Algoritmi şi structuri de date (I). Seminar 12-13: Aplicaţii ale programării dinamice.

Problema 1 (*Problema determinării celui mai lung subşir strict crescător*) Se consideră un şir de valori reale $a_1, a_2, \ldots, a_n$ şi se caută un subşir $a_{j_1}, a_{j_2}, \ldots, a_{j_k}$ cu $1 \leq j_1 < j_2 < \ldots < j_k \leq n$, $a_{j_1} < a_{j_2} < \ldots < a_{j_k}$ şi astfel încât k să fie cât mai mare. Un astfel de subşir nu este neapărat unic. De exemplu pentru şirul $(2, 5, 1, 3, 6, 8, 2, 10)$ există două subşiruri strict crescătoare de lungime 5: $(2, 3, 6, 8, 10)$ şi $(1, 3, 6, 8, 10)$. Pe de altă parte nu toate subşirurile de aceeaşi lungime se termină cu acelaşi element. De exemplu în şirul $(2, 1, 4, 3)$ există patru subşiruri crescătoare de lungime maximă (2): $(2, 4)$, $(2, 3)$, $(1, 4)$, $(1, 3)$.

Rezolvare.

a) *Caracterizarea soluţiei optime.* Fie $a_{j_1} < a_{j_2} < \ldots < a_{j_{k-1}} < a_{j_k}$ un subşir 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 cel mai lung dintre subşirurile crescătoare ale şirului parţial $(a_1, a_2, \ldots, a_p)$ care au proprietatea că ultimul element este chiar a_p. Prin urmare problema are proprietatea de substructură optimă şi lungimea unui subşir crescător care se termină în a_i poate fi exprimată în funcţie de lungimea subşirurilor crescătoare care se termină cu elemente a_p ce satisfac $a_p < a_i$.

b) *Stabilirea unei relaţii de recurenţă între lungimile subşirurilor.* Fie $(B_i)_{i=\overline{1,n}}$ un tablou ce conţine pe poziţia i numărul de elemente al 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 B_i:

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

Dacă mulţimea $M_i = \{B_j | 1 \leq j < i \text{ şi } a_j < a_i\}$ 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: $B = (1, 2, 1, 2, 3, 4, 3)$. Cea mai mare valoare din B indică numărul de elemente din cel mai lung subşir crescător.

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

Algorithm 1 Dezvoltarea relaţiei de recurenţă pentru determinarea celui mai lung subşir crescător

```
completare(a[1..n])
B[1] ← 1
for i ← 2, n do
    max ← 0
    for j ← 1, i − 1 do
        if (a[j] < a[i]) and (max < B[j]) then
            max ← B[j]
        end if
    end for
    B[i] ← max + 1
end for
return B[1..n]
```

.

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 B este de complexitate $\Theta(n^2)$.

d) *Construirea unei soluţii optime.* Se determină cea mai mare valoare din B. Aceasta indică numărul de elemente ale celui mai lung subşir 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

Algorithm 2 Construirea unui cel mai lung subşir crescător

```
construire(a[1..n], B[1..n])
// determinarea valorii şi poziţiei maximului în B
imax ← 1
for i ← 2, n do
  if B[imax] < B[i] then
    imax ← i
  end if
end for
m ← imax
k ← B[m]
x[k] ← a[m]
while B[m] > 1 do
  i ← m − 1
  while (a[i] ≥ a[m]) or (B[i] ≠ B[m] − 1) 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]
```

.

a elementelor sale căutând la fiecare etapă un element din şir mai mic decât ultimul completat în subşir şi căruia îi corespunde în B o valoare cu 1 mai mică decât cea curentă. Algoritmul este descris în 2.

Dacă după completarea elementului $a[m]$ în subşir există două subşiruri din $a[1..m-1]$ de lungime maximă egală cu $B[m]-1$: 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. Aplicând algoritmul de construire şirului $(2, 5, 1, 3, 6, 8, 4)$ se va porni cu $m = 6$ ($B[m] = 4$) ultimul element din subşir fiind astfel 8. Cum $B[5] = 3$ şi $6 < 8$ penultimul element va fi 6. Continuând procesul se selectează în continuare elementele 3 şi 1 obţinându-se subşirul crescător $(1, 3, 6, 8)$ de lungime 4.

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 B 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 $s_k = d_i$ 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 $s_k \neq d_i$ atunci $(s_1, s_2, \ldots, s_{k-1})$ trebuie să fie soluţie optimă a subproblemei $P(i - 1, j)$.

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
```

(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 încă o matrice $Q[1..n, 0..S]$ caracterizată prin:

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

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 *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:

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

Problema 4 *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:

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

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 5 (*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.

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
```

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:

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

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, j − 1); write "jos"
  else
    traseu(i − 1, j); write "dreapta"
  end if
end if
```

Pentru a obţine traseul se apelează algoritmul `traseu`(n, n).