Algoritmi de TARE parcurgere

Radu Vișinescu

În numărul 11/1 (ianuarie 2001) al revistei a fost publicat un articol în care a fost introdusă noțiunea de tare-parcurgere a unui graf și au fost prezentați câțiva algoritmi pentru realizarea acestei operații pentru grafuri care reprezintă labirinturi. În cadrul acestui articol vom generaliza acești algoritmi pentru a-i folosi pentru toate tipurile de grafuri.

Introducere

Un labirint este format din camere și coridoare de trecere (uși). Presupunem că într-o anumită cameră se află o persoană. În acest moment persoana nu are acces decât la informațiile pe care le poate obține în camera respectivă. Printre acestea se află informații referitoare la existența ușilor către camerele vecine.

Persoana poate prelucra informațiile pe un calculator pe măsură ce se deplasează prin labirint în scopul culegerii de date. Așadar, vom privi un labirint ca fiind un graf cu informație incompletă. Nodurile grafului vor fi reprezentate de camere, în timp ce muchiile vor fi reprezentate de ușile dintre acestea.

Definiție

Se numește ***labirint*** un graf neorientat $G = (X, M)$, unde X este mulțimea nodurilor și M este mulțimea muchiilor, împreună cu o submulțime de noduri distincte, $X' \subseteq X$, numite noduri de plecare.

În cadrul acestui articol vom presupune că graful este conex și există un singur nod de plecare.

În cazul grafurilor o parcurgere reprezintă o afișare (sau alt tip de prelucrare) a tuturor nodurilor, fiecare nod fiind vizitat o singură dată. Pentru un labirint putem introduce în mod natural noțiunea de tare-parcurgere.

Definiție

Se numește ***tare-parcurgere*** un ciclu care conține toate nodurile grafului împreună cu fixarea numărului de ordine a fiecărui nod în succesiunea din cadrul ciclului.

Astfel se extinde noțiunea de *parcurgere*; cu alte cuvinte, deplasarea se va efectua "din cameră în cameră". Dacă avem dată o anumită parcurgere, o tare-parcurgere care vizitează în aceeași ordine nodurile grafului se va numi *tare-parcurgere* conformă cu respectiva parcurgere. Dacă avem pentru început doar circuitul tare-parcurgerii și labirintul are un singur nod de plecare, putem forma o primă tare-parcurgere prin vizitarea fiecărui nod în momentul în care intrăm pentru prima dată în camera corespunzătoare lui.

Tare-parcurgeri ale vârfurilor

În continuare vom referi tare-parcurgerea definită anterior ca fiind o *tare-parcurgere a vârfurilor*. Ne punem problema dacă în cazul unor parcurgeri clasice putem găsi algoritmi care generează o tare-parcurgere conformă respectivei parcurgeri.

Pargurgerile DF și DL

S-a prezentat în [1] algoritmul tare-parcurgerii conforme parcugerii *DF* în cazul particular al unui labirint pe patru direcții în plan. Prezentăm acum algoritmul pentru cazul general al unui graf:

```
procedură TARE_DF(i)
  vizitează(i)
  marcaj(i) ← 1
  pentru toți vecinii j ai lui i
  execută
    dacă marcaj(j) = 0
    atunci
      TARE_DF(j)
      vizitează(i)
    sfârșit dacă
  sfârșit pentru
sfârșit procedură
```

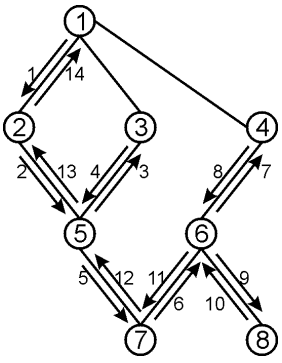

Figura 1

Pentru graful din figura 1 circuitul (ciclul) tare-parcurgerii este 1, 2, 5, 3, 5, 7, 6, 4, 6, 8, 6, 7, 5, 2, 1.

Putem însă să ne imaginăm că vizitarea propriu-zisă are loc la ultima trecere prin fiecare cameră. De exemplu, ar putea dori să mătură labirintul cameră cu cameră și să adune gunoiul în camera de plecare. Obținem astfel un alt tip de parcurgere a grafului pe care o vom numi ***DL*** (***D**eapth **L**ast*).

```
procedură DL(i)
  marcaj(i) ← 1;
  pentru toți vecinii j ai lui i execută
    dacă marcaj(j) = 0 atunci
      DL(j)
    sfârșit dacă
  sfârșit pentru
  vizitează(i)
sfârșit procedură
```

Parcurgerea *DL* corespunzătoare grafului din figura 1 este 3, 4, 8, 6, 7, 5, 2, 1. În cazul în care graful este arbore, parcurgerea *DL* reprezintă o traversare în postordine.

Parcurgerile D și DD

Prezentăm în continuare o variantă a algoritmului care realizează parcurgeri ***D*** (***D**epth*) și ***DD*** (***D**ouble **D**epth*). Vom folosi două stive de lucru: stiva primară (*SP*) și stiva finală (*SF*). Nodurile grafului vor fi inserate succesiv în cele două stive, iar algoritmul se încheie în momentul în care stiva finală devine vidă.

```
procedură TARE_D(i)
  SF ← vidă
  SP ← vidă
  inserează i în SP
  marcaj(i) ← 1
  repetă
    inserează i în SP
    vizitează(i)          // (1)
    inserează i în SF
    pentru toți vecinii j ai lui i execută
      dacă marcaj(j) = 0 atunci
        vizitează(j)      // (2)
        inserează j în SP
        marcaj(j) ← 1
        vizitează(i)
      sfârșit dacă
    sfârșit pentru
    cât timp (vârf(SP) nu este adiacent cu vârf(SF))
    execută
      inserează i în SF
      vizitează(vârf(SF))
    sfârșit cât timp
  până când (SF = vidă)
sfârșit procedură
```

În figura 2 este prezentată o porțiune din circuitul tare-parcurgerii împreună cu modificările realizate până în acel moment în cele două stive:

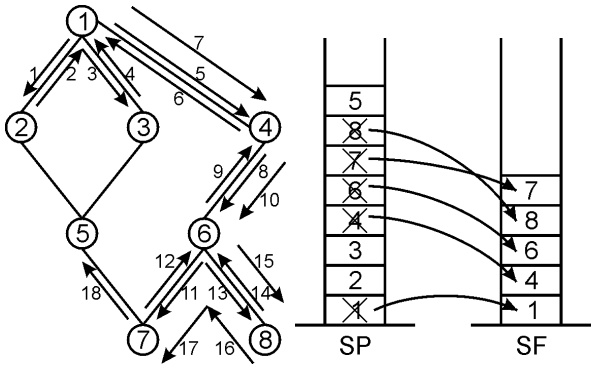

Figura 2

Dacă vizitarea nodurilor are loc doar în momentul inserării lor în prima stivă (linia `(1)`), atunci avem o parcurgere ***D***. Dacă nodurile sunt vizitate numai în momentul inserării lor în stiva finală (linia `(1)`), avem un nou tip de parcurgere pe care am denumit-o ***Double Depth***.

Pentru acest graf, o parcurgere *D* este 1, 2, 3, 4, 6, 7, 8, 5. Parcurgerea *DD* a aceluiași graf este 1, 4, 6, 8, 7, 5, 3, 2.

Încheiem această secțiune cu următoarea remarcă: de cele mai multe ori executarea unei funcții recursive este echivalentă cu o tare-parcurgere a vârfurilor arborelui asociat, tare-parcurgere conformă cu parcurgerile în preordine, inordine sau postordine ([2]).

Tare-parcurgeri ale muchiilor

Vom introduce acum noțiunea de tare-parcurgere a muchiilor unui graf.

Definiție

Se numește ***tare-parcurgere a muchiilor*** un ciclu care trece cel puțin o dată prin toate muchiile grafului.

Nu ne mai punem imediat problema generalizării unei parcurgeri; din definiție lipsește fixarea numărului de ordine pentru nodurile sau muchiilor în circuit (ciclu).

Trebuie observat faptul că această noțiune reprezintă o liniarizare a unui graf. Cu alte cuvinte reprezintă o stocare sub formă liniară a tuturor informațiilor necesare pentru refacerea grafului inițial. Prezentăm un prim algoritm pentru tare-parcuregerea muchiilor, obținut prin modificarea algoritmului de parcurgere în adâncime:

```
procedură TARE_MUCHII(i)
  vizitează(i)
  marcaj(i) ← 1
  pentru toți vecinii j ai lui i execută
    dacă marcaj(j) = 0 atunci
      TARE_MUCHII(j)
      vizitează(i)
    altfel
      vizitează(j)
      vizitează(i)
    sfârșit dacă
  sfârșit pentru
sfârșit procedură
```

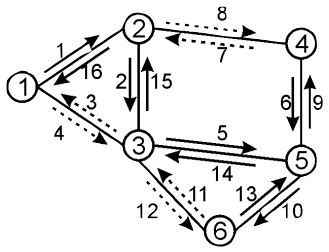

Figura 3

Pentru graful din figura 3 tare-parcurgerea *DF* a muchiilor este următoarea: 1, 2, 3, 1, 3, 5, 4, 2, 4, 5, 6, 3, 6, 5, 3, 2, 1.

Determinarea unei baze de cicluri fundamentale

Prin *baza de cicluri fundamentale* a unui graf înțelegem o mulțime maximală de cicluri în care fiecare ciclu conține cel puțin o muchie care nu face parte din nici unul dintre celelalte cicluri.

Pentru determinarea bazei de cicluri fundamentale a unui graf vom începe cu o tare-parcurgere *DF* a muchiilor grafului.

În continuare vom prelucra vectorul de noduri obținut astfel:

```
procedură cicluri(x:vector; l:lungime)
  variabile aux:vector
            k:întreg
  k ← 0
  pentru i ← 1, l execută
    k ← k + 1
    aux(k) ← v(i)
    găsit ← fals
    pentru j ← 1, k - 1 execută
      dacă aux(j) = aux(i) atunci
        găsit ← adevărat
        poz ← j
      sfârșit dacă
    sfârșit pentru
    dacă găsit atunci
      dacă poz = k - 2 atunci
        k ← k - 2 // un ciclu trebuie să conțină cel
                     puțin două muchii
      altfel
        scrie 'Ciclul:'
        pentru x ← poz + 1, k execută
          scrie aux(x), ' '
        sfârșit pentru
      sfârșit dacă
    sfârșit dacă
  sfârșit pentru // k va avea valoarea 1
sfârșit procedură
```

În figura 4 este prezentat conținutul vectorului aux în diverse puncte ale execuției algoritmului pentru graful din figura 3.

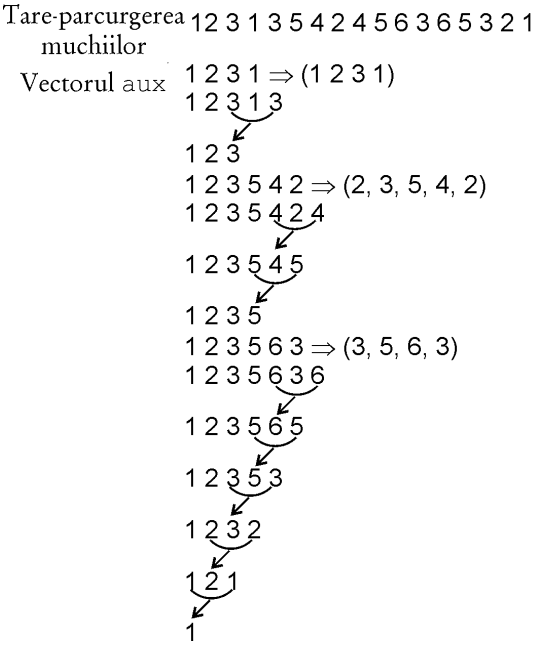

Figura 4

Problema comis-voiajorului chinez

În [3] este analizată *problema comis-voiajorului chinez*. Aceasta poate fi enunțată astfel: *Fie un graf neorientat cu costuri pe muchii; să se determine un traseu care trece cel puțin o dată prin fiecare muchie și are costul minim.*

Problema se reduce la determinarea unei tare-parcurgeri de cost minim a muchiilor. În [3] este prezentată o rezolvare în timp polinomial a acestei probleme.

Este firesc să ne punem întrebarea dacă există o astfel de rezolvare polinomială în cazul în care impunem condiția ca circuitul să treacă cel puțin o dată prin fiecare nod. Cu alte cuvinte, dorim să găsim un algoritm polinomial cu ajutorul căruia se poate determina o tare-parcurgere optimă a vârfurilor.

Dacă am găsi un algoritm polinomial de rezolvare a acestei probleme, am avea, practic, un algoritm polinomial pentru determinarea unui ciclu hamiltonian de cost minim. Așadar, problema se reduce la una *NP-completă*.

Analiza complexității

Pentru a studia complexitatea algoritmilor descriși alegem ca operație fundamentală deplasarea unei persoane dintr-o celulă în alta. În cazul tare-parcurgerii *DF* realizăm $2 \cdot (n - 1)$ astfel de deplasări, deci ordinul de complexitate este $O(n)$, unde n este numărul nodurilor. Pentru tare-parcurgerea *D* sunt necesare $4 \cdot (n - 1)$ deplasări, deci ordinul de complexitate este tot $O(n)$. Pentru tare-parcurgerea *DF* a muchiilor avem nevoie de $2 \cdot m$ deplasări, deci ordinul de complexitate este $O(m)$, unde m este numărul muchiilor.

Bibliografie

1. R. Vișinescu, *Algoritmi de tare-parcurgere*, Gazeta de informatică, ianuarie 2001
2. L. Livovschi, H. Georgescu, *Algoritmi. Elaborare și complexitate*
3. N. Christofides, *Graph theory: An algorithmic approach*