
Algoritmi și structuri de date (I). Seminar 12: Aplicații ale programării dinamice (I).

- familiarizare cu clasele de probleme care pot fi rezolvate folosind tehnica programării dinamice - verificarea proprietății de substructură optimă
 - construirea relației de recurență
 - dezvoltarea relației de recurență
 - construirea soluției
-

Problema 1 (*Problema determinării unui subșir strict crescător de sumă maximă.*) Fie a_1, a_2, \dots, a_n un șir de valori reale pozitive. Să se determine un subșir strict crescător pentru care suma elementelor este maximă. De exemplu, pentru $a = (2, 5, 1, 3, 6, 8, 4)$ subșirul $(2, 5, 6, 8)$ are această proprietate.

Indicație. Se aplică ideea de la determinarea celui mai lung subșir strict crescător doar că în loc să se contorizeze numărul de elemente din subșir se adună valoarea lor.

a) *Caracterizarea soluției optime.* Fie $a_{j_1} < a_{j_2} < \dots < a_{j_{k-1}} < a_{j_k}$ un subșir strict crescător de lungime maximă. Considerăm că $a_{j_k} = a_i$ iar $a_{j_{k-1}} = a_p$ (evident $p < i$). Prin reducere la absurd se poate arăta că $a_{j_1} < a_{j_2} < \dots < a_{j_{k-1}}$ este subșir strict crescător al șirului parțial (a_1, a_2, \dots, a_p) care are cea mai mare sumă a elementelor dintre toate subșirurile care se termină în a_p (dacă ar exista un subșir cu suma mai mare atunci prin adăugarea lui a_i s-ar obține un subșir al lui $a[1..i]$ care ar avea suma mai mare decât a subșirului considerat optim). Prin urmare problema are proprietatea de substructură optimă și suma elementelor unui subșir crescător care se termină în a_i poate fi exprimată în funcție de suma elementelor subșirurilor crescătoare care se termină cu elemente a_p ce satisfac $a_p < a_i$.

b) *Stabilirea unei relații de recurență între sumele elementele subșirurilor.* Fie $(S_i)_{i=1, n}$ un tablou ce conține pe poziția i suma elementelor celui mai lung subșir strict crescător al șirului (a_1, \dots, a_n) care are pe a_i ca ultim element (spunem că subșirul se termină în a_i). Ținând cont de proprietățile soluției optime se poate stabili o relație de recurență pentru S_i :

$$S_i = \begin{cases} 1 & i = a[1] \\ a[i] + \max\{S_j \mid 1 \leq j < i \text{ și } a_j < a_i\} & i > 1 \end{cases}$$

Dacă mulțimea $M_i = \{S_j \mid 1 \leq j < i \text{ și } a_j < a_i\}$ este vidă atunci $\max M_i = 0$. De exemplu pentru șirul inițial $a = (2, 5, 1, 3, 6, 8, 4)$ tabloul lungimilor subșirurilor optime care se termină în fiecare element al lui a este: $S = (2, 7, 1, 4, 13, 21, 8)$. Cea mai mare valoare din S indică suma maximă a elementelor dintr-un subșir strict crescător.

c) *Dezvoltarea relației de recurență în manieră ascendentă.* Se completează un tablou cu elementele lui $(S_i)_{i=1, n}$ așa cum este descris în Algoritmul 1.

Luând în considerare doar comparația dintre două elemente ale șirului ($a[j] < a[i]$) costul algoritmului este $T(n) = \sum_{i=2}^n \sum_{j=1}^{i-1} 1 = \sum_{i=2}^n (i-1) = n(n-1)/2$ deci completarea tabelului S este de complexitate $\Theta(n^2)$.

d) *Construirea unei soluții optime.* Se determină cea mai mare valoare din S . Aceasta indică suma maximă a elementelor dintr-un cel mai lung subșir strict crescător iar poziția pe care se află indică elementul, $a[m]$, din șirul inițial cu care se termină subșirul. Pornind de la acest element subșirul se construiește în ordinea inversă a elementelor sale căutând la fiecare etapă un element din șir mai mic decât ultimul completat în subșir și căruia îi corespunde în S o valoare care poate fi obținută din cea curentă prin scăderea elementului corespunzător din șir. Algoritmul este descris în 2.

Dacă după completarea elementului $a[m]$ în subșir există două subșiruri din $a[1..m-1]$ cu suma maximă a elementelor egală cu $B[m] - a[m]$: unul care se termină în $a[p]$ și unul care se termină în $a[q]$ (astfel încât $a[p] < a[m]$ și $a[q] < a[m]$), algoritmul va selecta ca element pentru subșir pe cel cu indicele mai apropiat de m , adică mai mare. Astfel dacă $p < q$ va fi selectat q .

Algorithm 1 Dezvoltarea relației de recurență pentru determinarea celui mai lung subșir strict crescător

```

completare( $a[1..n]$ )
 $S[1] \leftarrow a[1]$ 
for  $i \leftarrow 2, n$  do
     $imax \leftarrow 1$ 
    for  $j \leftarrow 1, i - 1$  do
        if ( $a[j] < a[i]$ ) and ( $S[imax] < S[j]$ ) then
             $imax \leftarrow j$ 
        end if
    end for
     $S[i] \leftarrow S[imax] + a[i]$ 
end for
return  $S[1..n]$ 

```

Întrucât în ciclul de construire căutarea continuă de la poziția noului element selectat, chiar dacă sunt două cicluri **while** suprapuse ordinul de complexitate este $\mathcal{O}(n)$. Cum și determinarea maximului lui S are același ordin de complexitate rezultă că algoritmul de construire are complexitate liniară.

Problema 2 (*Problema monedelor.*) Se consideră monede de valori $d_n > d_{n-1} > \dots > d_1 = 1$. Să se găsească o acoperire minimală (ce folosește cât mai puține monede) a unei sume date S .

Rezolvare. Întrucât există monedă de valoare 1 orice sumă poate fi acoperită exact. În cazul general, tehnica greedy (bazată pe ideea de a acoperi cât mai mult posibil din suma cu moneda de valoare cea mai mare) nu conduce întotdeauna la soluția optimă. De exemplu dacă $S = 12$ și se dispune de monede cu valorile 10, 6, 1 atunci aplicând tehnica greedy s-ar folosi o monedă de valoare 10 și două monede de valoare 1. Soluția optimă este însă cea care folosește două monede de valoare 6.

(a) *Analiza structurii unei soluții optime.* Considerăm problema generică $P(i, j)$ care constă în acoperirea sumei j folosind monede de valori $d_1 < \dots < d_i$. Soluția optimă corespunzătoare acestei probleme va fi un șir, (s_1, s_2, \dots, s_k) de valori corespunzătoare monedelor care acoperă suma j . Dacă $s_k = d_i$ (se folosește monedă cu valoarea d_i) atunci $(s_1, s_2, \dots, s_{k-1})$ trebuie să fie soluție optimă pentru subproblema $P(i, j - d_i)$ (i rămâne nemodificat întrucât pot fi folosite mai multe monede de aceeași valoare). Dacă $s_k \neq d_i$ (nu se folosește monedă cu valoarea d_i) atunci $(s_1, s_2, \dots, s_{k-1})$ trebuie să fie soluție optimă a subproblemei $P(i - 1, j)$.

(b) *Deducerea relației de recurență.* Fie $R(i, j)$ numărul minim de monede de valori d_1, \dots, 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 obține 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	Numărul minim de monede folosite se află pe
10	0	1	2	3	4	5	1	2	3	4	1	2	2	

ultima linie, ultima coloană. Pentru a ușura construirea soluției se poate construi încă o matrice $Q[1..n, 0..S]$ caracterizată prin:

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

Algorithm 2 Construirea unui subșir strict crescător de lungime maximă

```
construire( $a[1..n]$ ,  $S[1..n]$ )  
// determinarea valorii și poziției maximului în  $S$   
 $imax \leftarrow 1$   
for  $i \leftarrow 2, n$  do  
    if  $S[imax] < S[i]$  then  
         $imax \leftarrow i$   
    end if  
end for  
 $m \leftarrow imax$   
 $k \leftarrow 1$   
 $x[1] \leftarrow a[m]$   
 $suma \leftarrow S[m]$   
while  $suma > a[m]$  do  
     $i \leftarrow m - 1$   
    while  $(a[i] \geq a[m])$  or  $(B[i] \neq S[m] - a[m])$  do  
         $i \leftarrow i - 1$   
    end while  
     $m \leftarrow i$   
     $k \leftarrow k + 1$  // se adaugă un nou element în subșir  
     $x[k] \leftarrow a[m]$   
end while  
return  $x[1..k]$ 
```

În cazul exemplului analizat matricea Q va avea următorul conținut:

	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 relației de recurență.* Matricile R și Q pot fi completate după cum este descris în Algoritmul 3 ($R[n, s]$ reprezintă numărul minim de monede necesare pentru a acoperi suma S).

(d) *Construirea soluției.* Considerând că tabloul a în care se colectează soluția și variabila k ce conține numărul de elemente ale lui a sunt variabile globale, soluția poate fi construită în manieră recursivă așa cum e descris în algoritmul 3.

Problema 3 *Traseu de scor maxim.* Fie $a[1..n, 1..n]$ o matrice de valori reale. Se definește un traseu în matrice ca o succesiune de elemente $a[i_1, 1], a[i_2, 2], \dots, a[i_n, n]$ cu proprietatea că $i_{k+1} \in \{i_k, i_k - 1, i_k + 1\}$ (evident dacă $i_k = 1$ atunci $i_{k+1} \in \{i_k, i_k + 1\}$ iar dacă $i_k = n$ atunci $i_{k+1} \in \{i_k, i_k - 1\}$). Un astfel de traseu vizitează câte un element de pe fiecare coloană trecând de la un element curent la unul "vecin" de pe coloană următoare (modulul diferenței dintre indicii de linie este cel mult 1). Să se determine un traseu având suma elementelor maximă.

Rezolvare. Fie $P(i, j)$ problema determinării unui traseu de sumă maximă care se termină în $a[i, j]$. Soluția optimă a acestei probleme conține soluția optimă a uneia dintre subproblemele $P(i - 1, j - 1)$, $P(i, j - 1)$ respectiv $P(i + 1, j - 1)$. Dacă notăm cu $V(i, j)$ suma asociată traseului optim care se termină în $a[i, j]$ atunci relația de recurență asociată este:

$$V(i, j) = \begin{cases} a(i, j) & j = 1 \\ \max\{V(i - 1, j - 1) + a(i, j), V(i, j - 1) + a(i, j), \\ V(i + 1, j - 1) + a(i, j)\} & j > 1 \end{cases}$$

Algorithm 3 Dezvoltarea relației de recurență în cazul problemei monedelor și construirea soluției

```
completare( $d[1..n], S$ )
for  $i \leftarrow 1, n$  do
     $R[i, 0] \leftarrow 0$ 
end for
for  $j \leftarrow 1, S$  do
     $R[1, j] \leftarrow j$ 
end for
for  $i \leftarrow 1, n$  do
    for  $j \leftarrow 1, S$  do
        if  $j < d[i]$  then
             $R[i, j] \leftarrow R[i - 1, j]; Q[i, j] \leftarrow 0$ 
        else
            if  $R[i - 1, j] < 1 + R[i, j - d[i]]$  then
                 $R[i, j] \leftarrow R[i - 1, j]; Q[i, j] \leftarrow 0$ 
            else
                 $R[i, j] \leftarrow R[i, j - d[i]] + 1; Q[i, j] \leftarrow 1$ 
            end if
        end if
    end for
end for
```

```
solutie( $i, j$ )
if  $j \neq 0$  then
    if  $Q[i, j] == 1$  then
         $k \leftarrow k + 1$ 
         $a[k] \leftarrow d[i]$ 
        solutie( $i, j - d[i]$ )
    else
        solutie( $i - 1, j$ )
    end if
end if
```

Să considerăm următorul exemplu:

$$A = \begin{bmatrix} 10 & 4 & 1 & 6 \\ 3 & -2 & 8 & 2 \\ 2 & 15 & -6 & 4 \\ 7 & -3 & 4 & 9 \end{bmatrix}$$

Printr-o abordare de tip greedy (în care se pornește de la cel mai mare element de pe prima coloană și la fiecare etapă se alege elementul cel mai mare din cele vecine aflate pe coloana următoare) s-ar obține traseul (10, 4, 8, 6) având suma 28. Aplicând relația de recurență de mai sus se obține matricea:

$$V = \begin{bmatrix} 10 & 14 & 15 & 36 \\ 3 & 8 & 30 & 32 \\ 2 & 22 & 16 & 34 \\ 7 & 4 & 26 & 35 \end{bmatrix}$$

ceea ce conduce la traseul: (7, 15, 8, 6) având suma 36.

O dată completată matricea V , soluția ($s = (i_1, i_2, \dots, i_n)$) poate fi construită începând de la ultimul element așa cum este descris în Algoritmul 4.

Problema 4 (*Problema turistului în Manhattan.*) Se consideră o grilă pătratică $n \times n$ (care poate fi interpretată ca harta străzilor din Manhattan). Fiecare muchie în cadrul grilei are asociată o valoare (ce poate fi interpretată ca fiind câștigul turistului care parcurge porțiunea respectivă de stradă). Se caută un traseu care pornește din nodul (1,1), se termină în nodul (n, n), și care are proprietatea că la fiecare etapă se poate trece din nodul (i, j) fie în nodul ($i, j + 1$) (deplasare la dreapta) fie în nodul ($i + 1, j$) (deplasare în jos). În plus valoarea asociată traseului trebuie să fie maximă.

Rezolvare. Presupunem că valorile asociate muchiilor grilei sunt stocate în două matrici: $C_D[1..n, 1..n - 1]$ ($C_D[i, j]$ conține valoarea asociată trecerii din nodul (i, j) în nodul ($i, j + 1$)) iar $C_J[1..n - 1, 1..n]$ conține valoarea asociată trecerii din nodul (i, j) în nodul ($i + 1, j$).

Dacă notăm cu $P(i, j)$ problema determinării unui traseu optim care se termină în (i, j) atunci soluția optimă a acestei probleme conține soluția optimă a uneia dintre subproblemele $P(i, j - 1)$ respectiv $P(i - 1, j)$. Notând cu $C(i, j)$ valoarea asociată soluției optime a problemei $P(i, j)$, relația de recurență asociată este:

Algorithm 4 Determinarea unui traseu de sumă maximă

```
solutie(integer  $a[1..n, 1..n]$ ,  $s[1..n]$ ,  $V[1..n, 1..n]$ )
 $s[n] \leftarrow \text{imax}(V[1..n, n])$  // indicele celui mai mare element din ultima
//coloană a matricii  $V$ 
for  $k \leftarrow n - 1, 1, -1$  do
  if  $V[s[k + 1], k + 1] = V[s[k + 1], k] + a[s[k + 1], k + 1]$  then
     $s[k] \leftarrow s[k + 1]$ 
  else
    if  $s[k + 1] > 1$  and  $V[s[k + 1], k + 1] = V[s[k + 1] - 1, k] + a[s[k + 1], k + 1]$  then
       $s[k] \leftarrow s[k + 1] - 1$ 
    else
      if  $s[k + 1] < n$  and  $V[s[k + 1], k + 1] = V[s[k + 1] + 1, k] + a[s[k + 1], k + 1]$  then
         $s[k] \leftarrow s[k + 1] + 1$ 
      end if
    end if
  end if
end if
end for
```

$$C(i, j) = \begin{cases} 0 & i = 1, j = 1 \\ C(i, j - 1) + C_D(i, j - 1) & i = 1, j > 1 \\ C(i - 1, j) + C_J(i - 1, j) & i > 1, j = 1 \\ \max\{C(i, j - 1) + C_D(i, j - 1), \\ C(i - 1, j) + C_J(i - 1, j)\} & i > 1, j > 1 \end{cases}$$

După completarea matricii C (folosind fie varianta clasică fie cea bazată pe tehnica memoizării) elementul $C(n, n)$ indică valoarea asociată soluției optime. Traseul propriu-zis poate fi determinat simplu dacă o dată cu construirea matricii C se construiește și o matrice P ale cărei elemente indică direcția asociată ultimului pas făcut pentru a ajunge în nodul respectiv: "dreapta" sau "jos":

$$P(i, j) = \begin{cases} \text{"dreapta"} & i = 1, j > 1 \\ \text{"jos"} & i > 1, j = 1 \\ \text{"dreapta"} & i > 1, j > 1, \\ & C(i, j - 1) + C_D(i, j - 1) > C(i - 1, j) + C_J(i - 1, j) \\ \text{"jos"} & i > 1, j > 1, \\ & C(i, j - 1) + C_D(i, j - 1) < C(i - 1, j) + C_J(i - 1, j) \end{cases}$$

În varianta recursivă algoritmul poate fi descris prin:

```
traseu(integer  $i, j$ )
if  $i > 1$  or  $j > 1$  then
  if  $P[i, j] = \text{"jos"}$  then
    traseu( $i - 1, j$ ); write "jos"
  else
    traseu( $i, j - 1$ ); write "dreapta"
  end if
end if
```

Pentru a obține traseul se apelează algoritmul **traseu**(n, n).

Problema 5 (*Problema colectării obiectelor*) Se consideră o grilă de dimensiune $m \times n$ ale cărei celule pot fi de unul dintre următoarele trei tipuri: (i) celulă accesibilă goală; (ii) celulă accesibilă care conține un obiect; (iii) celulă inaccesibilă. Se pune problema determinării unui traseu pentru un robot care pleacă din celula din colțul din stânga sus cu scopul de a ajunge în celula din colțul din dreapta jos trecând prin celule accesibile și colectând cât mai multe obiecte.

Indicație. Se consideră că informațiile privind starea fiecărei celule sunt stocate într-o matrice $B[1..m, 1..n]$ cu elemente din $\{-1, 0, 1\}$: $B[i, j] = -1$ dacă celula este inaccesibilă, $B[i, j] = 0$ dacă celula este accesibilă dar nu conține obiect, $B[i, j] = 1$ dacă celula este accesibilă și conține un obiect. Un exemplu de matrice este:

$$B = \begin{bmatrix} 0 & -1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ -1 & -1 & -1 & 0 & 1 & 0 \end{bmatrix}$$

Considerând că $F[i, j]$ conține numărul maxim de obiecte colectate de-a lungul unui traseu care se termină în celula (i, j) relația de recurență pentru calculul lui $F[i, j]$ este:

$$F[i, j] = \begin{cases} B[0, 0] & i = 0, j = 0 \\ F[i - 1, 0] + B[i, 0] & j = 0, i > 0, B[i, 0] \neq -1 \\ -\infty & j = 0, i > 0, B[i, 0] = -1 \\ F[0, j - 1] + B[0, j] & i = 0, j > 0, B[0, j] \neq -1 \\ -\infty & i = 0, j > 0, B[0, j] = -1 \\ \max\{F[i - 1, j], F[i, j - 1]\} + B[i, j] & \text{altfel} \end{cases}$$

Problema 6 (*Secționarea optimă a unei tije*) Se consideră o tijă de lungime n care se dorește a fi secționată în fragmente de lungimi întregi. Stiind că pentru fiecare lungime de fragment este un alt preț se pune problemă determinării unei modalități de secționare care să conducă la un cost cât mai mic.

De exemplu pentru o tijă de lungime 8 pentru care prețurile secționărilor sunt $p = [2, 3, 1, 4, 6, 5, 6, 5]$ (obținerea unui segment de lungime 1 are prețul 2, obținerea unui segment de lungimea 2 are prețul 3, pentru un segment de lungime 3 prețul este 1 etc) o secționare optimă ar conduce la două segmente de lungime 3 și un segment de lungime 2.

Indicație. Problema secționării este similară cu cea a înmulțirii optimale a unui șir de matrici în sensul că pentru a se obține o secționare optimă, fiecare dintre cele două fragmente obținute în urma unei tăieturi trebuie secționat optimal. Tija poate fi considerată ca un șir de elemente de lungime 1. Considerăm $C[i, j]$ costul secționării optimale a porțiunii de tijă delimitată de elementele i și j . Relația de recurență corespunzătoare calculului lui $C[i, j]$ este:

$$C[i, j] = \begin{cases} p[0] & i = j \\ \min(p[j - i], \min_{i \leq l < j} C[i, l] + C[l + 1, j]) & i < j \\ 0 & i > j \end{cases}$$

Dacă $i = j$ înseamnă că fragmentul are deja lungimea 1 și nu mai trebuie secționat, iar prețul lui corespunde lui $p[1]$. Dacă $i < j$ atunci se alege varianta de cost minim: fie nu se aplică tăietură (în acest caz prețul este corespunzător lungimii fragmentului, $p[j - i]$) sau se aplică la poziția l ($i \leq l < j$) care conduce la costul cel mai mic (în condițiile în care fiecare dintre fragmente este la rândul său fragmentat optimal). Pentru a determina mai ușor pozițiile de tăiere este util ca în paralel cu construirea tabloului C să se completeze și un tablou cu pozițiile de tăiere: $t[i, j]$ conține poziția tăieturii aplicate tijei delimitate de elementele i și j .

$$t[i, j] = \begin{cases} i & i = j \\ j & \text{dacă } i < j \text{ și } C[i, j] = p[j - i] \\ lmin & \text{dacă } i < j \text{ și } C[i, lmin] + C[lmin + 1, j] \leq C[i, l] + C[l + 1, j], l = \overline{i, j - 1} \\ 0 & i > j \end{cases}$$

Pentru exemplul de mai sus se obține următoarea matrice de costuri:

```
[[2 3 1 3 4 2 4 5]
 [0 2 3 1 3 4 2 4]
 [0 0 2 3 1 3 4 2]]
```

```
[0 0 0 2 3 1 3 4]
[0 0 0 0 2 3 1 3]
[0 0 0 0 0 2 3 1]
[0 0 0 0 0 0 2 3]
[0 0 0 0 0 0 0 2]]
```

și matrice de tăieturi (cu indicii elementelor din tijă pornind de la 0 - o valoare egală cu 0 înseamnă că se aplică tăietura după primul element, o valoare egală cu 1 înseamnă că se aplică tăietura după al doilea element etc):

```
[[0 1 2 0 4 2 0 2]
 [0 1 2 3 1 5 3 1]
 [0 0 2 3 4 2 6 4]
 [0 0 0 3 4 5 3 7]
 [0 0 0 0 4 5 6 4]
 [0 0 0 0 0 5 6 7]
 [0 0 0 0 0 0 6 7]
 [0 0 0 0 0 0 0 7]]
```

Problema 7 (*Problema închiderii tranzitive*) Considerăm o relație binară $R \subset \{1, \dots, n\} \times \{1, \dots, n\}$. Închiderea sa tranzitivă este o relație binară $R^* \subset \{1, \dots, n\} \times \{1, \dots, n\}$ având proprietatea: dacă pentru $i, j \in \{1, \dots, n\}$ există $i_1, \dots, i_m \in \{1, \dots, n\}$ cu proprietățile: $(i_1, i_2) \in R, (i_2, i_3) \in R, \dots, (i_{m-1}, i_m) \in R$ iar $i = i_1$ și $j = i_m$ atunci $(i, j) \in R^*$.

Pentru a construi R^* se consideră următorul set de relații binare $R^0 = R, R^1, \dots, R^n = R^*$, definite astfel: dacă pentru $i, j \in \{1, \dots, n\}$ există $i_1, \dots, i_m \in \{1, \dots, k\}$ cu proprietățile: $(i_1, i_2) \in R, (i_2, i_3) \in R, \dots, (i_{m-1}, i_m) \in R$ iar $i = i_1$ și $j = i_k$ atunci $(i, j) \in R^k$. Relațiile pot fi descrise prin recurențe astfel: $(i, j) \in R^k$ dacă $(i, j) \in R^{k-1}$ sau $(i, k) \in R^{k-1}$ și $(k, j) \in R^{k-1}$.

Indicație. Pentru a elabora algoritmul de construire a lui R^* considerăm relațiile binare pe $\{1, 2, \dots, n\}$ reprezentate prin matrici ale căror elemente sunt definite astfel:

$$r_{ij} = \begin{cases} 1 & \text{dacă } (i, j) \in R \\ 0 & \text{dacă } (i, j) \notin R \end{cases}$$

Relațiile de recurență pentru construirea matricilor asociate relațiilor binare R^0, R^1, \dots, R^n sunt:

$$r_{ij}^k = \begin{cases} 1 & \text{dacă } r_{ij}^{k-1} = 1 \text{ sau } (r_{ik}^{k-1} = 1 \text{ și } r_{kj}^{k-1} = 1) \\ 0 & \text{altfel} \end{cases} \quad k = \overline{1, n}$$

cu $r_{ij}^0 = r_{ij}$. Folosind aceste relații de recurență se poate construi matricea asociată relației binare R^n , utilizând doar două matrici auxiliare în modul descris în Algoritmul 5.

Algoritmul pentru determinarea închiderii tranzitive este cunoscut și sub numele de algoritmul lui Warshall.

Algorithm 5 Determinarea închiderii tranzitive a unei relații binare

```
închidere_tranzitivă( $R[1..n, 1..n]$ )  
 $R2[1..n, 1..n] \leftarrow R[1..n, 1..n]$   
for  $k \leftarrow 1, n$  do  
   $R1[1..n, 1..n] \leftarrow R2[1..n, 1..n]$   
  for  $i \leftarrow 1, n$  do  
    for  $j \leftarrow 1, n$  do  
      if ( $R1[i, j] = 1$ ) or ( $R1[i, k] = 1$  and  $R1[k, j] = 1$ ) then  
         $R2[i, j] \leftarrow 1$   
      else  
         $R2[i, j] \leftarrow 0$   
      end if  
    end for  
  end for  
end for  
return  $R2[1..n, 1..n]$ 
```

Probleme suplimentare.

1. *Problema selecției monedelor.* Se consideră o secvență de n monede cu valorile c_1, c_2, \dots, c_n (nu neapărat distincte) și se pune problema selecției unui subset de monede care să aibă valoarea totală maximă dar să nu conțină monede "învecinate" (dacă este selectată moneda i atunci nu poate fi selectată nici moneda $i - 1$ nici moneda $i + 1$). De exemplu pentru secvența de valori 5, 1, 2, 10, 6 și 2 subsetul de sumă maximă este constituit din monedele cu valorile 5, 10 și 2.

Indicație. Dacă $S(i)$ reprezintă suma maximă a unui subset extras din secvența c_1, c_2, \dots, c_i atunci relația de recurență pentru calculul lui $S(i)$ se construiește pe baza observației că subsetul selectat fie conține fie nu conține moneda i , ceea ce conduce la:

$$S(i) = \begin{cases} 0 & i = 0 \\ c_1 & i = 1 \\ \max\{S(i-1), S(i-2) + c_i\} & i > 2 \end{cases}$$

2. Se consideră un șir de valori reale (pozitive și negative). Să se determine subșir de elemente cu semne alternate (un element pozitiv este urmat de un element negativ iar un element negativ este urmat de un element pozitiv) pentru care suma valorilor modulelor este maximă.

Indicație. Se aplică ideea de la determinarea unui subșir strict crescător de sumă maximă doar că în loc ca elementele să fie ordonate crescător acestea trebuie să fie cu semne alternate (și se cumulează modulele elementelor).

3. Aplicați tehnica memoizării în algoritmul de determinare a distanței de editare.