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

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

Rezolvare. În ambele variante soluția poate fi reprezentată printr-un șir (s_1, \dots, s_n) unde $s_i \in \{1, \dots, n\}$ reprezintă indicele orașului vizitat la etapa i . Pentru a obține circuitul trebuie doar adăugat orașul de pornire (de exemplu, 1). Restricțiile ce trebuie satisfăcute sunt: $s_i \neq s_j$ pentru orice $i \neq j$ (nu se vizitează de două ori același oraș) și $D(s_{i-1}, s_i) > 0$ pentru $i = 2, \bar{n}$ (există drum direct între orașele vizitate la etape succesive).

Condițiile de continuare deduse din restricțiile de mai sus sunt: $s_k \neq s_i$ pentru $i = \overline{1, k-1}$ și $D(s_{k-1}, s_k) > 0$. În cazul căutării celui mai scurt traseu numărul traseelor generate poate fi redus dacă se completează condițiile 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 inițializează înainte de apelul algoritmului de generare cu o valoare suficient de mare ($Lmin = \infty$). În ambele variante ale problemei se consideră că s-a obținut o soluție când au fost completate toate cele n componente. Pentru a obține lungimea circuitului se adaugă la suma $\sum_{i=2}^n D(s_{i-1}, s_i)$ distanța dintre orașul s_n și primul oraș (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)	1: optTSP(integer k)
if $k - 1 = n$ then	2: if $(k - 1 = n)$ then
if $D[s[n], s[1]] > 0$ then	3: if $D[s[n], s[1]] > 0$ AND $L + D[s[n], s[1]] <$
write $s[1..n]$	$Lmin$ then
end if	4: $Lmin \leftarrow L + D[s[n], s[1]]$
else	5: $smin[1..n] \leftarrow s[1..n]$
for $i \leftarrow 2, n$ do	6: end if
$s[k] \leftarrow i$	7: else
if validare($s[1..k]$) = true	8: for $i \leftarrow 2, n$ do
TSP($k + 1$)	9: $s[k] \leftarrow i$
end if	10: if validare($s[1..k]$) = true then
end for	11: $L \leftarrow L + D[s[k-1], s[k]]$
end if	12: optTSP($k + 1$)
	13: $L \leftarrow L - D[s[k-1], s[k]]$
	14: end if
	15: end for
	16: end if

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

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

parțială o margine inferioară a lungimii traseului și efectuând apelul recursiv de la linia 12 doar dacă această limită este mai mică decât $Lmin$ (inițializată 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 soluției parțiale (s_1, \dots, s_k) este:

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

unde d_i este media aritmetică a distanțelor dintre orașul i și cele mai apropiate alte două orașe care nu au fost vizitate încă.

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

Indicație. Soluția poate fi reprezentată printr-un tablou conținând 64 de perechi distincte de indici (i, j) ($i, j \in \{1, \dots, 8\}$). Două perechi consecutive ale soluției, (i, j) și (i', j') trebuie să reprezinte poziții între care calul se poate deplasa printr-o singură mutare, adică: $|i - i'| = 1$ și $|j - j'| = 2$ sau $|i - i'| = 2$ și $|j - j'| = 1$.

Problema 3 *Pătrat magic.* Un pătrat magic de dimensiune n este o matrice cu n linii și n coloane care conține valorile din mulțimea $\{1, 2, \dots, n^2\}$ astfel încât suma elementelor de pe fiecare linie și de pe fiecare coloană este aceeași.

- (a) Care este suma elementelor de pe o linie (coloană) dintr-un pătrat magic de dimensiune n .
- (b) Folosind tehnica căutării sistematice, generați toate pătratele magice de dimensiune n .

Indicație. (a) Suma elementelor de pe fiecare linie/coloană este suma elementelor din $\{1, 2, \dots, n^2\}$ împărțită la numărul de linii (n). (b) O soluție este o permutare a mulțimii $\{1, 2, \dots, n^2\}$ care satisface restricțiile referitoare la sumele elementelor de pe linii/coloane. O variantă de rezolvare constă în generarea permutărilor de ordin n^2 și selecția celor care satisfac restricțiile.

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

Indicație. O soluție poate fi reprezentată ca un vector de forma (s_1, s_2, \dots, s_k) unde $s_i \in \{1, 2, \dots, C\}$, $s_i \leq s_{i+1}$, $i = 1, k-1$ și $s_1 + s_2 + \dots + s_k = C$. Condiția de continuare este $s_k \geq s_{k-1}$ și $s_1 + s_2 + \dots + s_k \leq C$ iar condiția de oprire (corespunzătoare apelului destinat completării componente k) este $s_1 + s_2 + \dots + s_{k-1} = C$.

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 poziția curentă în oricare dintre pozițiile libere vecine (stânga, dreapta, sus, jos). Presupunând că celulele $(1, 1)$ și (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 și de traseu care unește cele două celule este ilustrat în Figura 1.

Rezolvare. Considerăm că informația privind starea unei celule este stocată în matricea $M[1..n, 1..n]$ în care $M[i, j]$ este 0 dacă celula e liberă și 1 în caz contrar. O soluție a problemei este un șir de perechi de coordonate identificând celulele libere prin care se trece: $s = (s_1, \dots, s_m)$ cu $s_l = (i_l, j_l)$. Restricțiile ce trebuie satisfăcute de către soluție sunt: elementele lui s sunt perechi distincte și 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 obținut o soluție. Dintr-o celulă de coordonate (i, j) se poate trece într-una dintre cele patru celule vecine având coordonatele: $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ și $(i, j+1)$ (ținându-se cont de limitările existente pentru celulele de pe frontieră). Algoritmul de validare returnează **false** dacă cel puțin una dintre următoarele condiții este încălcată:

1. $1 \leq s[k].i \leq n$, $1 \leq s[k].j \leq n$
2. $M[s[k].i, s[k].j] = 0$

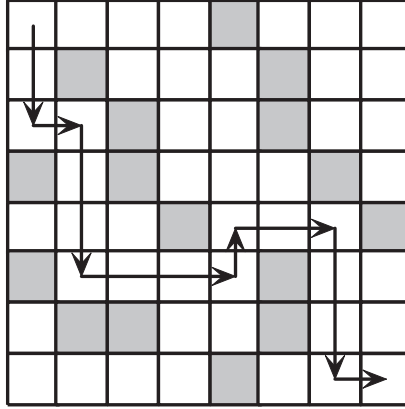


Figure 1: Exemplu de labirint și de traseu care unește colțul din stânga sus cu colțul din dreapta jos

Algorithm 2 Determinarea unui traseu prin labirint

```

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

```

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

Algoritmul care generează traseele este descris în Alg. 2.