Algoritmi şi structuri de date (I). Seminar 14: Aplicaţii ale tehnicii căutării cu revenire.

Problema 1 (*Problema comis voiajorului (TSP-Traveling Salesman Problem)).* Se consideră un set de n oraşe şi un comis voiajor care trebuie să viziteze toate cele n oraşe pornind din primul oraş, trecând o singură dată prin fiecare oraş şi întorcându-se în primul oraş. Se consideră cunoscută matricea $D[1..n, 1..n]$ în care elementul $D[i, j]$ este 0 dacă nu există drum direct între oraşul i şi oraşul j respectiv lungimea drumului direct dintre cele două oraşe. (i) Să se genereze toate circuitele pe care le poate parcurge comis voiajorul pornind din oraşul 1; (ii) Să se determine cel mai scurt circuit.

Rezolvare. În ambele variante soluţia poate fi reprezentată printr-un şir $(s_1, \ldots, s_n)$ unde $s_i \in \{1, \ldots, n\}$ reprezintă indicele oraşului vizitat la etapa i. Pentru a obţine circuitul trebuie doar adăugat oraşul de pornire (de exemplu, 1). Restricţiile ce trebuie satisfăcute sunt: $s_i \neq s_j$ pentru orice $i \neq j$ (nu se vizitează de două ori acelaşi oraş) şi $D(s_{i-1}, s_i) > 0$ pentru $i = \overline{2, n}$ (există drum direct între oraşele vizitate la etape succesive).

Condiţiile de continuare deduse din restricţiile de mai sus sunt: $s_k \neq s_i$ pentru $i = \overline{1, k-1}$ şi $D(s_{k-1}, s_k) > 0$. În cazul căutării celui mai scurt traseu numărul traseelor generate poate fi redus dacă se completează condiţiile de continuare cu $\sum_{i=2}^{k} D(s_{i-1}, s_i) < Lmin$ unde $Lmin$ este lungimea celui mai scurt traseu generat până la momentul curent. Variabila $Lmin$ se iniţializează înainte de apelul algoritmului de generare cu o valoare suficient de mare ($Lmin = \infty$). In ambele variante ale problemei se consideră că s-a obţinut o soluţie când au fost completate toate cele n componente. Pentru a obţine lungimea circuitului se adaugă la suma $\sum_{i=2}^{n} D(s_{i-1}, s_i)$ distanţa dintre oraşul s_n şi primul oraş (stocată în $D(s_n, 1)$).

În prima variantă, algoritmul este descris în 1.

Algorithm 1 Problema comis voiajorului. Generarea tuturor circuitelor (stânga). Generarea unui circuit de lungime minimă (dreapta)

```
TSP(integer k)
if k − 1 = n then
  if D[s[n], s[1]] > 0 then
    write s[1..n]
  end if
else
  for i ← 2, n do
    s[k] ← i
    if validare(s[1..k]) =true
    then
      TSP(k + 1)
    end if
  end for
end if
```

```
1:  optTSP(integer k)
2:  if (k − 1 = n) then
3:    if D[s[n], s[1]] > 0 AND L+D[s[n], s[1]] <
      Lmin then
4:      Lmin ← L + D[s[n], s[1]]
5:      smin[1..n] ← s[1..n]
6:    end if
7:  else
8:    for i ← 2, n do
9:      s[k] ← i
10:     if validare(s[1..k]) =true then
11:       L ← L + D[s[k − 1], s[k]]
12:       optTSP(k + 1)
13:       L ← L − D[s[k − 1], s[k]]
14:     end if
15:   end for
16: end if
```

Înainte de apelul lui `TSP`, $s[1]$ se setează pe 1 (se porneşte întotdeauna din primul oraş) iar algoritmul se apelează pentru $k = 2$. În algoritmul `validare` se verifică faptul că $s[k] \neq s[i]$ pentru $i = \overline{1, k-1}$ şi faptul că $D[s[k-1], s[k]] > 0$. În cazul în care una dintre aceste condiţii este încălcată se returnează **false**.

În a doua variantă se folosesc variabilele globale $smin[1..n]$ şi $Lmin$ pentru a stoca cel mai scurt traseu generat până la momentul curent, respectiv lungimea lui. $Lmin$ se iniţializează cu o valoare suficient de mare. În plus se foloseşte variabila globală L care conţine la fiecare etapă lungimea traseului dintre primul şi ultimul oraş vizitat. Înainte de apel L se iniţializează cu 0. Algoritmul este descris în 1. Algoritmul `optTSP` poate fi transformat pe baza ideii de la tehnica "ramifică şi mărgineşte" calculând pentru fiecare soluţie

parţială o margine inferioară a lungimii traseului şi efectuând apelul recursiv de la linia 12 doar dacă această limită este mai mică decât $Lmin$ (iniţializată înaintea apelului `optTSP`(1) cu o valoare suficient de mare - de exemplu $n \max_{i,j} D_{ij}$). O variantă de calcul a limitei inferioare pentru lungimea traseului în cazul soluţiei parţiale $(s_1, \ldots, s_k)$ este:

$$L_B = \sum_{i=1}^{k-1} D_{s_i s_{i+1}} + (n-k) \sum_{i=k+1}^{n} d_i + \min_{i=\overline{k+1,n}} D_{i1}$$

unde d_i este media aritmetică a distanţelor dintre oraşul i şi cele mai apropiate alte două oraşe care nu au fost vizitate încă.

Problema 2 (*Turul calului.*) Se consideră un cal plasat pe o poziţie (i_0, j_0) pe o tablă de şah. Se pune problema determinării unei secvenţe de poziţii prin care trebuie să treacă calul pentru a vizita o singură dată fiecare poziţie.

Indicaţie. Soluţia poate fi reprezentată printr-un tablou conţinând 64 de perechi distincte de indici (i, j) $(i, j \in \{1, \ldots, 8\})$. Două perechi consecutive ale soluţiei, (i, j) şi (i', j') trebuie să reprezinte poziţii între care calul se poate deplasa printr-o singură mutare, adică: $|i - i'| = 1$ şi $|j - j'| = 2$ sau $|i - i'| = 2$ şi $|j - j'| = 1$.

Problema 3 *Pătrat magic.* Un pătrat magic de dimensiune n este o matrice cu n linii şi n coloane care conţine valorile din mulţimea $\{1, 2, \ldots, n^2\}$ astfel încât suma elementelor de pe fiecare linie şi de pe fiecare coloană este aceeaşi.
(a) Care este suma elementelor de pe o linie (coloană) dintr-un pătrat magic de dimensiune n.
(b) Folosind tehnica căutării sistematice, generaţi toate pătratele magice de dimensiune n.

Indicaţie. (a) Suma elementelor de pe fiecare linie/coloană este suma elementelor din $\{1, 2, \ldots, n^2\}$ împărţită la numărul de linii (n). (b) O soluţie este o permutare a mulţimii $\{1, 2, \ldots, n^2\}$ care satisface restricţiile referitoare la sumele elementelor de pe linii/coloane. O variantă de rezolvare constă în generarea permutărilor de ordin n^2 şi selecţia celor care satisfac restricţiile.

Problema 4 (*Partiţionarea unui număr.*) Pentru un număr natural C să se determine toate modalităţile distincte de a descompune C ca sumă de numere naturale nenule (două descompuneri sunt considerate distincte dacă nu conţin aceiaşi termeni).

Indicaţie. O soluţie poate fi reprezentată ca un vector de forma $(s_1, s_2, \ldots s_k)$ unde $s_i \in \{1, 2, \ldots, C\}$, $s_i \leq s_{i+1}$, $i = \overline{1, k-1}$ şi $s_1 + s_2 + \ldots + s_k = C$. Condiţia de continuare este $s_k \geq s_{k-1}$ şi $s_1 + s_2 + \ldots + s_k \leq C$ iar condiţia de oprire (corespunzătoare apelului destinat completării componentei k) este $s_1 + s_2 + \ldots + s_{k-1} = C$.

Problema 5 (*Problema labirintului.*) Se consideră o grilă pătratică $n \times n$ în care anumite celule sunt libere iar altele sunt ocupate. Celulele libere ale grilei pot fi vizitate prin deplasare din poziţia curentă în oricare dintre poziţiile libere vecine (stânga, dreapta, sus, jos). Presupunând că celulele $(1, 1)$ şi (n, n) sunt libere să se genereze toate traseele care permit trecerea doar prin celule libere de la $(1, 1)$ la (n, n). Un exemplu de labirint si de traseu care uneşte cele două celule este ilustrat în Figura 1.

Rezolvare. Considerăm că informaţia privind starea unei celule este stocată în matricea $M[1..n, 1..n]$ în care $M[i, j]$ este 0 dacă celula e liberă şi 1 în caz contrar. O soluţie a problemei este un şir de perechi de coordonate identificând celulele libere prin care se trece: $s = (s_1, \ldots, s_m)$ cu $s_l = (i_l, j_l)$. Restricţiile ce trebuie satisfăcute de către soluţie sunt: elementele lui s sunt perechi distincte şi se trece doar prin celule libere $(M[s[l].i, s[l].j] = 0)$. Când se ajunge în celula (n, n) se consideră că s-a obţinut o soluţie. Dintr-o celulă de coordonate (i, j) se poate trece într-una dintre cele patru celule vecine având coordonatele: $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ şi $(i, j+1)$ (ţinându-se cont de limitările existente pentru celulele de pe frontieră). Algoritmul de validare returnează **false** dacă cel puţin una dintre următoarele condiţii este încălcată:

1. $1 \leq s[k].i \leq n$, $1 \leq s[k].j \leq n$

2. $M[s[k].i, s[k].j] = 0$

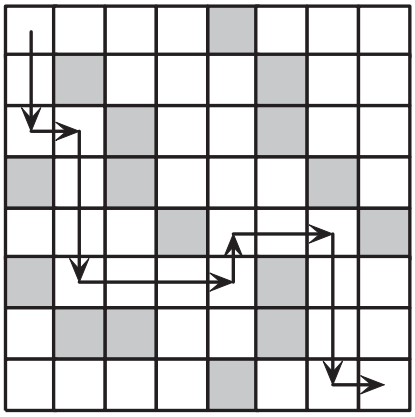

Figure 1: Exemplu de labirint şi de traseu care uneşte colţul din stânga sus cu colţul din dreapta jos

Algorithm 2 Determinarea unui traseu prin labirint

```
labirint(integer k)
if s[k-1] = (n, n) then
  write s[1..k-1]
else
  s[k] ← (s[k-1].i - 1, s[k-1].j)
  if validare(s[1..k]) then
    labirint (k+1)
  end if
  s[k] ← (s[k-1].i + 1, s[k-1].j)
  if validare(s[1..k]) then
    labirint (k+1)
  end if
  s[k] ← (s[k-1].i, s[k-1].j - 1)
  if validare(s[1..k]) then
    labirint (k+1)
  end if
  s[k] ← (s[k-1].i, s[k-1].j + 1)
  if validare(s[1..k]) then
    labirint (k+1)
  end if
end if
```

3. $s[k] \neq s[l]$, $l = \overline{1, k-1}$

Algoritmul care generează traseele este descris în Alg. 2.

Problema 6 Se consideră un text din care au fost eliminate spaţiile (de exemplu "acestaeunexemplu") şi un dicţionar (de exemplu {ac, acest, acesta, e, este, un, sta, exemplu, ta}). Să se determine toate propoziţiile constituite din cuvinte prezente în dicţionar care pot fi construite prin inserare de spaţii în şirul iniţial de caractere.