analiza structurii unei soluţii optime.* Considerăm problema generică $P(i, j)$ care constă în acoperirea sumei j folosind monede de valori $d_1 < \ldots < d_i$. Soluţia optimă corespunzătoare acestei probleme va fi un şir, $(s_1, s_2, \ldots, s_k)$ de valori corespunzătoare monedelor care acoperă suma j. Dacă $s_k = d_i$ atunci $(s_1, s_2, \ldots, s_{k-1})$ trebuie să fie soluţie optimă pentru subproblema $P(i, j - d_i)$ (i rămâne nemodificat întrucât pot fi folosite mai multe monede de aceeaşi valoare). Dacă $s_k \not= d_i$ atunci $(s_1, s_2, \ldots, s_{k-1})$ trebuie să fie soluţie optimă a subproblemei $P(i - 1, j)$.
(b) *Deducerea relaţiei de recurenţă.* Fie $R(i, j)$ numărul minim de monede de valori $d_1, \ldots, d_i$ care acoperă suma j. $R(i, j)$ satisface:

$$R(i,j) = \begin{cases} 0 & j = 0 \\ j & i = 1 \\ R(i-1,j) & j < d_i \\ \min\{R(i-1,j), 1 + R(i, j-d_i)\} & j \geq d_i \end{cases}$$

Pentru exemplul de mai sus se obţine matricea:

0	0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	2	3	4	5	6	7	8	9	10	11	12
6	0	1	2	3	4	5	1	2	3	4	5	6	2
10	0	1	2	3	4	5	1	2	3	4	1	2	2

Pentru a uşura construirea soluţiei se poate construi ină o matrice $Q[1..n, 0..S]$ caracterizată prin:

$$Q(i,j) = \begin{cases} 1 & d_i \text{ se foloseşte pentru acoperirea sumei } j \\ 0 & d_i \text{ nu se foloseşte pentru acoperirea sumei } j \end{cases}$$

In cazul exemplului analizat matricea Q va avea următorul conţinut:

	0	1	2	3	4	5	6	7	8	9	10	11	12
1	0	1	1	1	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	0	1	1	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	0	1	1	0

(c) *Dezvoltarea relaţiei de recurenţă.* Matricile R şi Q pot fi completate după cum urmează:

```
completare(d[1..n],S)
for i ← 1, n do R[i, 0] ← 0 endfor
for j ← 1, s do R[1, j] ← j endfor
for i ← 1, n do
   for j ← 1, s do
      if j < d[i] then R[i, j] ← R[i − 1, j]; Q[i, j] ← 0
      else
         if R[i − 1, j] < 1 + R[i, j − d[i]] then R[i, j] ← R[i − 1, j]; Q[i, j] ← 0
         else R[i, j] ← R[i, j − d[i]] + 1;Q[i, j] ← 1
         endif
      endif
   endfor
endfor
```

$R[n, s]$ reprezintă numărul minim de monede necesare pentru a acoperi suma S.

(d) *Construirea solutiei.* Considerând că tabloul a în care se colectează soluţia şi variabila k ce conţine numărul de elemente ale lui a sunt variabile globale, soluţia poate fi construită în manieră recursivă în modul următor:

```
solutie(i, j)
if j ≠ 0 then
   if Q[i, j] = 1 then
      k ← k + 1
      a[k] ← d[i]
      solutie(i,j-d[i])
   else solutie(i − 1, j)
   endif
endif
```

5. *Problema submulţimii de sumă dată.* Fie $A = \{a_1, \ldots, a_n\}$ o mulţime de valori naturale şi c un număr natural. Să se determine o submulţime $S \subset A$ pentru care suma elementelor nu depăşeşte valoarea c dar este maximală ($\sum_{s \in S} s \leq c$ şi $\sum_{s \in S} s$ este maximă).

Rezolvare. In cazul general problema $P(i, j)$ se referă la determinarea unei submulţimi a mulţimii $\{a_1, \ldots, a_i\}$ pentru care suma elementelor este cât mai apropiată de j. Notând cu $S(i, j)$ suma elementelor unei soluţii optimale a problemei $P(i, j)$ relaţia de recurenţă corespunzătoare este:

$$S(i,j) = \begin{cases} 0 & i = 0 \text{ sau } j = 0 \\ S(i-1,j) & a_i > j \\ \max\{S(i-1,j), S(i-1,j-a_i) + a_i\} & a_i \leq j \end{cases}$$

O dată completată matricea $S[0..n, 0..c]$, soluţia $R[1..k]$ poate fi construită în manieră recursivă:

```
solutie(i, j)
if i ≠ 0 AND j ≠ 0 do
    if a[i] > j OR (a[i] ≤ j AND S[i − 1, j] > S[i − 1, j − a_i] + a_i)then solutie(i − 1, j)
    else solutie(i − 1, j − a[i]); k ← k + 1; R[k] ← a[i]
    endif
endif
```

Temă (termen: 8 ianuarie - 13 ianuarie)

1. Algoritm corespunzător primei variante de rezolvare a problemei împachetării.
2. Se consideră un şir de valori reale (pozitive şi negative). Să se determine subşir de elemente cu semne alternate (un element pozitiv este urmat de un element negativ iar un element negativ este urmat de un elemente pozitiv) pentru care suma valorilor modulelor este maximă.
3. Propuneţi o variantă iterativă pentru construirea soluţiei în cazul problemei monedelor.
4. Fie A o mulţime de valori naturale. Aplicând tehnica programării dinamice să se descompună mulţimea A în două submulţimi B şi C astfel încât $A = B \cup C$, $B \cap C = \emptyset$ şi suma elementelor din B este cât mai apropiată de suma elementelor din C.