

Metoda Backtracking

Acest **curs** prezinta **Metoda Backtracking**.

In acest PDF poti vizualiza cuprinsul si bibliografia (daca sunt disponibile) si aproximativ doua pagini din documentul original.

Arhiva completa de pe site contine un fisier, intr-un numar total de **7 pagini**.

Fisierele documentului original au urmatoarele extensii: doc.

Extras

Prezentarea tehnicii Backtracking

Aceasta tehnica se foloseste în rezolvarea problemelor care îndeplinesc simultan urmatoarele conditii:

- solutia lor poate fi pusa sub forma unui vector $S=x_1, x_2, \dots, x_n$, cu $x_1 \in A_1$,

$x_2 \in A_2 \dots, x_n \in A_n$

- multimile A_1, A_2, \dots, A_n sunt multimi finite, iar elementele lor se considera ca se afla într-o relatie de ordine bine stabilita;

- nu se dispune de o alta metoda de rezolvare, mai rapida

- x_1, x_2, \dots, x_n pot fi la rândul lor vectori;

- A_1, A_2, \dots, A_n pot coincide.

La întâlnirea unei astfel de probleme, daca nu cunoastem aceasta tehnica, suntem tentati sa generam toate elementele produsului cartezian A_1, A_2, \dots, A_n si fiecare element sa fie testat daca este solutie. Rezolvând problema în acest mod, timpul de executie este atât de mare, încât poate fi considerat infinit, algoritmul neavând nici o valoare practica.

De exemplu, daca dorim sa generam toate permutarile unei multimi finite A , nu are rost sa generam produsul cartezian $A \times A \times \dots \times A$, pentru ca apoi, sa testam, pentru fiecare element al acestuia, daca este sau nu permutare (nu are rost sa generam $1, 1, 1, \dots, 1$, pentru ca apoi sa constatam ca nu am obtinut o permutare, când de la a doua cifra 1 ne puteam da seama ca cifrele nu sunt distincte).

Tehnica Backtracking are la baza un principiu extrem de simplu:

- se construiesc solutia pas cu pas: x_1, x_2, \dots, x_n

- daca se constata ca, pentru o valoare aleasa, nu avem cum sa ajungem la solutie, se renunta la acea valoare si se reia cautarea din punctul în care am ramas.

Concret:

- se alege primul element x , ce apartine lui A ;

- presupunând generate elementele x_1, x_2, \dots, x_k , aparținând multimilor $A_1,$

A_2, \dots, A_k , se alege (daca exista) x_{k+1} , primul element disponibil din multimea A_{k+1} , apar doua posibilitati :

1) Nu s-a gasit un astfel de element, caz în care se reia cautarea considerând generate elementele x_1, x_2, \dots, x_{k+1} , iar aceasta se reia de la urmatorul element al multimii A_k ramas netestat;

2) A fost gasit, caz în care se testeaza daca acesta îndeplineste anumite conditii de continuare aparând astfel doua posibilitati:

- îndeplineste, caz în care se testeaza daca s-a ajuns la solutie si apar din nou doua posibilitati

- s-a ajuns la solutie, se tipareste solutia si se reia algoritmul considerând generate elementele x_1, x_2, \dots, x_k , (se cauta în continuare, un alt element al multimii A_k , ramas netestat);

- nu s-a ajuns la solutie, caz în care se reia algoritmul considerând generate elementele x_1, x_2, \dots, x_k , si se cauta un prim element $x_{k+2} \in A_k$.

- nu le îndeplineste caz în care se reia algoritmul considerând generate elementele x_1, x_2, \dots, x_k , iar elementul x_{k-1} se cauta între elementele multimii A , ramase netestate.

Algoritmii se termina atunci când nu exista nici un element $x_1 \in A_1$ netestat.

Observatie: tehnica Backtracking are ca rezultat obtinerea tuturor solutiilor problemei. În cazul în care se cere o sigura solutie se poate forta oprirea, atunci când a fost gasita.

Am aratat ca orice solutie se genereaza sub forma de vector. Vom considera ca generarea solutiilor se face într-o stiva. Astfel, $x_1 \in A_1$, se va gasi pe primul nivel al stivei, $x_2 \in A_2$ se va gasi pe al doilea nivel al stivei, ... $x_k \in A_k$ se va gasi pe nivelul k al stivei. În acest fel, stiva (notata ST) va arata astfel:

ST

Nivelul $k+1$ al stivei trebuie initializat (pentru a alege, în ordine, elementele multimii $k+1$). Initializarea trebuie facuta cu o valoare aflata (în relatia de ordine considerata, pentru multimea A_{k+1}) înaintea tuturor valorilor posibile din multime. De exemplu, pentru generarea permutarilor multimii $\{1, 2, \dots, n\}$, orice nivel al stivei va lua valori de la 1 la n . Initializarea unui nivel (oarecare) se face cu valoarea 0. Procedura de initializare o vom numi $INIT$ si va avea doi parametri: k (nivelul care trebuie initializat si ST (stiva)).

Gasirea urmatorului element al multimii A_k (element care a fost netestat) se face cu ajutorul procedurii $SUCCESSOR (AS, ST, K)$. Parametrul AS (am succesori) este o variabila booleana. În situatia în care am gasit elementul, acesta este pus în stiva si AS ia valoarea $TRUE$, contrar (nu a ramas un element netestat) AS ia valoarea $FALSE$.

Odata ales un element, trebuie vazut daca acesta îndeplineste conditiile de continuare (altfel spus, daca elementul este valid). Acest test se face cu ajutorul procedurii $VALID (EV, ST, K)$.

Testul daca s-a ajuns sau nu la solutia finala se face cu ajutorul functiei

.....
.....
.....

Documentul complet de 7 pagini il poti citi daca il descarci din Biblioteca.RegieLive.ro

Imagini din documentul complet:

... mulțime A_1, A_2, \dots, A_n , sunt mulțimi finite, iar elementele lor se consideră că au un singur ordin de ordine bine stabilit;

- nu se depune de a altă mulțime de rezolvare, sau rapid
- x_1, x_2, \dots, x_n , pot fi în ordinea lor venite;
- A_1, A_2, \dots, A_n , pot coincide.

La întâlnirea actuală a problemei, dacă nu conștientizăm această tehnică, atunci trebuie să găsim toate soluțiile problemei rezolvând sistemele A_1, A_2, \dots, A_n , și fiecare element al fiecărei mulțimi de rezolvare, în acest mod, timpul de calcul este atât de mare, încât poate fi considerat infinit, algebră modulară și nu o soluție practică.

De exemplu, dacă dorim să găsim toate soluțiile unui sistem de ecuații A_1, A_2, \dots, A_n , sau un set de parametri care să satisfacă A_1, A_2, \dots, A_n , putem ca apoi, să trecem, pentru fiecare element al sistemului, dacă este un parametru în acest caz, la un sistem de ecuații $1, 1, \dots, 1$, pentru ca apoi să considerăm că nu are soluție o parametră, căci de la a doua ecuație îl putem deforma și să-l eliminăm din sistem (nu sunt diferite).

Teoria Backtracking este în fapt un principiu extrem de simplu:

- se construiește soluția pas cu pas x_1, x_2, \dots, x_n ;
- dacă se constată că pentru o valoare actuală, nu avem soluție și ajungem la soluție, se renunță la acea valoare și se începe din nou înlocuirea în care am rămas.

CONCLUZIE

- se alege primul element x_1 , se ajunge la A_1 ;
- se presupune că elementele x_2, x_3, \dots, x_n , aparținând mulțimilor A_2, A_3, \dots, A_n , se alege (fără costuri) primul element disponibil din mulțimea A_2 , apoi din mulțimile:

- 1) Nu se găsește un astfel de element, unde în care este în care se renunță elementului curent și se începe din nou înlocuirea elementului din mulțimea A_2 următorului;
- 2) A fost găsit, caz în care se înlocuiește din nou înlocuirea actuală condiție de continuare aplicată celui din urmă;

- înlocuirea, care în care se înlocuiește din nou înlocuirea și apoi din nou din nou înlocuirea
- se ajunge la soluție, se înlocuiește soluția și se renunță algoritmul considerând generarea elementelor x_2, x_3, \dots, x_n , (se caută în continuare, un alt element al mulțimii A_1 , din care să înceapă)
- nu se ajunge la soluție, caz în care se renunță algoritmul considerând generarea elementelor x_2, x_3, \dots, x_n , și se caută un nou element x_1 din A_1 ;
- nu se înlocuiește caz în care se renunță algoritmul considerând generarea elementelor x_2, x_3, \dots, x_n , și se caută un nou element din mulțimea A_1 , din care să înceapă.

Algoritm se învârtă atunci când nu există nici un element $x_i \in A_i$, restant.

Observație: Teoria Backtracking are ca rezultat obținerea tuturor soluțiilor problemei, în cazul în care se caută o soluție soluție se poate fi de optime, atunci când a fost găsit.

An altfel al unei soluții se generază prin formă de vector. Vom considera al generarea soluțiilor cu două valori: 0 și 1. Aici, $x_i \in A_i$, se va găsi pe primul nivel al nivelului, $x_i \in A_i$ se va găsi pe al doilea nivel al nivelului, $x_i \in A_i$, se va găsi pe nivelul i al nivelului, în acest fel, se poate să se scrie astfel:

0
1
0
1
0
1
0
1

ST

Nivelul i al nivelului trebuie să fie mai mic decât i , în cazul, elementele mulțimii A_i înlocuiesc toate elementele care au o valoare actuală de valoare de valoare considerată, pentru mulțimea A_i , înlocuiesc toate valorile posibile din mulțime. De exemplu, pentru generarea parametrilor mulțimii $\{1, 2, \dots, n\}$, nivelul actual al nivelului este valoarea de la i la n , înlocuiesc toate nivelurile (construcție) fără ca valoarea i . Procedura de înlocuire a valorii este TRUE și va avea doi parametri în interval care trebuie să fie mai mic decât i (sau 0).

Căutarea soluțiilor element al mulțimii A_i , element care a fost actualizat și face ca sistemul procedurii SET/GENER (A1, ST, C). Parametrul AS (construcție) este o variabilă booleană în situația în care am găsit elementul, acesta este pus la nivel și AS la valoarea TRUE, contrar în cazul în care un element actualizat AS la valoarea FALSE.

Când alți elemente, trebuie să fie mai mic decât valoarea condiției de continuare (soluție) sau, dacă elementul este valid. Acest test se face ca sistemul procedurii VALID (V, ST, C).

Finalizarea a sistemului este soluția finală se face cu ajutorul funcției SOLUTIE (sau a soluției se înlocuiește cu ajutorul procedurii TPAE. Procedura de continuare este Backtracking.

$k=1$; while $k \leq n$ do begin
 while $k \leq 0$ do begin
 repeat generate (in, k, k);
 if se face valabil (in, k);

soluție (sau nu se (sau nu se))
 if se face
 if soluția (sau nu se)
 the begin
 k:=k+1
 until (k, n)
 end

din k:=k+1
 end;

Observație: Problema este rezolvată prin această tehnică în timp îndelungat. De asemenea, este lista de soluții rezolvate atunci când nu avem în dispoziție un alt algoritm mai eficient. Cu toate că există probleme pentru care se pot folosi alți algoritmi mai eficienți, teoria Backtracking trebuie aplicată numai în situații speciale.

Fie un tabel de job, de dimensiune n, n , se cere toate soluțiile de amplasare ale mașinilor, astfel încât să nu se afle două mașini pe aceeași linie, coloană sau diagonala (diagonala de sus și jos).

Exemplu: Propunem să dispunem de a tabel de dimensiune 4×4 , o soluție se face astfel:

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Observăm că două mașini să fie plasate unul pe linie. Pentru prima mașină pe linia 1, coloana 1.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

A doua mașină poate fi plasată în coloana 2.

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Mai multe detalii se gasesc in [pagina documentului din Biblioteca.RegieLive.ro](http://Biblioteca.RegieLive.ro)