ALGORITM NP-COMPLET PENTRU DETERMINAREA ARBORELUI PARȚIAL CU NUMĂR MAXIM DE FRUNZE

Ovidiu Domşa

Abstract. Searching algorithms are the most common for data structares like string, heap or graph. We propose to solve a NP complete problem of searchiog in a large graph. The algorithm try to find a lubstructure with properties in a real time.
Keywords. *alghorithm, graph, tree, data structures*

I. TEMA

Una din structurile de date care permit reprezentarea datelor şi a legăturilor dintre ecestea este graful conex. De regulă căutarea soluțiilor unor probleme pe structuri ramificate presupune un consum mare de timp şi memorie. Ne propunem in continuare să dăm un algoritm care rezolvă o problemă de căutare a unei structuri cu o anumită proprietate într-un graf..

II. PROBLEMA

Se consideră un graf neorientat G=(X,U), conex, cu n noduri. Să se determine unul din arborii parţiali A=(X, V), V submultime a lui U, ai grafului dat, a cărui număr de frunze este maxim. Exemplu:

Pentru exemplificare am ales un graf conex cu n = 8 noduri şi 15 muchii, Fig. 1.

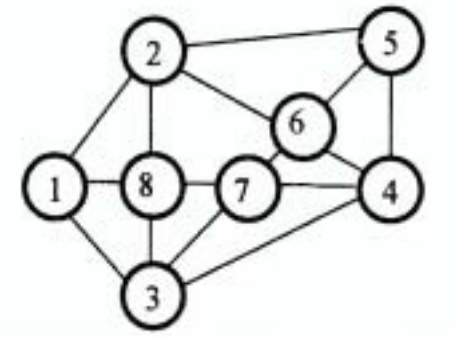

Fig. 1. *Graf neorientat cu 8 vârfuri*

Pentru graful de mai sus un arbore parțial cu număr maxim de frunze este dat in Fig.2.

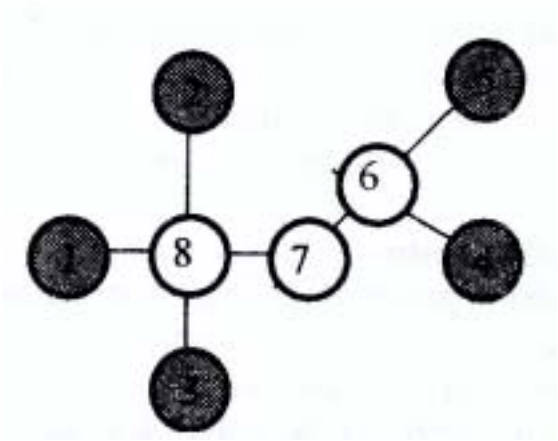

Fig.2. Arbore parțial cu număr maxim de frunze

Date de intrare

În fişierul **ARBORE.IN** pe prima linie este scris un număr natural n (3 < n < 2000) reprezentând numărul nodurilor. Pe următoarele n linii urmează descrierea grafului, care va avea cel mult 10000 de muchii şi este conex. Primul număr de pe cea de a j+l-a linie precizează numărul de noduri k, adiacente cu nodul j. După acest număr urmează k numere naturale $x_k e\{l,n\}\backslash\{j\}$ reprezentând nodurile adiacente nodului j. Datele scrise pe aceeaşi linie sunt despărțite prin câte un spațiu.

Date de ieşire

În fişierul **ARBORE.OUT,** pe prima linie se va sscrie numărul maxim de frunze. Pe următoarele n-1 linii se vor scrie perechi de numere naturale (despărțite printr-un spațiu) reprezentând muchiile arborelui având acest număr maxim de frunze, în cazul în care există mai multe soluții, se va preciza una singură. Ordinea în care se afişează muchiile şi ordinea în care se precizează vârfurile muchiilor este oarecare.

Exemplu

ARBORE. CSI	**ARBORE.OUT**
8	5
3238	18
41568	28
41478	38
43567	78
3246	76
42457	46
43468	56
41237	

III. MODURI DE REZOLVARE

O primă idee se bazează pe observația că problema, fiind NP-completă, ne trebuie o strategie euristică, dar care va fi optimizată astfel încât să asigure o soluție cât mai apropiată de optimul cerut

Algoritmul următor este de 2-aproximare (în cazul cel mai defavorabil rezultatul este de cel mult de două ori mai rău decât optimul căutat). Poate fi implementat pentru a rula într-un timp liniar, folosind structuri de date simple. {Galbiati et. al [2] au demonstrat că problema este MAX-SNP completă, şi deci nu există nici un algoritm de aproximare în timp polinomial pentru problemă, decât dacă P = NP. }

Există în literatura de specialitate rezultate precum: orice graf conex cu *n* vârfuri şi grad minim *k=3* are un arbore de acoperire cu cel puțin n/4+2 frunze. Pentru *k=4* s-a demonstrat că numărul de frunze este cel puțin (2«+8)/5 etc.

Algoritmul nostru foloseşte *reguli de expansiune.* Totuşi, atribuim priorități regulilor şi le folosim pentru a construi o pădure în locul unui arbore. In mod întâmplător, pădurea *F* construită de regulile noastre este o pădure cu frunze, şi deci profităm de structura sa specială pentru a construi un arbore de acoperire cu un număr de frunze apropiat de cel al pădurii.

Informai, regulile de expansiune de prioritate mică folosite de algoritm măresc numărul de frunze din pădure cu o cantitate mică, iar regulile de prioritate mare măresc numărul de frunze al pădurii cu o cantitate mare. Putem demonstra că factorul de aproximare al algoritmului este 2 arătând că fiecare regulă de prioritate mică adaugă pădurii cel puțin un vârf care trebuie să fie interior în orice arbore T' cu număr maxim de frunze. Mai mult, această mulțime de vârfuri interne este disjunctă față de mulțimea vârfurilor interne necesare pentru interconectarea subarborilor induşi pe T' de vârfurile acoperite de *F.* Aceasta este suficient pentru de demonstra marginea de 2 pentru factorul de aproximare al algoritmului.

IV. ALGORITM

Fie *G=(V,E)* un graf neorientat conex. Notăm cu *m* numărul de muchii şi cu n numărul de vârfuri al lui G. Fie T' un arbore de acoperire al lui G având un număr maxim de frunze.

Algoritmul construieşte întâi o pădure *F* folosind un şir de *reguli de expansiune,* ce vor fi definite imediat. Apoi arborii din *F* sunt conectați între ei, formând un arbore de acoperire *T.* Când arborii din *F* sunt uniți, unele frunze ale lui *F* devin vârfuri interne ale lui *T.* Spunem că o frunză din *F* este *ucisă* dacă ea devine un vârf intern al lui T. Regulile de expansiune folosite pentru construirea lui *F* sunt concepute in aşa fel încât o mare parte din vârfurile lui *F* sunt frunze şi când Teste format, se ucide cel mai mic număr de frunze posibil din *F.*

Fiecare arbore T_i al pădurii *F* este construit alegând întâi un vârf cu gradul cel puțin 3 drept rădăcină a sa. (Dacă *n* nu este foarte mare poate fi utilă o ordonare a vârfurilor în sens descrescător după gradele lor.) Apoi se folosesc regulile de expansiune. Aceste reguli sunt aplicate frunzelor arborelui. Dacă o frunză *x* are cel puțin doi vecini ce nu se află în T_i toți vecinii lui *x* ce nu aparțin arborelui T_i devin copii ai lui. Dacă x are un singur vecin *y* care nu

aparține lui *T,* şi cel puțin doi vecini ai lui *y* nu sunt în T_i atunci îl punem pe *y* drept unic fiu al lui *x* şi toți vecinii lui *y* ce nu se află în *T,* devin copii ai lui *y.* Când regula este aplicată unui vârf *x* spunem că vârful *x* este *expandat* de regulă.

Atribuim priorități regulilor de expansiune după cum urmează. Regula prin care se expandează o frunză *x* care are un singur vecin *y* ce nu se află în *F* şi y are exact doi vecini ce nu aparțin lui F are prioritatea 1. Toate celelalte reguli de expansiune au prioritatea 2. Când construim un arbore, dacă două frunze diferite pot fi expandate, este expandată prima cea care poate fi expandată cu o regulă de prioritate mai mare. Dacă două frunze pot fi expandate cu reguli de aceeaşi prioritate, atunci se alege una în mod arbitrar pentru expansiune. Un arbore T_i, este expandat până când nici o regulă de expansiune nu mai poate fi aplicată frunzelor sale.

Regula l adaugă două noi frunze unui arbore T_i şi în acelaşi timp ucide o frunză din T_i.

Algoritm *arbore(G)*
$F \leftarrow \emptyset$
F4-O
cât timp
există un vârf v de grad cel puțin 3
execută
Construieşte un arbore T_i cu rădăcina v şi având ca frunze vecinii lui v
cât timp
cel puțin o frunză din T_i poate fi expandată
execută
găseşte frunza din T_i care poate fi expandată cu o regulă de prioritate maximă şi se expandează
sfârşit cât timp
$F \leftarrow F \cup T_i$,
Elimină din G toate vârfurile din Ti şi toate muchiile incidente lor
sfârşit cât timp
Uneşte arborii din F şi vârfurile ce nu se află în F pentru a forma un arbore de acoperire T.

V. ANALIZA ALGORITMULUI

Fie $F = \{ T_0, \ldots, T_k \}$ pădurea construită de algoritm, şi fie *X* mulțimea vârfurilor neacoperite de *F.* Vom numi *X* mulțimea *vârfurilor exterioare.* E uşor de observat că un vârf exterior nu poate fi adiacent unui vârf intern al unui arbore T_i. Regulile de expansiune ne permit să demonstrăm următoarele proprietăți ale vârfurilor exterioare.

Lema 3.1 *Fie G'=(V, E') graful format prin contractarea fiecărui arbore $T_i \in F$ într-un singur vârf şi prin eliminarea muchiilor multiple între perechile de vârfuri. In G fiecare vârf exterior are gradul cel mult egal cu 2.*

Demonstrație. Să presupunem că există un vârf $v \in X$ care are gradul cel puțin egal cu 3 în G'. Să considerăm 3 dintre vecinii lui v. Se observă că aceste 3 vârfuri nu pot fi vârfuri exterioare pentru că atunci algoritmul *arbore* l-ar fi ales pe v ca rădăcină a noului arbore. De aici, cel puțin un vecin al lui v se află în *F*. Fie T_i , primul arbore generat de algoritm, care conține unul din vecinii u ai lui v. Deoarece v este adiacent cu două vârfuri ce nu se află in T_i, algoritmul *arbore* l-ar fi expandat pe *u.*

Lema 3.2 *Fie u o frunză a unui arbore* $T_i \in F$. *Dacă u este adiacent cu două vârfuri v, w* $\in T_i$, *atunci v şi w sunt w frunze ale aceluiaşi arbore* $T_j \in F$.

Demonstrație. Evident, v şi w nu pot fi ambele vârfuri exterioare. De asemenea, nici v nici w nu pot fi vârfuri interne ale unui arbore T_i , pentru că dacă, de exemplu, v este un vârf intern al lui T_j , atunci

1. dacă algoritmul construieşte arborele T_i înaintea arborelui T_i atunci vârful *u* ar fi plasat drept copil al lui v în T_i, şi

2. dacă T_i este construit primul, atunci v ar fi fost plasat drept copil al lui *u.* Pentru a demonstra acest lucru trebuie să observăm că fiecare vârf intern al unui arbore $T_i \in F$ are cel puțin 3 vecini în T_i. . Astfel, vârful w ar fi fost expandat în timpul construirii arborelui T_i.

De aici, cel puțin unul din vârfurile *u, w* trebuie să fie o frunză în *F*. Fie v o frunză într-un arbore T_j. Fie *p* părintele lui u în T_i. Dacă w nu este o frunză în T_j, atunci putem presupune fără a reduce generalitatea că, atunci când algoritmul *arbore* adaugă vârful v arborelui T_i, vârful v nu aparține lui *F*. Se observă că arborele T_i nu poate fi construit înaintea lui T_j, pentru că algoritmul 1-ar fi plasat pe v şi w drept copii ai lui *u.* Deci fie T_j construit înaintea lui T_i. Atunci vârfurile *u, p şi w nu se* află încă în *F* atunci când este construit T_j. Aceasta înseamnă că algoritmul *arbore* 1-ar expanda pe v şi 1-ar plasa pe *u* drept copil al său în T_i . De aici, v si w trebuie să fie frunze în T_i.

A doua modalitate de rezolvare este de asemenea euristică.:

Inițial se marchează, toate nodurile cu 0.

Se alege nodul de grad maxim şi se marchează fiecare vecin al acestuia cu l, şi se speră că dintre acestea unele vor rămâne frunze.

Se parcurg pe rând, în ordine descrescătoare a gradelor, aceste noduri şi se procedează similar, nodurile care devin noduri interne se marchează cu 2.

Algoritmul se termină când nu există nici un nod având marcajul 0.

VI. PROGRAMARE C

```
#include <stdio.h> tfinclude <stdlib.h>
#defineNMAX4000 #define MMAX 10000
int *1[NMAX+1], nl[NMAX+l], deg[NMAX+l],
m[NMAX+l];
int IsafNMAX], IsbfNMAX], nls;
int i j,k,t,n,max,nf,pr, nm;
```

```
void citeste()
{
FILE* f= fopen("arbore.in","rt");
fecanf(f,"%d",&n);
for(i=l;i<=n;i++)
{
fscanf(f,"%d",&t);
nl[i]= t; if (0[i]= new int[t+l])==NULL) {puts("alloc err"); exit(l);};
for (k=l; k<=t; k++) ftcanf(f,11%d",&l[i][k]);
}
fclose(f);
}
void mark(int a, int b)
{
nls-H-; lsa[nls]= a; lsb[nls]= b;
static int i;
for(i=l;i<=nl[b];i++) deg[l[b][i]]-;
nf++; nm-H-;
m[b]=l;
}
void rezolva()
{
for (i=l; i<=n;i-H-) deg[i]= nl[i];
pr=1;
while (1)
{
max=0; for(i=l;i<=n; i++)
if ((m[i]==l || pr) && deg[i]>max)
{
max= deg[i]; k=i;
}
if (pr) { mark(0,k); nls--; pr= 0; }
if (!max)break;
nf--;
m[k]= 2;
for(i=l;i<=nl[k];i++)
if(!ml[[k][i]])mark(k,l[k][i]);
}
if (nm<n) {puts("Neconex!");>;
}
void scrie()
}
FILE* f= fopenC'arbore.out".”wt");
if (nm!=n) fputs(“Neconex!\n”,f);
fprintf(f,”%d %d\n”, nf);
```

```
for (i=l; i<=nls; i++)
fprintf(f, ”%d %d\n”, lsa[i],lsb[i]);
fclose(f) ;
}
void main()
{
citeste();
rezolva();
scrie();
}
```

BIBLIOGRAFIE

[1] Cormen T.H., Leiserson E.C., Rivest R.R., *Introducere în algoritmi*, Editura Libris Agora, 2000 (traducere în limba română).

[2] Dahi O.J., Dijkstra E.W., Hoare C.A.R., *Structured Programing*, Academic Press, 1972.

[3] Frențiu M, Motogna S., Lazăr L, Prejmerean V., *Elaborarea algoritmilor*, Litografia Universității "Babeş Bolyai", Cluj Napoca, 1998.

Autor.
Ovidiu DOMŞA, Universitatea „l Decembrie 1918", Alba lulia Alba lulia, e-mail: ovidiu@uab.ro