Backtracking

Backtracking este un principiu fundamental de elaborarea a algoritmilor pentru probleme de optimizare, sau de găsire a unor soluții la o problemă dată. Algoritmii de tip backtracking se bazează pe o tehnică specială de explorare în grafurile orientate implicite. Aceste grafuri sunt de obicei arbori, sau cel puțin nu conțin cicluri.

Pentru exemplificare, vom considera o problemă clasică: cea a plasării a opt dame pe tabla de șah, astfel ca niciuna să nu intre în zona controlată de cealaltă. O metodă simplă este cea de a încerca toate combinațiile de plasare a celor 8 regine, verificând de fiecare dată dacă nu s-a obținut o soluție. Există în total $\binom{64}{8} = 4{,}426{,}165{,}368$ combinații posibile, clar această abordare nu este practică.

O primă îmbunătățire ar fi să nu plasăm mai mult de o regină pe fiecare linie deoarece altfel clar am avea cel puțin două regine care ”se atacă”. Această restricție reduce reprezentarea unei configurații pe tabla de șah la un simplu vector, $posibil[1..8]$ cu regina de pe linia i, $1 \leq i \leq 8$ aflându-se pe coloana $poisbil[i]$ adică pe tabla de șah în poziția $(i, posibil[i])$. De exemplu, vectorul $(3,1,6,2,8,6,4,7)$ nu reprezintă o soluție pentru că reginele de pe liniile trei și șase sunt pe aceeași coloană. În plus există două perechi de regine situate pe aceeași diagonală.

Algoritmul care găsește o soluție a problemei este cel de mai jos:

$procedure\ regine1$

1. $\boldsymbol{for}\ i_1 \leftarrow 1\ \boldsymbol{to}\ 8\ \boldsymbol{do}$
2. $\quad \boldsymbol{for}\ i_2 \leftarrow 1\ \boldsymbol{to}\ 8\ \boldsymbol{do}$
3. $\quad\quad \ldots\ \boldsymbol{for}\ i_8 \leftarrow 1\ \boldsymbol{to}\ 8\ \boldsymbol{do}$
4. $\quad\quad\quad posibil\ \leftarrow (i_1, i_2, \ldots, i_8)$
5. $\quad\quad\quad \boldsymbol{if}\ solutie(poisbil)\ \boldsymbol{then}\ \boldsymbol{write}\ posibil$
6. $\quad\quad\quad\quad\quad\quad\quad\quad \boldsymbol{stop}$
7. $\boldsymbol{write}\ "nu\ exist\check{a}\ solutie"$

De această dată numărul combinațiilor s-a redus la $8^8 = 16{,}777{,}216$, algoritmul oprindu-se de fapt după ce inspectează 1,299,852 și găsește prima soluție.

În continuare vom îmbunătăți acest algoritm. Dacă introducem și restricția ca două regine să nu se afle pe aceeași diagonală, o configurație pe tabla de șah se poate reprezenta ca o *permutare* a primilor opt întregi. Algoritmul devine

$procedure\ regine2$

1. $posibil \leftarrow$ permutarea inițială
2. $\boldsymbol{while}\ posibil\ \neq$ permutarea finală $\boldsymbol{and\ not}\ soluție(posibil)\ \boldsymbol{do}$
3. $\quad posibil\ \leftarrow$ următoarea permutare
4. $\boldsymbol{if}\ solutie(posibil)\ \boldsymbol{then}\ write\ posibil$
5. $\quad\quad\quad\quad\quad\quad \boldsymbol{else\ write}$ *"nu există soluție"*

Sunt mai multe posibilități de a genera sistematic toate permutările primilor n întregi. De exemplu, putem pune în fiecare din cele n elemente, pe rând, în prima poziție, generând de fiecare dată recursiv toate permutările celor $n-1$ elemente rămase. Iată mai jos algoritmul de generare a permutărilor a n numere:

$procedure\ perm(i)$

1. $\boldsymbol{if}\ i = n\ \boldsymbol{then}\ utilizează(T)\quad unde\ T\ este\ o\ nouă\ permutare$
2. $\quad\quad\quad \boldsymbol{else\ for}\ j \leftarrow i\ \boldsymbol{to}\ n\ \boldsymbol{do}\ interschimba\ T[i]\ şi\ T[j]$
3. $\quad\quad\quad\quad\quad\quad\quad\quad perm(i+1)$
4. $\quad\quad\quad\quad\quad\quad\quad\quad interschimba\ T[i]\ şi\ T[j]$

În acest algoritm de generare a permutărilor, $T[1..n]$ este un tablou global inițializat cu $[1,2,\dots,n]$, iar primul apel al procedurii este $perm(1)$. Dacă $utilizează(T)$ necesită un timp constant, atunci $perm(1)$ este în timp $O(n!)$. Această abordare va reduce numărul de configurații posibile la $8! = 40{,}320$. Dacă se folosește algoritmul $perm$ atunci până la prima soluție sunt generate 2830 de permutări. Verificarea faptului că o configurație este o soluție este simplă: trebuie verificat să nu fie două regine pe aceeași diagonală.

Chiar și cu aceste ajustări, nu am reușit să eliminăm o deficiență comună algoritmilor de mai sus și anume instrucțiunea $\boldsymbol{if}\ solutie(posibil)$ se face plasând toate reginele pe tabla de șah. Aceasta înseamnă un număr de 8 instrucțiuni în cazul unei table de 8X8.

Iată o soluție pentru a elimina acest impediment. Pentru început reformulăm problema de căutare a reginelor ca o problemă de căutare în arbore. Fie vectorul $P[1..k]$ de întregi între 1 și 8 ce conține plasamentul unei regine. Spunem că acest vector este k promițător pentru $0 \leq k \leq 8$, dacă zonele controlate de cele k regine plasate în pozițiile $(1, P[1])$, $(2, P[2])$, …, $(k, P[k])$ sunt disjuncte. Din punct de vedere matematic, aceasta înseamnă că:

$$P[i] - P[j] \notin \{i - j, 0, j - i\} \quad pentru\ orice\ 0 \leq i, j \leq k, i \neq j$$

Pentru $k \leq 1$, orice vector P este k promițător. Soluțiile problemei celor 8 regine corespund vectorilor 8-promițători.

Fie V mulțimea vectorilor k-promițători cu $0 \leq k \leq 8$. Definim graful orientat $G(V, E)$ astfel: $(P, Q) \in E$ dacă și numai dacă există un întreg k astfel încât P este k promițător, Q este $(k + 1)$ promițător și $P[i] = Q[i]$ pentru fiecare $0 \leq i \leq k$. Acest graf este un arbore cu rădăcina în vectorul vid $(k = 0)$ și vârfurile terminale sunt fie soluții $(k = 8)$, fie vârfuri *moarte* $(k < 8)$, în care este imposibil de plasat regina pe următoarea linie fără ca ea să nu *atace* alta deja plasată. Pentru aceasta nu este necesar să generăm, în mod explicit arborele: vârfurile vor fi generate și abandonate pe parcursul explorării.

Vom parcurge arborele G în adâncime, ceea ce este echivalent aici cu o parcurgere în preordine, coborând în arbore numai dacă există șanse de a ajunge la o soluție.

Acest mod de abordare are două avantaje față de algoritmul $regine2$. În primul rând, numărul de vârfuri în arbore este mai mic decât $8!$ și anume pe cale empirică $|V| = 2057$. De fapt, este suficient să explorăm 114 vârfuri pentru a ajunge la prima soluție. În al doilea rând, pentru a decide dacă un vector este $(k + 1)$ promițător, trebuie să verificăm ca ultima regină să nu fie pusă într-o poziție controlată de reginele deja plasate. Mai jos avem algoritmul:

$procedure\ regine(k)$

1. $posibil[1..\mathrm{k}]$ este $k - promiţător$
2. $\boldsymbol{if}\ k = 8\ \boldsymbol{then}\ \boldsymbol{write}\ posibil - am\ găsit\ o\ soluție$
3. $\boldsymbol{else} -\ explorează\ extensiile\ (k+1) promiţătoare\ ale\ lui\ posibil$
4. $\boldsymbol{for}\ j \leftarrow 1\ \boldsymbol{to}\ 8\ \boldsymbol{do}$
5. $\boldsymbol{if}\ plasare(k, j)\ \boldsymbol{then}\ posibil[k+1] \leftarrow j$
6. $regine(k+1)$

$function\ plasare(k, j)$ – $returnează\ true\ doar\ dacă\ se\ poate\ plasa\ o\ regină\ pe\ poz\ (k+1, j)$

$\boldsymbol{for}\ i \leftarrow 1\ \boldsymbol{to}\ k\ \boldsymbol{do}$

    $\boldsymbol{if}\ j - posibil[i] \in \{k+1-i, 0, i-k-1\}\ \boldsymbol{then\ return}\ false$

$\boldsymbol{return}\ true$

Din programul principal se va apela $regine(0)$ presupunând că $posibil[1..8]$ este un tablou global. Problema se poate generaliza, astfel încât să plasăm n regine pe o tablă de n linii și n coloane. Cu ajutorul acestor contraexemple, se poate demonstra că problema celor n regine nu are în mod necesar o soluție.

Iată mai jos schema generală a unui algoritm de backtracking, varianta recursivă:

$procedure\ backtrack(v[1..k])$

1. $v\ este\ un\ vector\ k\ promițător$
2. $\boldsymbol{if}\ v\ este\ solutie\ \boldsymbol{then\ write}\ v$
3. $\boldsymbol{else\ for}\ fiecare\ vector\ w\ (k+1)\ promițător$
4. $astfel\ încât\ w[1..k] = v[1..k]\ \boldsymbol{do}$
5. $backtrack(w[1..k+1])$

Sunt foarte multe aplicații ale algoritmilor de backtracking. Puteți încerca rezolvarea problemelor întâlnite în capitolele anterioare: problema monezilor, a rucsacului. De asemenea se pot rezolva problema colorării hărților, problema comis-voiajorului, în care admitem că există orașe fără legătură directă și nu se cere calculul optim. Parcurgerea în adâncime, folosită în algoritmul regine, devine și mai avantajoasă atunci când ne mulțumim cu o singură soluție a problemei. Sunt unele probleme pentru care acest mod de explorare nu este avantajos.