Ciclu eulerian

June 5, 2018

Student: Croitoru Mihăiţă Cosmin

Calculatoare şi Tehnologia Informaţiei

Grupa CR1.1 B

Anul 1

Introducere

Algoritmul lui Fleury este un algoritm elegant, dar ineficient, care datează din 1883.Se dă un graf despre care se ştie că are toate muchiile în aceeaşi componentă şi cel mult două noduri de grad impar. Algoritmul porneşte de la un nod de grad impar, sau, dacă graficul nu are niciunul, începe cu un nod arbitrar ales. La fiecare pas se alege urmatoarea muchie din drum ca fiind una a carei ştergere nu ar face graful sa îşi piardă conexitatea, iar dacă nu există nicio astfel de muchie, se alege muchia răsmasă pentru nodul curent. Apoi se trece la celălalt capăt al muchiei, iar muchia aleasă anterior se şterge. La sfarşitul algoritmului nu mai există muchii, şi secvena în care au fost alese muchiile formează un ciclu eulerian unde graful nu are noduri de grad impar, sau un drum eulerian, dacă există exact două noduri de grad impar.

Un graf neorientat are un ciclu eulerian dacă şi numai dacă fiecare nod are grad par, şi toate nodurile cu grad nenul aparţin unei singure componente conexe(d.)

Un graf neorientat poate fi descompus în cicluri cu muchii disjuncte, dacă şi numai dacă toate nodurile sale au grad par. Deci, un graf are un ciclu eulerian dacă şi numai dacă el poate fi descompus n cicluri cu muchii disjuncte şi nodurile sale de grad nenul aparţin unei singure componente conexe.

Un graf neorientat are un drum eulerian dacă şi numai dacă exact zero sau două noduri au grad impar, şi toate nodurile cu grad nenul aparin unei singure componente conexe.

Cerinta problemei

Eulerian cycles. Implement two algorithms to determine all Eulerian cycles in undirected graphs, e.g., Rosenstiehl Fleury.

Pseudocod

```
grad(int nod)
1.  pozitie <- index(nod)
2.  pentru iterator_2 de la 0 la n executa
3.    daca MatriceAdiacenta[pozitie][iterator_2] = 'd' atunci
4.      grad_1++
5.  return grad_1
```

```
radacina()
1.  pentru iterator_1 de la 0 la n executa
2.    daca grad(b[iterator]) % 2 != 0 atunci
3.      numarator++
4.        current <- iterator_1
5.  daca numarator != 0 && numarator != 2 atunci
6.    return 0
7.  altfel return b[current]
```

```
index(int nod)
1.  cat timp nod != b[iterator_3] execcuta
2.    iterator_3++
3.  return iterator_3
```

```
ultimul_nod(int nod)
1.  pentru iterator_1 de la 0 la n executa
2.    suma_gradelor <- suma_gradelor + grad(b[iterator_1])
3.  daca suma_gradelor = 2 atunci
4.    return 1
5.  altfel
6.    return 0
```

```
nodul_urmator(int nod)
1.  pozitie <- index(nod)
2.  pentru iterator_1 de la 0 la n executa
3.    daca MatriceAdiacenta[pozitie][iterator_1] = 'd' atunci
4.      daca !are_o_muchie(b[iterator_1] atunci
5.        return b[iterator_1]
6.      altfel
7.        daca ultimul_nod(b[iterator_1] atunci
```

```
8.          return b[iterator_1]
```

```
are_o_muchie(int nod)
1.  daca grad(nod) = 1 atunci
2.    return 1
3.  altfel
4.    return 0
```

```
complet()
1.  pentru iterator_1 de la 0 la n executa
2.    daca grad(b[iterator_1]) > 0 atunci
3.      return 0
4.  return 1
```

```
stergere_muchie(radacina1, eNod)
1.  pozitia_1 <- index(radacina1)
2.  pozitia_2 <- index(eNod)
3.  MatriceAdiacenta[pozitia_1][pozitia_2] <- 'n'
4.  MatriceAdiacenta[pozitia_2][pozitia_1] <- 'n'
```

```
euler(int radacina1)
1.  cat timp Complet() != 0 executa
2.    eNod <- nodul_urmator(radacina1)
3.    calea_finala[fp++] <- eNod
4.    stergere_muchie(radacina1,eNod)
5.    radacina1 <- eNod
```

Proiectarea Aplicatiei

Functiile utilizate la rezolvarea problemei sunt:
—void afisare()
—int grad(int nod)
—int radacina()
—int index(int nod)
—int ultimul-nod(int nod)
—int nodul-urmator(int nod)
—int are-o-muchie(int nod)
—int Complet()
—void stergere-muchie(radacina1,eNod)
—void euler(int radacina1)
—void afisare2()

Functia void afisare():
— Afişează calea sau circuitul eulerian.

Functia int grad(int nod):
— Este folosită pentru a obtine gradul nodului
— Gradul nodului este numărul de muchii care sunt conectate la aceasta

Functia int radacina():
— Pentru a atribui rădăcina graficului
— Condiia 1: Dacă toate nodurile au un grad egal, ar trebui să existe un Circuit / Ciclu euler
— Putem porni calea de la orice nod
— Condiia 2: Dacă exact 2 noduri au un grad impar, ar trebui să existe o cale euler.
— Trebuie să pornim de la un nod care are un grad impar
— Condiia 3: Dacă mai mult de 2 noduri sau exact un nod au un grad impar, calea / circuitul euler nu este posibilă

Functia int index(int nod):
— Pentru a obine indicele curent al nodului in matricea b [] a nodului

Functia int ultimul-nod(int nod):
— Returnează 0 dacă numărul ”y” este egal cu 2 în matrice

— Indică faptul că nu există alte muchii pe care să le vizitai n acest graf
— Deja au fost vizitate toate muchiile

Functia int Complet():
— Dacă nu există "y" în matrice inseamn că am vizitat toate varfurile si muchiile

Functia int are-o-muchie():
— Dacă gradul de varf este egal cu 1 inseamnă câ varful are o muchie

Functia void euler(int radacina1):
— Este folosită pentru a găsi circuitul / calea Euler şi a-l depozita in matricea finalPath []

Source Code

```
#include<stdio.h>
#include<conio.h>
int n;
int calea_finala[100];
char MatriceAdiacenta[100][100];
int numarator = 0;
int fp = 0;
int b[100];
```

```
//----------------------------Functii-------------------------------


void afisare(){
     int iterator_1;
     for(iterator_1 = 0; iterator_1 < fp; iterator_1++){
       printf("%d->",calea_finala[iterator_1]);
     }
}

int grad(int nod){
    int iterator_2, grad_1 = 0;
    int pozitie = Index(nod);
    for(iterator_2 = 0; iterator_2 < n ; iterator_2++){
      if(MatriceAdiacenta[pozitie][iterator_2] == 'd'){
        grad_1++;
```

```
      }
    }
    return grad_1;
}

/* Conditia : Daca toate nodurile au gradul egal, atunci exita circuit
    sau ciclu eulerian
Putem incepe calea din orice nod*/

int radacina(){

     int iterator_1, current = 1;
     for(iterator_1 = 0; iterator_1 < n; iterator_1++){
        if(grad(b[iterator_1]) %2 != 0){
           numarator++;
           current = iterator_1;
        }
     }
     if(numarator != 0 && numarator != 2){
        return 0;
     }
     else return b[current];
}


int Index(int nod){
    int iterator_3 = 0;
    while(nod != b[iterator_3]){
        iterator_3++;
    }
    return iterator_3;
}

int ultimul_nod(int nod){Pentru a gsi circuitul / calea Euler i a-l
     depozita n matricea finalPath []
    int iterator_1 = 0;
    int suma_gradelor = 0;
    for(iterator_1 = 0; iterator_1 < n; iterator_1++){
     suma_gradelor = suma_gradelor + grad(b[iterator_1]);
    }
    if(suma_gradelor == 2)
      return 1;
    else
      return 0;
}

int nodul_urmator(int nod){
    int iterator_1 = 0;
    int pozitie = Index(nod);
```

```
    for(iterator_1 = 0; iterator_1 < n; iterator_1++){
      if(MatriceAdiacenta[pozitie][iterator_1] == 'd'){
        if(!are_o_muchie(b[iterator_1])){
          return b[iterator_1];
        }
        else{
            if(ultimul_nod(b[iterator_1]))
            return b[iterator_1];
        }
      }
    }
    return -1;
}

int are_o_muchie(int nod){
    if(grad(nod)==1)
      return 1;
    else
      return 0;
}

int Complet(){
    int iterator_1 = 0;
    for(iterator_1 = 0; iterator_1 < n; iterator_1++){
        if(grad(b[iterator_1])>0)
             return 0;
    }
    return 1;
}

void stergere_muchie(radacina1,eNod){
     int pozitia_1 = 0,pozitia_2 = 0;
     pozitia_1=Index(radacina1);
     pozitia_2=Index(eNod);
     MatriceAdiacenta[pozitia_1][pozitia_2] = 'n';
     MatriceAdiacenta[pozitia_2][pozitia_1] = 'n';
}

// Cauta circuitul/calea euler si o stocheaza in caleafinala[]
void Euler(int radacina1){
     int eNod;
     while(!Complet()){
        eNod=nodul_urmator(radacina1); //adauga nodul urmator
        calea_finala[fp++]=eNod;  //adauga nodul in calea euleriana
        stergere_muchie(radacina1,eNod); //sterge muchia
        radacina1=eNod; //schimba radacina
     }
}

void afisare_2(){
```

```
     int iterator_1;
      for( iterator_1 = 0; iterator_1 < n; iterator_1++){
        printf("%d ",b[iterator_1]);//stocheaza nodrueile in b
    }
}

//--------------------------------Functia
    main()--------------------------------


int main(){
    char v;
    int iterator_1, iterator_2, l;
    printf("Introduceti numarul de noduri din graf: \n");
    scanf("%d",&n);
    printf("Introduceti valorile nodului\n");
    for( iterator_1 = 0; iterator_1 < n; iterator_1++){
     scanf("%d",&b[iterator_1]);//stocheaza nodurile in in b[]
    }



    printf("\nDaca exista muchie intre doua noduri introduceti d, altfel
        introduceti n !\n");
    for( iterator_1 = 0; iterator_1 < n; iterator_1++){
           printf(" %d ",b[iterator_1]);
        }
    for( iterator_1 = 0; iterator_1 < n; iterator_1++){
         printf("\n%d ",b[iterator_1]);
         for( iterator_2 = 0; iterator_2 < n; iterator_2++){
             printf("%c ",v=getch());
             MatriceAdiacenta[iterator_1][iterator_2] = v;
         }
         printf("\n\n");
    }

    //caleafinala[] va returna 0 daca calea/circcuitul euler nu este
        posibil
    //otherwise it will return array index of any node as root
    int radacina1;
    if(radacina1=radacina()){
      if(numarator){
        printf("Available Euler Path is\n");
      }
      else printf("Ciclul eulerian valabil este:\n");
      calea_finala[fp++] = radacina1;
      Euler(radacina1);
      afisare();
    }
    else printf("Ciclul sau calea euleriana nu este valabil!\n");
```

```
    getch();
    return 0;
}
```

Experimente si Rezultate

```
Exemplul 1:

Introduce\c{t}i numarul de noduri din graf:
6
Introduce\c{t}i valorile nodului
0
1
2
3
4
5

Dac\u{a} exista muchie \^{i}ntre doua noduri introduce\c{t}i d, altfel
    introduce\c{t}i n !
 0  1  2  3  4  5
0 n d n d n n


1 d n d n d d


2 n d n d d d


3 d n d n n n


4 n d d n n n


5 n d d n n n

Output:
Ciclul eulerian valabil este:
1->0->3->2->1->4->2->5->1->
```

```
Exemplul 2

Introduce\c{t}i num\u{a}rul de noduri din graf:
5
Introduce\c{t}i valorile nodului:
1
2
3
4
5

Dac\u{a} exista muchie \^{i}ntre doua noduri introduce\c{t}i d, altfel
    introduce\c{t}i n !
 1  2  3  4  5
1 n d d d n


2 d n d n d


3 d d n d d


4 d n d n d


5 n d d d n



Output:
   Ciclul sau calea euleriana nu este valabil!
```

Concluzii

Lucrând la acest proiect am reuşit să-mi Îmbunătatesc abilitatile de proiectare şi implementare a algoritmilor bazati pe parurgerea grafurilor. De asemenea, in timpul lucrări mi-am fixat şi teoria cu privire la ciclurile eulerine, dar şi apicabilitatea acestora în practică. Drumurile euleriene sunt utilizate în bioinformatică pentru a reconstrui secvene de ADN(d) din fragmente. Ele sunt utilizateşi în proiectarea circuitelor CMOS pentru a găsi ordonări optime ale porilor logice.

Referinte

1)https://ro.wikipedia.org/wiki/Drum$_e ulerian$

2)$http://web.info.uvt.ro/\ mmarin/lectures/TGC/Curs10.pdf$

3)$https://infoarena.ro/problema/ciclueuler$

4)$https://www.geeksforgeeks.org/eulerian-path-undirected-graph/$