Algoritmi şi structuri de date (I). Seminar 12: Aplicaţii ale programării dinamice (I).

- familiarizare cu clasele de probleme care pot fi rezolvate folosind tehnica programării dinamice - verificarea proprietăţii de substructură optimă
- construirea relaţiei de recurenţă
- dezvoltarea relaţiei de recurenţă
- construirea soluţiei

Problema 1 (*Problema determinării unui subşir strict crescător de sumă maximă.*) 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. De exemplu, pentru $a = (2, 5, 1, 3, 6, 8, 4)$ subşirul $(2, 5, 6, 8)$ are această proprietate.

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

a) *Caracterizarea soluţiei optime.* Fie $a_{j_1} < a_{j_2} < \ldots < a_{j_{k-1}} < a_{j_k}$ un subşir 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 $a_{j_1} < a_{j_2} < \ldots < a_{j_{k-1}}$ este subşir strict crescător al şirului parţial $(a_1, a_2, \ldots, a_p)$ care are cea mai mare sumă a elementelor dintre toate subşirurile care se termină în a_p (dacă ar exista un şubşir cu suma mai mare atunci prin adăugarea lui a_i s-ar obţine un subşir al lui $a[1..i]$ care ar avea suma mai mare decât a şubşirului considerat optim). Prin urmare problema are proprietatea de substructură optimă şi suma elementelor unui subşir crescător care se termină în a_i poate fi exprimată în funcţie de suma elementelor subşirurilor crescătoare care se termină cu elemente a_p ce satisfac $a_p < a_i$.

b) *Stabilirea unei relaţii de recurenţă între sumele elementele subşirurilor.* Fie $(S_i)_{i=\overline{1,n}}$ un tablou ce conţine pe poziţia i suma elementelor celui mai lung subşir strict crescător al şirului $(a_1, \ldots a_n)$ care are pe a_i ca ultim element (spunem că subşirul se termină în a_i). Ţinând cont de proprietăţile soluţiei optime se poate stabili o relaţie de recurenţă pentru S_i:

$$S_i = \begin{cases} a_1 & i = 1 \\ a_i + \max\{S_j | 1 \leq j < i \text{ şi } a_j < a_i\} & i > 1 \end{cases}$$

Dacă mulţimea $M_i = \{S_j | 1 \leq j < i \text{ şi } a_j < a_i\}$ este vidă atunci $\max M_i = 0$. De exemplu pentru şirul iniţial $a = (2, 5, 1, 3, 6, 8, 4)$ tabloul lungimilor subşirurilor 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 subşir strict crescător.

c) *Dezvoltarea relaţiei de recurenţă în manieră ascendentă.* Se completează un tablou cu elementele lui $(S_i)_{i=\overline{1,n}}$ aşa cum este descris în Algoritmul 1.

Luând în considerare doar comparaţia dintre două elemente ale şirului ($a[j] < a[i]$) 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 soluţii optime.* Se determină cea mai mare valoare din S. Aceasta indică suma maximă a elementelor dintr-un cel mai lung subşir strict crescător iar poziţia pe care se află indică elementul, $a[m]$, din şirul iniţial cu care se termină subşirul. Pornind de la acest element subşirul se construieşte în ordinea inversă a elementelor sale căutând la fiecare etapă un element din şir mai mic decât ultimul completat în subşir şi căruia îi corespunde în S o valoare care poate fi obţinută din cea curentă prin scăderea elementului corespunzător din şir. Algoritmul este descris în 2.

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

Algorithm 1 Dezvoltarea relaţiei de recurenţă pentru determinarea unui subşir strict crescător de sumă maximă

```
completare(a[1..n])
S[1] ← a[1]
for i ← 2, n do
  imax ← 1
  for j ← 1, i − 1 do
    if (a[j] < a[i]) and (S[imax] < S[j]) then
      imax ← j
    end if
  end for
  S[i] ← S[imax] + a[i]
end for
return S[1..n]
```

.

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

Problema 2 (*Problema monedelor.*) Se consideră monede de valori $d_n > d_{n-1} > \ldots > d_1 = 1$. 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 $s_k = d_i$ (se foloseşte monedă cu valoarea 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 $s_k \neq d_i$ (nu se foloseşte monedă cu valoarea 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

Numărul minim de monede folosite se află pe ultima linie, ultima coloană. Pentru a uşura construirea soluţiei se poate construi încă o matrice $Q[1..n, 0..S]$ caracterizată prin:

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

Algorithm 2 Construirea unui unui subşir strict crescător de sumă maximă

```
construire(a[1..n], S[1..n])
// determinarea valorii şi poziţiei maximului în S
imax ← 1
for i ← 2, n do
  if S[imax] < S[i] then
    imax ← i
  end if
end for
m ← imax
k ← 1
x[1] ← a[m]
suma ← S[m]
while suma > a[m] do
  i ← m − 1
  while (a[i] ≥ a[m]) or (B[i] ≠ S[m] − a[m]) do
    i ← i − 1
  end while
  m ← i
  k ← k + 1 // se adaugă un nou element în subşir
  x[k] ← a[m]
end while
return x[1..k]
```

.

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 este descris în Algoritmul 3 ($R[n, s]$ 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 aşa cum e descris în algoritmul 3.

Problema 3 *Traseu de scor maxim.* Fie $a[1..n, 1..n]$ o matrice de valori reale. Se defineşte un traseu în matrice ca o succesiune de elemente $a[i_1, 1], a[i_2, 2], \ldots, a[i_n, n]$ 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 diferenţei 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 $a[i, j]$. Soluţia optimă a acestei probleme conţine soluţia 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 relaţia de recurenţă asociată este:

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

Algorithm 3 Dezvoltarea relaţiei de recurenţă în cazul problemei monedelor şi construirea soluţiei

```
completare(d[1..n],S)
for i ← 1, n do
  R[i, 0] ← 0
end for
for j ← 1, S do
  R[1, j] ← j
end for
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
      end if
    end if
  end for
end for
```

```
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)
  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 porneşte de la cel mai mare element de pe prima coloană şi la fiecare etapă se alege elementul cel mai mare din cele vecine aflate pe coloana următoare) s-ar obţine traseul $(10, 4, 8, 6)$ având suma 28. Aplicând relaţia de recurenţă de mai sus se obţine 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, soluţia ($s = (i_1, i_2, \ldots, i_n)$) poate fi construită începând de la ultimul element aşa 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 câştigul turistului care parcurge porţiunea respectivă de stradă). Se caută un traseu care porneşte din nodul (1,1), se termină în nodul (n, n), şi 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: $C_D[1..n, 1..n-1]$ ($C_D[i, j]$ conţine valoarea asociată trecerii din nodul (i, j) în nodul $(i, j + 1)$) iar $C_J[1..n-1, 1..n]$ conţine 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 soluţia optimă a acestei probleme conţine soluţia optimă a uneia dintre subproblemele $P(i, j-1)$ respectiv $P(i-1, j)$. Notând cu $C(i, j)$ valoarea asociată soluţiei optime a problemei $P(i, j)$, relaţia de recurenţă 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] ← imax(V[1..n, n]) // indicele celui mai mare element din ultima
                        //coloană a matricii V
for k ← 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] ← 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] ← 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] ← s[k + 1] + 1
      end if
    end if
  end if
end for
```

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

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

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

In varianta recursivă algoritmul poate fi descris prin:

```
traseu(integer i, j)
if i > 1 or j > 1 then
  if P[i, j] = "jos" then
    traseu(i − 1, j); write "jos"
  else
    traseu(i, j − 1); write "dreapta"
  end if
end if
```

Pentru a obţine 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 conţine un obiect; (iii) celulă inaccesibilă. Se pune problema determinării unui traseu pentru un robot care pleacă din celula din colţul din stânga sus cu scopul de a ajunge în celula din colţul din dreapta jos putându-se deplasa înspre dreapta sau în jos prin celule accesibile şi colectând cât mai multe obiecte.

Indicaţie. Se consideră că informaţiile 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 conţine obiect, $B[i, j] = 1$ dacă celula este accesibilă şi conţine 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 $F[i, j]$ conţine numărul maxim de obiecte colectate de-a lungul unui traseu care se termină în celula (i, j) relaţia de recurenţă 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 (*Secţionarea optimală a unei tije*) Se consideră o tijă de lungime n care se doreşte a fi secţionată în fragmente de lungimi întregi. Stiind că pentru fiecare lungime de fragment este un alt preţ se pune problemă determinării unei modalităţi de secţionare care să conducă la un cost cât mai mic.

De exemplu pentru o tijă de lungime 8 pentru care preţurile secţionărilor sunt $p = [2, 3, 1, 4, 6, 5, 6, 5]$ (obţinerea unui segment de lungime 1 are preţul 2, obţinerea unui segment de lungimea 2 are preţul 3, pentru un segment de lungime 3 preţul este 1 etc) o secţionare optimală ar conduce la două segmente de lungime 3 şi un segment de lungime 2.

Indicaţie. Problema secţionării este similară cu cea a înmulţirii optimale a unui şir de matrici în sensul că pentru a se obţine o secţionare optimală, fiecare dintre cele două fragmente obţinute în urma unei tăieturi trebuie secţionat optimal. Tija poate fi considerată ca un şir de elemente de lungime 1. Considerăm $C[i, j]$ costul secţionării optimale a porţiunii de tijă delimitată de elementele i şi j. Relaţia de recurenţă 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 şi nu mai trebuie secţionat, iar preţul lui corespunde lui $p[0]$. Dacă $i < j$ atunci se alege varianta de cost minim: fie nu se aplică tăietură (în acest caz preţul este corespunzător lungimii fragmentului, $p[j - i]$) sau se aplică la poziţia l ($i \leq l < j$) care conduce la costul cel mai mic (în condiţiile în care fiecare dintre fragmente este la rândul său fragmentat optimal). Pentru a determina mai uşor poziţiile de tăiere este util ca în paralel cu construirea tabloului C să se completeze şi un tablou cu poziţiile de tăiere: $t[i, j]$ conţine poziţia tăieturii aplicate tijei delimitate de elementele i şi j.

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

Pentru exemplul de mai sus se obţine 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]]
```

şi matrice de tăieturi (cu indicii elementelor din tijă pornind de la 0 - o valoare egală cu 0 înseamnă că se aplică tăietură după primul element, o valoare egală cu 1 înseamnă că se aplică tăietură 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 relaţie binară $R \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$. Închiderea sa tranzitivă este o relaţie binară $R^* \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$ având proprietatea: dacă pentru $i, j \in \{1, \ldots, n\}$ există $i_1, \ldots, i_m \in \{1, \ldots, n\}$ cu proprietăţile: $(i_1, i_2) \in R$, $(i_2, i_3) \in R$, ... ,$(i_{m-1}, i_m) \in R$ iar $i = i_1$ şi $j = i_m$ atunci $(i, j) \in R^*$.

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

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

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

Relaţiile de recurenţă pentru construirea matricilor asociate relaţiilor binare $R^0, R^1, \ldots, R^n$ sunt:

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

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

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

Algorithm 5 Determinarea închiderii tranzitive a unei relaţii binare

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

Probleme suplimentare.

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

   Indicaţie. Dacă $S(i)$ reprezintă suma maximă a unui subset extras din secvenţa c_1,c_2, ..., c_i atunci relaţia de recurenţă pentru calculul lui $S(i)$ se construieşte pe baza observaţiei că subsetul selectat fie conţine fie nu conţine 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 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 element pozitiv) pentru care suma valorilor modulelor este maximă.

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

3. Aplicaţi tehnica memoizării în algoritmul de determinare a distanţei de editare.