Algoritmi de TARE-PARCURGERE

Radu Cristian Vișinescu

Acest articol introduce conceptul de tare-parcurgere a unui graf. În acest sens, sunt descrise adaptări ale unor algoritmi de parcurgere cunoscuți cum ar fi BF, D sau DF; în final se introduce parcurgerea DL (Deapth Last). Nu în ultimul rând, acest articol este o "pledoarie în favoarea lui Tezeu".

Problema labirintului

Considerăm un labirint ca fiind o construcție pătratică formată din camere (celule). Un labirint se poate codifica printr-o matrice A de ordin n în care intrarea a_{ij} este un număr între 0 și 15 cu următoarea semnificație: *dacă se consideră vecinii camerei (i, j) in ordinea nord, est, sud, vest și se reține 1 în cazul existenței unei uși spre vecinul respectiv și 0 în caz contrar; numărul binar de patru cifre astfel format este convertit în baza 10.*

Observație

O ușă se poate deschide în ambele direcții, deci nu orice tablou pătratic având ca elemente numere între 0 și 15 codifică un labirint; elementele tabloului nu sunt total independente.

Unii consideră că problema ieșirii dintr-un labirint nu se poate rezolva în timp polinomial. De exemplu, într-un articol în care era vorba de dificultatea găsirii unor algoritmi eficienți în inteligența artificială, pentru rezolvarea automată de probleme era folosită în mod eronat următoarea analogie: *dacă operația de verificare a corectitudinii unei demonstrații este analoagă verificării dacă un drum dat asigură ieșirea din labirint, găsirea unei demonstrații corespunde construirii unui astfel de drum.*

Abordarea prin teoria grafurilor

Vom arăta că putem găsi un drum de ieșire din labirint într-un timp $O(n^2)$. Mai mult, determinarea drumului de lungime minimă de ieșire dintr-un labirint poate fi realizată și ea într-un timp $O(n^2)$.

Distanța dintre camere se presupune a fi constantă și egală cu 1.

Un labirint este de fapt un graf cu n^2 vârfuri și cu mai puțin de $4n^2$ muchii. Drept noduri considerăm camerele iar drept muchii ușile dintre camere. O parcurgere DF a acestui graf se efectuează într-un timp $O(n^2)$, aplicarea acestui algoritm asigură deja găsirea unei soluții într-un timp $O(n^2)$. Algoritmul de parcurgere DF este următorul:

```
algoritm DF(i)
  vizitează nodul i
  marcat(i)← adevărat
  pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
    DF(j)
  sfârșit pentru
```

Inițializăm marcajele cu valoarea 0 și apelăm procedura pentru camera de plecare.

Pe același graf putem aplica o parcurgere tip BF în care vom determina în paralel distanțele minime de la camera de plecare și toate celelalte camere (algoritmul lui *Lee*). Oprim execuția parcurgerii BF în momentul în care ajungem într-o cameră de ieșire.

Algoritmul lui Lee este prezentat în continuare:

```
algoritm Lee(i)
  Coada ← vidă
  STIVA ← vidă
  vizitează nodul i
  marcaj(j) ← adevărat
  d ← 0
  Coada ← (i,0,d)
  cât timp Coada nu este vidă execută
    (i,pred_i,d) ← Coada
```

```
    STIVA ← (i,pred_i)
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcaj(j) ← adevărat
      Coada ← (j,i,d+1)
    sfârșit pentru
  sfârșit cât timp
```

În acest caz inițializăm, de asemenea, marcajele cu 0 și apelăm procedura pentru camera de plecare.

Ordinul de complexitate al acestei parcurgeri BF este liniar în numărul de noduri ale grafului, deci este $O(n^2)$ în cazul problemei de față.

Drumul de la prima cameră de ieșire găsită la camera de plecare se poate determina printr-o singură parcurgere în sens invers a vectorului STIVA. Acest vector are dimensiunea cel mult n^2.

Să analizăm mai atent algoritmul:

- Coada va conține o listă de noduri vizitate.
- Fiecare nod este vizitat și intră în coadă o singură dată.
- Prelucrarea unei camere, cu alte cuvinte vizitarea vecinilor nemarcați, se execută la extragerea din coadă. În final în STIVA se vor afla toate nodurile prelucrate până în momentul găsirii camerei de ieșire, fiecare dintre ele având legătura înapoi către camera de pornire (prin concatenerea acestor legături se determină drumul cel mai scurt).

În concluzie, problema găsirii drumului optim de ieșire este rezolvată folosind algoritmul lui *Lee,* tot într-un timp $O(n^2)$.

În continuare vom rafina analiza complexității notând cu n' numărul de noduri care pot fi efectiv parcurse (cele din componenta conexă care conține camera de pornire) și cu m' numărul de muchii ale acestei componente conexe. Ambii algoritmi prezentați anterior au ordinul de complexitate $O(n'+m')$.

Observație

Algoritmul "clasic" de găsire a tuturor ieșirilor din labirint (dat spre exemplu în **Tudor Sorin**, *Tehnici de programare*) poate determina prima soluție după un timp exponențial.

Problema labirintului pe cazul real

Problema reală a omului prins în labirint este de fapt o reformulare a problemei inițiale și anume transformarea ei într-o problemă cu informație incompletă.

Labirintul este "dat" în întregime la început, dar omul prins în labirint nu dispune de întreaga sa hartă. În camera de pornire nu cunoaște la început decât existența a cel mult patru uși de legătură cu vecinii direcți (nord, est, sud și vest). Nu va cunoaște existența unei uși între nodurile i și j până când nu ajunge fie în camera i, fie în camera j.

De aici rezultă că la aplicarea oricărui algoritm de ieșire din labirint trebuie impusă condiția ca această parcurgere să fie efectuată "din cameră în cameră".

Cu alte cuvinte, în analiza complexității acestor algoritmi este natural să considerăm ca operație fundamentală trecerea dintr-o cameră într-o cameră vecină.

În cazul parcurgerii DF numărul de operații fundamentale efectuate coincide cu numărul de vizitări. Dacă renunțăm la optimalitate, prin acest algoritm găsim o ieșire din labirint într-un timp $O(n')$.

Vom arăta că aplicarea algoritmului lui *Lee* duce la un timp de ordinul $O(n'^2)$.

Prin acest algoritm putem reține în permanență, pentru fiecare cameră vizitată, lanțul care o unește cu camera de pornire.

Acest lucru poate fi realizat eliminând stiva, și introducând în coadă fiecare nod vizitat împreună cu întreg lanțul care îl unește cu nodul de plecare.

Deoarece următoarea cameră care trebuie prelucrată este prima de la ieșirea din coadă, pentru a o putea accesa este suficient să parcurgem lanțul din camera curentă (ultima prelucrată) în camera de plecare și apoi din camera de plecare în camera care trebuie prelucrată.

Distanța maximă dintre oricare două camere este $n' - 1$, deci acest pas de parcurgere "din cameră în cameră" este realizat în maximum $2n'$ pași. Pasul trebuie repetat pentru fiecare dintre cele n' camere. În concluzie, complexitatea în timp devine $O(n'^2)$, în raport cu operația fundamentală considerată.

Vom descrie acum un labirint pentru care aplicarea parcurgerii BF astfel modificată duce la un timp pătratic față de mulțimea efectivă a nodurilor. Considerăm de la sud la nord lanțul de camere 1, 2, …, n',camera de pornire fiind n', iar spre est lanțul $n' + 1$, $n' + 2$, …, $2n'$. Conform algoritmului vom trece din camera $n' - i$ în camera $n' + i$ pentru fiecare i. Suma deplasărilor va fi $2(1 + 2 + \ldots n')$, deci ordinul de complexitate va fi $O(n'^2)$.

Consumul de memorie suplimentară este foarte mare; omul prins în labirint trebuie să rețină în permanență date referitoare la subgraful deja parcurs.

Iată două argumente care conduc și ele către alegerea parcurgerii DF.

Concret, în plan, aceasta revine la următoarele: la fiecare intersecție se alege drumul cel mai din stânga și, în plus, trebuie să existe un "fir al *Ariadnei*" care leagă în permanență camera curentă de camera de plecare. Vom prezenta și o condiție suficientă pentru a ne putea dispensa de acest "fir al *Ariadnei*", precum și motivul pentru care acesta este necesar în cazul general.

Cazul labirintului arbore

În continuare vom considera că grafurile care reprezintă labirintul sunt "scufundate" în plan. Astfel vom putea da o orientare muchiilor din fiecare cameră de intersecție, deoarece planul poate fi considerat ca fiind orientat. Corectitudinea următorilor algoritmi se bazează pe astfel de raționamente de orientare.

Dacă labirintul reprezintă un graf care este și arbore, atunci în parcurgerea sa putem face abstracție de conținu-

tul stivei. De fapt, vom descrie un algoritm de parcurgere în plan a unui arbore fără memorie suplimentară. Ideea de bază este de a alege la fiecare intersecție cel mai din stânga drum (care, la limită, este ușa pe care tocmai am intrat).

```
algoritm arbore(i)
  i0 ← i
  scrie i
  d ← N
  repetă
    d ← succ(succ(succ(succ(d))))
                        /* "prima la stânga" */
    cât timp nu există nod pe direcția d execută
      d ← succ(d)
    sfârșit cât timp
    j ← un nod de pe direcția d
    i ← j
  scrie i
  până când i=i0
```

Observații

- Considerăm cele patru direcții nord, este, sud și vest și definim funcția `succ` astfel încât `succ(nord)=est`; `succ(est)=sud`; `succ(sud)=vest`; `succ(vest)=nord`. Arborele va avea cel puțin două noduri.
- În cazul existenței unui posibil ciclu în labirint acest ultim algoritm poate eventual intra într-un ciclu infinit. Așadar acesta este un algoritm euristic.
- Există camere vizitate de mai multe ori. O vizitare este indicată de un apel **scrie**.
- În cazul real ordinul de complexitate este $O(n')$

Algoritmul folosit de Tezeu

Așa cum am mai spus, din punctul de vedere al omului prins în labirint, problema este cu informație incompletă. Deoarece labirintul în care a fost prins *Tezeu*, nu era neapărat de tip arbore, el a avut nevoie de "firul *Ariadnei*" și a ales cu siguranță parcurgerea DF.

Ca memorie suplimentară este folosit doar un vector de marcaj de lungime n' (elementele marcate constituie "firul *Ariadnei*"). Algoritmul este următorul:

```
algoritm Tezeu(i)
  vizitează nodul i
  marcaj_vizit(i) ← adevărat
  scrie i
  pred ← 0
  marcaj(i) ← adevărat
  repetă
    d ← succ(succ(succ(succ(d))))
                          /* prima la stânga */
    cât timp nu există nod pe direcția d sau există nod
          marcat pe direcția d și nodul marcat de pe
                direcția d nu este predecesorul nodului
                                        curent execută
      d ← succ(d)
    sfârșit cât timp
    j ← un nod de pe direcția d
    dacă nu marcaj_vizit(i) atunci
      vizitează nodul i
      marcaj_vizit(i) ← adevărat
    sfârșit dacă
    dacă j=pred atunci marcaj(i) ← fals
                altfel marcaj(i) ← adevărat
    sfârșit dacă
    i ← j
    scrie i
    dacă există p vecin al lui i și marcaj(p)
      atunci pred ← p
      altfel stop
    sfârșit dacă
  până când fals
```

Să analizăm mai atent algoritmul:

- Pentru fiecare nod după vizitare determinăm predecesorul acestuia ca fiind acel unic nod vecin care este și marcat.
- Singurul nod care nu are predecesor în acest punct va fi nodul de pornire (este totuși și el marcat).
- După calculul predecesorului prin tehnica "prima la stânga" calculăm succesorul nodului `i`. Trebuie tratate două cazuri: *succesorul coincide sau nu cu predecesorul*. Aceste cazuri sunt tratate corespunzător actualizării "firului *Ariadnei*".

Observații

- Ca memorie suplimentară se folosesc doar doi vectori de biți.
- În raport cu operația fundamentală algoritmul are ordinul de complexitate $O(n')$, producând la ieșire un circuit cu $2(n - 1)$ muchii, circuit ce parcurge arborele DF.
- Circuitul tare-parcurgerii este listat prin apeluri **scrie**.

Considerații de teoria grafurilor

Problemele de parcurgere prezentate se extind natural la cazul unui graf conex. Acest graf este dat prin listele de adiacență (liste circulare), pentru fiecare nod reținându-se și gradul său.

Așadar, găsirea succesorului după regula "prima la stânga" poate fi ușor generalizată.

Problema reală a labirintului conduce în mod natural la următoarea noțiune de tare-parcurgere a unui graf:

Definiție

Fie un graf neorientat $G = (V, M)$. Vom numi o tare-parcurgere a sa un circuit care trece cel puțin o dată prin toate vârfurile grafului G, împreună cu o alegere pe acest circuit a mulțimii vârfurilor lui G.

Observație

Noțiunea de parcurgere de până acum însemna o liniarizare (listare sau altă operație) a nodurilor grafului, așadar o tare-parcurgere determină o parcurgere prin nodul său de

plecare. Vom spune că tare-parcurgerea respectivă este conformă cu parcurgerea determinată de ea.

Să rezumăm:

- În primii doi algoritmi vizitarea nodurilor poate fi pusă în corespondență cu listarea circuitului corespunzător unei tare-parcurgeri. Prin utilizarea unui vector suplimentar de marcaj care are la început toate valorile setate pe 0, și executarea unei operații asupra lui la prima vizitare (de exemplu, marcarea sa ca nod ales pe circuit), se completează generarea unei tare-parcurgeri conform definiției date. În concluzie, parcurgerea DF poate fi imediat adaptată să furnizeze o tare-parcurgere conformă cu parcurgerea propriu-zisă. Algoritmul nu este optim în raport cu operația fundamentală (furnizează o tare-parcurgere de circuit de lungime minimă).
- În cele prezentate anterior am adaptat algoritmul BF (*Lee*) care acum poate furniza o tare-parcurgere conformă cu parcugerea clasică, în ordinea distanțelor minime de la nodul de plecare. Complexitatea este $O(n'^2)$ în raport cu operația fundamentală.
- Ne punem problema dacă orice algoritm de parcurgere poate fi adaptat pentru a genera o tare-parcurgere conformă cu respectiva parcurgere.

Cazul algoritmului de parcurgere D (Depth)

Vă prezentăm în continuare descrierea în pseudocod a algoritmului clasic. Această descriere îi aparține d-lui ***prof. dr. Horia Georgescu***.

```
algoritm D(i)
  vizitează nodul i
  marcat(i) ← adevărat
  STIVA ← vidă
  repetă
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcat(j) ← adevărat
      STIVA ← j
    sfârșit pentru
    dacă STIVA=vidă atunci stop
                    altfel STIVA←i
      sfârșit dacă
    până când fals
```

Observații

- Stiva conține o listă de noduri vizitate și neprelucrate
- La fiecare pas al buclei **repetă** prelucrăm un nod curent i; în acest moment toate nodurile aflate în stivă au proprietatea că sunt legate de un nod din drumul de la nodul de plecare la nodul i, drum care va fi memorat, pentru simplitate, în altă stivă auxiliară.
- Algoritmul are ordinul de complexitate $O(n'+m')$

Vă prezentăm în continuare descrierea în pseudocod a adaptării sale pentru a genera o tare-parcurgere conformă.

```
algoritm D_tare(i)
  vizitează nodul i
  marcat(i) ← adevărat
  STIVA ← vidă
  S_AUX ← i
  scrie i
  repetă
    pentru toți vecinii j ai lui lui i pentru care
                          marcat(j)=fals execută
      vizitează nodul j
      marcat(j) ← adevărat
      STIVA ← j
    sfârșit pentru
    S_AUX ← vârful stivei STIVA
    scrie vârful stivei STIVA
    dacă STIVA=vidă atunci stop
    altfel
      STIVA ← i
      repetă
        S_AUX ← x
        scrie x
      până când x este vecin cu i;
      S_AUX ← x
    sfârșit dacă
  până când fals
```

Concluzii

Adaptarea algoritmului D generează la rândul său o tare-parcurgere de circuit de lungime minimă. De fapt, tare-parcurgerea generată de ambii algoritmi este identică din punct de vedere al circuitului parcurs.

De ce a ales *Tezeu* algoritmul DF în locul algoritmului D? Din punctul său de vedere ambii algoritmi sunt echivalenți ca ordin de complexitate. Probabil că în labirint era întuneric și nu putea să marcheze sau să memoreze camerele. Aceasta revine la minimizarea memoriei suplimentare.

Dăm în încheiere o definiție a unui labirint ca fiind un *graf cu informație incompletă*.

Definiție

Numim labirint un graf neorientat G = (V, M) împreună cu o submulțime V_p de vârfuri distincte numite noduri de plecare.

Bibliografie

1. **Atanasiu A.**, *Culegere de probleme Pascal*, Ed. Petrion
2. **Knuth D.E.,** *Algoritmi fundamentali*, Ed. Tehnică - 1976
3. **D.E.Knuth,** *Sortare și căutare*, Ed. Tehnică - 1976
4. **Gazeta Matematică**, colecție
5. **PC REPORT,** colecție
6. **Leon Livovschi, Horia Georgescu,** *Bazele informaticii - Algoritmi. Elaborare și complexitate*

Domnul profesor Radu Cristian Vișinescu este cadru didactic la Liceul "Ion Luca Caragiale" din Ploiești. Poate fi contactat prin e-mail la adresa **raduv@lphilc.sfos.ro.**