

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

a) *Caracterizarea soluției optime.* Fie  $a_{j_1} < a_{j_2} < \dots < a_{j_{k-1}} < a_{j_k}$  un subșir 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 cel mai lung dintre subșirurile crescătoare ale șirului parțial  $(a_1, a_2, \dots, a_p)$  care au proprietatea că ultimul element este chiar  $a_p$ . Prin urmare problema are proprietatea de substructură optimă și lungimea unui subșir crescător care se termină în  $a_i$  poate fi exprimată în funcție de lungimea subșirurilor crescătoare care se termină cu elemente  $a_p$  ce satisfac  $a_p < a_i$ .

b) *Stabilirea unei relații de recurență între lungimile subșirurilor.* Fie  $(B_i)_{i=1, \dots, n}$  un tablou ce conține pe poziția  $i$  numărul de elemente al 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  $B_i$ :

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

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

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

---

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

---

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

---

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

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

---

**Algorithm 2** Construirea unui cel mai lung subșir crescător

---

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

---

a elementelor sale căutând la fiecare etapă un element din șir mai mic decât ultimul completat în subșir și 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 subșir există două subșiruri din  $a[1..m-1]$  de lungime maximă egală cu  $B[m] - 1$ : 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  $q$ . Aplicând algoritmul de construire șirului  $(2, 5, 1, 3, 6, 8, 4)$  se va porni cu  $m = 6$  ( $B[m] = 4$ ) ultimul element din subșir fiind astfel 8. Cum  $B[5] = 3$  și  $6 < 8$  penultimul element va fi 6. Continuând procesul se selectează în continuare elementele 3 și 1 obținându-se subșirul crescător  $(1, 3, 6, 8)$  de lungime 4.

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

---

**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
```

(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
10	0	1	2	3	4	5	1	2	3	4	1	2	2

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

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

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** *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:

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

**Problema 4** *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:

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

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 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 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.

---

**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
```

---

*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:

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

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

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