Sortare Topologica

June 4, 2018

Student: Budure Sorin Marian

Calculatoare si tehnologia informatiei
(limba romana)

C.R 1.1 A

Anul 1

Introducere

In matematica, si mai precis in teoria grafurilor, un ***graf orientat*** (sau digraf) este un graf ale carui muchii au asociat un sens.
Grafurile aciclice orientate sunt grafuri orientate fara cicluri orientate.

Pentru un nod, numarul de capete adiacente unui nod este numit gradul interior al nodului si numarul de cozi adiacente unui nod este gradul exterior.

Sortarea Topologica

Dandu-se un graf orientat aciclic, sortarea topologica realizeaza o aranjare liniara a nodurilor in functie de muchiile dintre ele. Orientarea muchiilor corespunde unei relatii de ordine de la nodul sursa catre cel destinatie. Astfel, daca (u,v) este una dintre muchiile grafului, u trebuie sa apara inaintea lui v in insiruire. Daca graful ar fi ciclic, nu ar putea exista o astfel de insiruire (nu se poate stabili o ordine intre nodurile care alcatuiesc un ciclu).

Sortarea topologica poate fi vazuta si ca plasarea nodurilor de-a lungul unei linii orizontale astfel incat toate muchiile sa fie directionate de la stanga la dreapta.

Enuntul problemei

Alexandru, care este un copil mic, a primit cadou de ziua lui un joc Lego. Cu toate acestea, el nu a reuit sa construiasca jucaria Lego, deoarece exista multe piese. Alexandru a observat ca fiecare piesa este numerotata cu un numar de la 0 la 1000. Instruciunile jocului precizeaza piesele care trebuie asamblate inainte de fiecare piesa. Ajutati-l pe Alexandru sa-si construiasca jucaria Lego dezvoltand o aplicaie care determina ordinea corecta in care piesa Lego trebuie asamblata.

Pseudocod

In primul rand, vom genera un graf aciclic orientat pe care il vom memora intr-o matrice de adiacenta urmand ca pe aceasta matrice sa aplimacam 2 sortari topologice (prima bazata pe gradul intern al nodurilor si cea de-a doua pe DFS).

```
void(generare_grf(int x)
1.      time_t t;
2.      srand((unsigned) time(&t));
3.         pentru i de la 1 la x executa
4.            iterator1=rand() % 999;
5.            iterator2=rand() % 999;
6.              daca a[iterator1][iterator 2]=1 atunci
7.                i=i-1;
8.              altfel
9.                 a[iterator1][iterator2]=1
```

```
void dfc(int nod)
1.      intreg k;
2. viz[nod]=1;
3.  pentru k de la 1 la 1000 executa
4.      daca a[nod][k]==1 atunci
5.          daca viz[k]!=0 atunci
                dfc(k);
            altfel
                ac=1;
```

```
void sort_topologic(double a[1000][1000])
1. pentru i dela = la 1000 executa
2.      indeg[i]=0;
3.      flang[i]=0;
4.pentru i de la 0 la 1000 executa
5.  pentru j de la 0 la 1000 executa
6.      indeg[i]=indeg[i]+a[j][i];
7.afiseaza The topologic order is:
8.cat timp count<1000{
9.  pentru k de la 0 la 1000 executa{
10.     daca indeg[k]=0 & flang[k]=0 atunci
11.         {}afiseaza k+1;
12.         flag[k]=1;}
13.     pentru i de la 0 la 1000 executa
14.         daca a[i][k]=0 atunci
15.             indeg[k]=indeg[k]-1;
16.         }
17.     }
18.count=count+1;
```

```
int main()
1.  time_t t;
2.       srand((unsigned) time(&t));
3.       x=rand() %1000;
4.  afiseaza x;
5.repeta {
6.      ac=0;
7.      generare_graf(x)
8.      dfc(b);
9.      daca ac=1 atunci
10.         pentru i de la 0 la 1000 executa
11.             pentru j de la 0 la 1000 executa
12.                 a[i][j]=0;
13.}pana cand ac=1;
14.sort_topologic(a);
15.return 0;
```

Proiectarea Aplicatiei

Libraria contine un header graph.h care include toate functiile utilizate la rezolvarea problemei:
-void readData();
-void printTimpi();
-void explorare(int u);
-void DFS ();
-void topSort();

Fisierul graph.c contine sortarea topologica bazata pe parcurgerea grafului utilizand DFS(Depth First Search)
Parcurgerea in adancime (Depth-First Search - DFS) porneste de la un nod dat (nod de start), care este marcat ca fiind in curs de procesare. Se alege primul vecin nevizitat al acestui nod, se marcheaza si acesta ca fiind in curs de procesare, apoi si pentru acest vecin se cauta primul vecin nevizitat, si asa mai departe. In momentul in care nodul curent nu mai are vecini nevizitati, se marcheaza ca fiind deja procesat si se revine la nodul anterior. Pentru acest nod se cauta primul vecin nevizitat. Algoritmul se repeta pana cand toate nodurile grafului au fost procesate. In urma aplicarii algoritmului DFS asupra fiecarei componente conexe a grafului, se obtine pentru fiecare dintre acestea cate un arbore de acoperire (prin eliminarea muchiilor pe care nu le folosim la parcurgere). Pentru a putea reconstitui acest arbore, pastram pentru fiecare nod dat identitatea parintelui sau. Pentru fiecare nod se vor retine: timpul descoperirii, timpul finalizarii, parintele si culoarea. Algoritmul de explorare DFS nu este nici complet (cautare pe un subarbore infinit) nici optimal (nu gaseste nodul cu adancimea minima).

Dandu-se un graf orientat aciclic, sortarea topologica realizeaza o aranjare liniara nodurilor in functie de muchiile dintre ele. Orientarea muchiilor corespunde unei relatii de ordine de la nodul sursa catre cel destinatie. Astfel, daca (u,v) este una dintre muchiile grafului, u trebuie sa apara inaintea lui v in insiruire. Daca graful ar fi ciclic, nu ar putea exista o astfel de insiruire (nu se poate stabili o ordine intre nodurile care alcatuiesc un ciclu).

Cea de-a doua librarie contine un header graph.h care include toate functiile utilizate la rezolvarea problemei:
-void generare graf(x)
- void dfc(b)

- void sort topologic(a)
Functia generare graf si functia dfc ne genereaza un graf acciclic astfel:
- se porneste de la premisa ca exista ciclu
- se face DF dintr-un nod oarecare
- daca, la un moment dat, se ajunge intr-un nod care are ca vecin un nod prin care s-a trecut inainte , atunci exista un ciclu
Functia sort topoligic contine sortarea topologica bazata pe gradul intern al nodurilor

Source Code

```
//-----------------------------graph.h------------------------------

#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

 void generare_graf(x);
 void dfc(b);
 void sort_topologic(a);

#endif // GRAPH_H_INCLUDED
```

```
//-----------------------------graph.c------------------------------

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "graph.h"
double a[1000][1000];
int i,j,n,ac,x,k,iterator1,iterator2,viz[1000],nod,b=1;

void generare_graf(int x)
{
    time_t t;
    srand((unsigned) time(&t));

    for(i=1;i<=x;i++)
        {
         iterator1=rand() % 999;
         iterator2=rand() % 999;
         if(a[iterator1][iterator2]==1)
            i--;
         else
            a[iterator1][iterator2]=1;
        }
}
```

```
void dfc(int nod){
int k;
viz[nod]=1;
    for(k=1;k<1000;k++)
    {
        if(a[nod][k])
            if(!viz[k])
                dfc(k);
            else
                ac=1;
    }
}
void sort_topologic(double a[1000][1000])
{
    double indeg[1000]={0},flag[1000]={0},count=0;
    int k;

    for(i=0;i<1000;i++){
        indeg[i]=0;
        flag[i]=0;
    }

    for(i=0;i<1000;i++)
        for(j=0;j<1000;j++)
            indeg[i]=indeg[i]+a[j][i];

    printf("\nThe topological order is:");

    while(count<1000){
        for(k=0;k<1000;k++){
            if((indeg[k]==0) && (flag[k]==0)){
                printf("%d ",(k+1));
                flag [k]=1;
            }

            for(i=0;i<1000;i++){
                if(a[i][k]==1)
                    indeg[k]--;
            }
        }

        count++;

    }
}




//--------------------------------main.c-------------------------------
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
double a[1000][1000];
int i,j,n,ac,x,k,iterator1,iterator2,viz[1000],nod,b=1;

int main()
{     time_t t;
       srand((unsigned) time(&t));
       x=rand() %1000;
       printf("%d",x);

    do
    {
        ac=0;
        generare_graf(x);
        dfc(b);
        if(ac==1)
            for(i=0;i<1000;i++)
                for(j=0;j<1000;j++)
                    a[i][j]=0;
    }while(ac==1);

    sort_topologic(a);

    return 0;
}

//---------------------------------graph.c--------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "graph.h"

int n;

void readData()
{
FILE * f;
int i, j;

f = fopen( "in2.txt","r" );
assert( f );

fscanf( f, "%d", &n );
graf = ( struct Nod * ) malloc ( n * sizeof( struct Nod ) );

for ( i = 0; i < n; ++i ) {
graf[i].val = i;
fscanf( f, "%d", &graf[i].nrVecini );
```

```
graf[i].vecini = ( int * ) malloc ( graf[i].nrVecini * sizeof ( int ) );
for ( j = 0; j < graf[i].nrVecini; ++j )
fscanf( f, "%d", &graf[i].vecini[j] );
graf[i].parinte = -1;
graf[i].culoare = 0;
graf[i].tDescoperire = -1;
graf[i].tFinal = -1;
}
fclose( f );
}

/* afisarea timpilor din alg DFS*/
void printTimpi()
{
int i;
printf( "\n" );
for ( i = 0; i < n; ++i )
printf ( "Nod %d: descoperit la t=%d, explorat la t=%d \n", i,
    graf[i].tDescoperire, graf[i].tFinal );
printf( "\n" );
}

int timp = 0;

/* parcurgere DFS a grafului */
void explorare(int u) {

graf[u].tDescoperire = timp++;
graf[u].culoare = 1;
int j;
for(j = 0 ; j < graf[u].nrVecini ; j++) {
int x = graf[u].vecini[j];
if(graf[x].culoare ==0) {
graf[x].parinte = u;
explorare(x);
}
}
graf[u].culoare = 2;
graf[u].tFinal = timp++;

}

void DFS ()
{
int i;
for (i=0; i < n ; i++)
if(graf[i].culoare == 0)
explorare(i);
}
```

```
/* sortare topologica */
void topSort()
{
int ok = 0,i,j;
struct Nod aux;
while (ok == 0) {
ok = 1;
for(i=0; i < n-1;i++)
if(graf[i].tFinal < graf[i+1].tFinal) {
ok = 0; aux = graf[i]; graf[i]=graf[i+1]; graf[i+1]=aux; }
}
}
//----------------------------------graph.h--------------------------------
#ifndef GRAPH_H_INCLUDED
#define GRAPH_H_INCLUDED

void readData();
void printTimpi();
void explorare(int u);
void DFS ();
void topSort();
struct Nod
{
int val;
int nrVecini;
int * vecini;
int culoare;
int parinte;
int tDescoperire;
int tFinal;
};
struct Nod * graf;

#endif // GRAPH_H_INCLUDED
//---------------------------------main.c---------------------------------
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "graph.h"


/* citeste lista de adiacenta din fisier */
int n;

int main(void){
int i;

//incarca graful din fisier
```

```
readData();

//parcurgere BFS
DFS();

//afisarea timpilor
printTimpi();

//sortare topologica
topSort();
printf( "Sortare topologoca: " );
for ( i = 0; i < n; ++i )
printf ( "%d ", graf[i].val );
printf("\n");

for( i = 0; i < graf[i].nrVecini; ++i )
free(graf[i].vecini);
free(graf);
return 0;
}
```

Concluzii

Lucrand la acest proiect am reusit sa-mi imbunatatesc abilitatile de proiectare si implementare a algoritmilor bazati pe parcurgerea grafurilor orientate cat si pe afisarea nodurilor in functie de muchiile dintre ele (Sortare Topologica). De asemenea, lucrand un timp indelungat cu grafuri am inteles aplicabilitatea lor in viata de zi cu zi, spre exemplu: Retele de calculatoare (ex: stabilirea unei topologii fara bucle),pagini Web (ex: Google PageRank [1]),retele sociale (ex: calcul centralitate [2]),harti cu drumuri (ex: drum minim),modelare grafica (ex: prefuse [3], graph-cut [4]).

Referinte

1)https://ocw.cs.pub.ro/courses/pa/laboratoare/laborator-07

2)https://the-graph-i-75.wikispaces.com/Algoritmii+fundamentali

3)http://boomblebee.blogspot.com/2010/04/dfs-si-sortare-topologica.html

4)https://infoarena.ro/problema/sortaret

5)http://www.sharelatex.com