Grafuri tratate matematic si informatic

Universitatea "Constantin Brâncuşi" Tg-Jiu

Secția Ingineria Sistemelor, anul I

Profesor coordonator: Adrian Runceanu

Autori: 1. Mihai Alexandru Ciortan

2. Ionut Popescu

3. Lavinia Bogdea

4. Olivia Maria

CUPRINS:

1. Introducere in grafuri orientate

1. DEFINITII.

Definitie 1:

Se numeste graf orientat, o pereche ordonata de multimi (X,U), unde:

-X este o multime finita si nevida de elemente numite noduri;

-U este o multime de perchi ordonate de cate doua elemente din X, numite arce.

U={[x,y] /x,y apartin X}

Observatie: Muchia [x,y] NU este identica cu [y,x].

Definitia 2: Fie G=(X,U) un graf orientat si [x,y] un arc. Spunem ca x este extremitate initiala iar y extremitate finala a arcului.

Spunem ca varfurile x si y sunt adiacente ,si ca varful x sau y , este incident cu muchia [x,y].

Se numesc arce incidente doua arce u_i si u_j care au o extremitate comuna, nodul x_k.

Definitie 3: Fie G=(X,U) un graf orientat si x un nod.

Gradul intern al unui arc x, notat $d^-(x)$, reprezinta numarul arcelor care intra in nodul x

Gradul extern al unui arc x, notat $d^+(x)$, reprezinta numarul arcelor care ies in nodul x

Definitia 4: Graful Gc=(X, V) se numeste graf complementar al grafului G=(X,U) daca are proprietatea ca doua noduri sunt adiacenmte in Gc, numai daca NU sunt adiacente in G (U intersectat cu V este multimea vida)

Definitia 5: Un graf orientat in care, intre oricare doua noduri exista un singur arc si numai unul, se numeste graf turneu. Arcul dintre doua noduri poate avea oricare din cele doua orientari. Graful turneu este un graf complet.

2. REPREZENTAREA GRAFURILOR ORIENTATE IN MEMORIE

Fie G= (X,U) un graf orientat cu n nodurri si m arce.

2.1 Matricea de adiacenta : matrice binara de ordin n

a[i,j] = 1 , daca exista arc (i,j), i<>j;

0 , in caz contrar

Proprietati:

1)Aii=0, i=1 to n (elementele de pe diagonala principala sa fie egale cu 0)

2)Matricea NU este simetrica fata de diagonala principala

3)Matricea este complexitatea spatiu (n*n) de ordinul n patrat.

O(n*n)

4) Suma elementelor de pe linia i este egala cu gradul extern al nodului i.

5) Suma elementelor de pe coloana i este egala cu gradul intern al nodului i.

2.2.Vector de muchii

Vom reprezenta graful prin intermediul unui vector cu elemente de tip inregistrare, fiecare inregistrare avand 2 campuri:extremitatile muchiilor.

Proprietati: Complexitatea spatiu a acestei reprezentari este de ordinul O(2m).

2.3. Liste de adiacenta

O(n+2m)

Proprietate: Complexitatea spatiu a acestei reprezentari este de ordinul O(n+2m)

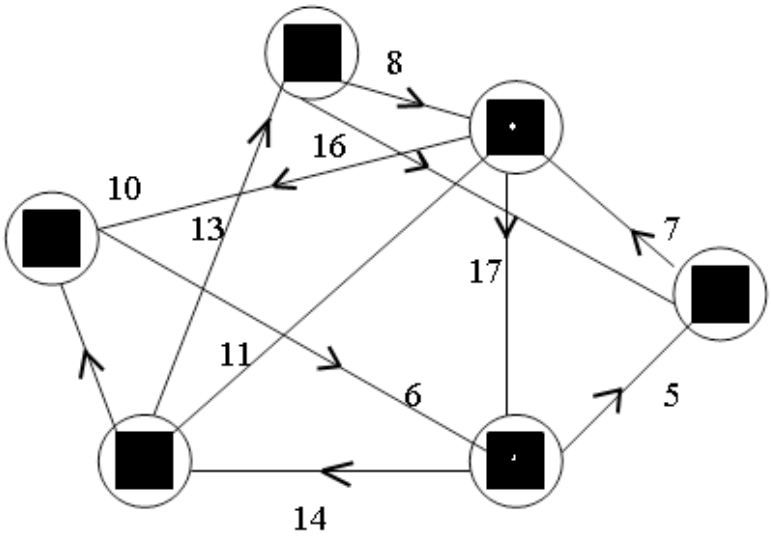
2. Algoritmul lui Dijkstra – prezentară generală

Se consideră un graf orientat, conex cu N noduri. Graful este dat prin matricea costurilor A.

Se consideră un nod inițial R. Se cer lungimile drumurilor minime de la R la celelalte noduri ale grafului, precum şi nodurile prin care trec aceste drumuri.

Problema are numeroase aplicații practice. Să presupunem că dispunem de o hartă cu oraşele unei țări. Acestea sunt unite prin şosele. Oraşele formează nodurile grafului, iar şoselele – arcele acestuia. Algoritmul furnizează drumurile optime de la un oraş inițial la celelalte oraşe. Este interesant de observat faptul că, spre deosebire de alți algoritmi, acesta furnizează şi nodurile prin care trec drumurile optime, nu numai lungimile acestora.

Exemplu: Se consideră graful din figură:



Algoritmul selectează nodurile grafului, unul câte unul, în ordinea crescătoare a costului drumului de la nodul R la ele, într-o mulțime S, care inițial conține numai nodul R. În felul acesta ne încadrăm în strategia generală GREEDY. În procesul de prelucrare se folosesc trei vectori: D,S şi T.

Vectorul D este vectorul costurilor de la nodul R la celelalte noduri ale grafului. Prin D(I), unde I∈{1..N}, se înțelege costul drumului găsit la un moment dat, între nodul R şi nodul I.

Vectorul T indică drumurile găsite între nodul R şi celelalte noduri ale grafului.

Vectorul S (vector caracteristic) indică mulțimea S a nodurilor selectate. S(I)=0 dacă nodul I este neselectat şi S(I)=1 dacă nodul I este selectat.

Prezentarea algoritmului.

P1) Nodul R este adăugat mulțimii S inițial vidă (S(R)=1);

- costurile drumurilor de la R la fiecare nod al grafului se preiau în vectorul D de pe linia R a matricii A;

- pentru toate nodurile I având un cost al drumului de la R la ele finit, se pune T(I)=R.

P2) Se execută de n-1 ori secvența

- printre nodurile neselectate se caută cel aflat la distanța minimă față de R şi se selectează adăugându-l mulțimii S;

- pentru fiecare din nodurile neselectate se actualizează în D costul drumurilor de la R la el, utilizând ca nod intermediar nodul selectat, procedând în felul următor: se compară distanța existentă în vectorul D cu suma dintre distanța existentă în D pentru nodul selectat şi distanța de la nodul selectat la nodul pentru care se face actualizarea distanței (preluată din A), iar în cazul în care suma este mai mică, elementul din D corespunzător nodului pentru care se face actualizarea capătă ca valoare această sumă şi elementul din T corespuzător aceluiaşi nod ia valoarea nodului selectat (drumul trece prin acest nod). Presupunând că a fost selectat nodul K, se actualizează distanța pentru nodul L şi se compară D(K)+A(K,L) cu D(L).

P3) Pentru fiecare nod al grafului, cu excepția lui R, se trasează drumul de la R la el.

3. Implementare clasică Dijkstra

```
// sa se implemeteze un program care aplica algoritmul DIJKSTRA

//pentru determinarea drumurilor de lungime minima

//nodul de plecare e 1

5

1 2 1

1 3 9

1 5 3

2 4 3

2 3 7

4 3 2

4 1 1

5 2 4

5 4 2

=>drumul minim de la 1 la 3 e 6 si trece prin nodurile 1 2 4 3
```

```
#include "stdafx.h"
#include <fstream>
#include <iostream>
#include <math.h>
using namespace std;
const float Pinfinit=1.e20;
float A[50][50],D[50],min;
int S[50],T[50],n,I,j,r,poz;
void Citire_cost (char Nume_fis[20],float A[50][50],int& n)
{
  int I,j;
  float c;
  fstream f("graf.txt",ios::in);
  f>>n;
  for (I=1;I<=n;I++)
    for (j=1;j<=n;j++)
     if (I==j) A[I][j]=0;
     else A[I][j]=Pinfinit;
while (f>>I>>j>>c) A[I][j]=c;
f.close();
}
void drum(int I)
{
if (T[I]) drum(T[I]);
cout<<I<<" ";
}
main()
```

```
{ float min;
Citire_cost("Graf.txt",A,n);
cout<<"Introduceti nodul de pornire "<<endl;
cout<<"r=";cin>>r;S[r]=1;
for (I=1;I<=n;I++)
{
D[I]=A[r][I];
if (I!=r)
 if (D[I]<Pinfinit) T[I]=r;}
for (I=1;I<=n;I++)
      {
  min=Pinfinit;
for(j=1;j<=n;j++)
if (S[j]==0)
  if (D[j]<min)
     {
min=D[j];
poz=j;
     }
S[poz]=1;
for (j=1;j<=n;j++)
 if (S[j]==0)
   if (D[j]>D[poz]+A[poz][j])
         {
D[j]=D[poz]+A[poz][j];
T[j]=poz;
        }
```

```
        }
for (I=1;I<=n;I++)
 if (I!=r)
   if(T[I])
       {
cout<<"distanta de la "<<r<<" la "<<I<<" este "<<D[I]<<endl;
drum(I);
cout<<endl;
      }
else
  cout<<"nu exista drum de la "<<r<<" la "<<I<<endl;
}
```

4. Dijkstra implementat pe heap-uri

```
#include<cstdio>
#include<cstring>
#define NMAX 100001
#define INFI 1 << 30
#define CL(x) memset(x, 0, sizeof(x))

using namespace std;

typedef struct Graph
{
  long node; long cost;
  Graph *next;
```

```
};


FILE *fin, *fout;


long Heap[ NMAX ], NodHeap[ NMAX ], Poz[ NMAX ], Minim[ NMAX ],
Father[ NMAX ], Complet[ NMAX ];

long Nmax, N, M, Final;


int VALID_UNU, VALID_DOI;


Graph *List[ NMAX ];


inline void swap(long &V1, long &V2)  // Functie inline de inversare a
doua valori in heap

{

  long t = V1;

  V1 = V2;

  V2 = t;

}


void ReadSel1() //Citeste nodurile pentru cazul in care s-a ales
graful 1

{

  fin = fopen("graf1.in", "r");


  long X, Y, Z;

  Graph *adr;
```

```
  if(!fin)
  {
    printf("Fisierul necesar nu exista! Program incheiat!\n");
    VALID_DOI = 0;
    VALID_UNU = 0;

  }
  else
  {
    fscanf(fin, "%ld", &N);
    fscanf(fin, "%ld", &M);

    for(long i = 1; i <= M; ++i)
    {
      fscanf(fin, "%ld %ld %ld", &X, &Y, &Z);

      adr = new Graph;           ///adauga in lista de adiacenta muchia
citita
      adr -> node = Y;
      adr -> cost = Z;
      adr -> next = List[ X ];
      List[ X ] = adr;
    }
  }
}
```

```
void ReadSel2() //Citeste nodurile pentru cazul in care s-a ales
graful 2
{
  fin = fopen("graf2.in", "r");

  long X, Y, Z;
  Graph *adr;

  if(!fin)
  {
    printf("Fisierul necesar nu exista! Program incheiat!\n");
    VALID_DOI = 0;
    VALID_UNU = 0;

  }
  else
  {
    fscanf(fin, "%ld", &N);
    fscanf(fin, "%ld", &M);

    for(long i = 1; i <= M; ++i)
    {
      fscanf(fin, "%ld %ld %ld", &X, &Y, &Z);

      adr = new Graph;           ///adauga in lista de adiacenta muchia
citita
      adr -> node = Y;
```

```
      adr -> cost = Z;
      adr -> next = List[ X ];
      List[ X ] = adr;
    }
  }
}

void ReadSel3() //Citeste nodurile pentru cazul in care s-a ales
optiunea 3
{
  printf("Numarul de noduri: ");
  scanf("%ld", &N);
  printf("\nNumarul de muchii: ");
  scanf("%ld", &M);
  printf("\nIntroduceti muchiile in formatul A B C (A - nod de iesire,
B - nod de intrare, C - costul muchiei ) \n");

  long X, Y, Z;
  Graph *adr;

  for(long i = 1; i <= M; ++i)
  {
    scanf("%ld %ld %ld", &X, &Y, &Z);

    adr = new Graph;        ///adauga in lista de adiacenta muchia
citita
    adr -> node = Y;
    adr -> cost = Z;
```

```
    adr -> next = List[ X ];
    List[ X ] = adr;
  }
}

void ReadSel4() //Citeste nodurile pentru cazul in care s-a ales
optiunea 4
{
  char File[ 255 ];
  int MAXIM = 0;
  Graph *adr;
  long X, Y, Z;

  printf("Introdu numele fisierului ce contine datele in formatul
urmator:\nN M\nA1 B1 C1\nA2 B2 C2\n...\nAm Bm Cm\n\n");
  printf("unde N - numarul de noduri, M - numarul de muchii, A, B, C -
exista o muchie de la A la B, avand costul C\n");
  printf("Numele: ");
  scanf("%s", &File);
  fin = fopen(File, "r");

  while(!fin)
  {
    printf("Fisierul introdus nu exista! Incercati din nou (%d
incercari ramase) : ", 2 - MAXIM);
    scanf("%s", &File);
    ++MAXIM;
    fin = fopen(File, "r");
```

```
    if(MAXIM == 3)
    {
      VALID_UNU = 1;
      VALID_DOI = 1;
    };
    break;
  }

  fscanf(fin, "%ld", &N);
  fscanf(fin, "%ld", &M);

  for(long i = 1; i <= M; ++i)
  {
    fscanf(fin, "%ld %ld %ld", &X, &Y, &Z);

    adr = new Graph;        ///adauga in lista de adiacenta muchia
citita
    adr -> node = Y;
    adr -> cost = Z;
    adr -> next = List[ X ];
    List[ X ] = adr;
  }

  fclose(fin);
}
```

```
void UpHeap(long p)   ///urca un nod cu o valoare mai mica de la
pozitia curenta p, catre varful heapului
{
  while( p > 1)
  {
    if(Heap[ p / 2 ] > Heap[ p ])
    {
      swap( Heap[ p / 2 ], Heap[ p ] );
      swap( NodHeap[ p / 2 ], NodHeap[ p ]);
      swap( Poz[ NodHeap[ p / 2 ] ], Poz[ NodHeap[ p ] ]);

      p /=2;
    }
    else return;
  }
}

void DownHeap(long p)      //coboara nodul din pozitia p a heapului,
daca valoarea lui este mai mare decat fii sai
{
  long P;

  while(p <= Nmax)
  {
    if(2 * p <= Nmax)
    {
      P = 2 * p;
```

```
      if(P + 1 <= Nmax && Heap[ P + 1 ] < Heap[ P ])   //determin fiul
maxim
        ++P;

      if(Heap[ P ] < Heap[ p ])
      {
        swap( Heap[ P ], Heap[ p ] );
        swap( NodHeap[ P ], NodHeap[ p ]);
        swap( Poz[ NodHeap[ P ] ], Poz[ NodHeap[ p ] ]);

     p = P;
      }
      else return;
    }
    else return;
  }
}

void InitMin()              //initializeaza valorile minime ale
drumurilor
{
  for(long i = 1; i <= N; ++i)
  {
    Minim[ i ] = INFI;
    Poz[ i ] = -1;
  }
```

```
}

void SolveFunction(long Start)   //Rezolva drumul de cost minim,
                 //pornind de la nodul Start
{
  CL(Heap); CL(NodHeap); CL(Minim); CL(Complet); CL( Poz ); CL( Father
);

  Graph *adr;
  long Actual, T;

  InitMin();

  Heap[ 1 ] = 0, NodHeap[ 1 ] = Start, Poz[ Start ] = 1, Father[ Start
] = 0;
  Nmax = 1;

  while( Nmax > 0 )
  {
    //extrag maximul din heap - drum stabil pentru nodul respectiv

    Actual = NodHeap[ 1 ], Minim[ Actual ] = Heap[ 1 ];
    Heap[ 1 ] = Heap[ Nmax ], NodHeap[ 1 ] = NodHeap[ Nmax ], Poz[
NodHeap[ 1 ] ] = 1;

    --Nmax;

    DownHeap(1);
```

```
    adr = List[ Actual ];


    //updatez heapul cu valorile drumurilor care se modifica, sau cu
cele care de-abia acum se aloca


    while( adr != NULL)
    {
      if( Minim[ adr -> node ] == INFI)
      {
     if( Poz[ adr -> node ] == -1 )         //nod neatins
     {
       Heap[ ++Nmax ] = Minim[ Actual ] + adr -> cost;
       Father[ adr -> node ] = Actual;
       NodHeap[ Nmax ] = adr -> node, Poz[ NodHeap[ Nmax ] ] = Nmax;


       UpHeap( Nmax );
     }
     else
        if( Heap[ Poz[ adr -> node ] ] > (T = Minim[ Actual ] + adr ->
cost))     //drum mai scurt
     {
          Heap[ Poz[ adr -> node ] ] = T;
       Father[ adr -> node ] = Actual;


       UpHeap( Poz[ adr -> node ] );
     }
```

```
    }

    adr = adr -> next;
  }
}

//afisez drumul pe care il parcurgi de la Start catre Final

long NPoz, Nod;

if(Minim[ Final ] != INFI && Minim[ Final ])
{
  printf("Costul minim al drumului este: %ld\n", Minim[ Final ]);
  printf("Drumul de cost minim este: ");

  Nod = Final;
  NPoz = 0;

  while(Father[ Nod ] != Final && Father[ Nod ] != 0)
  {
    Complet[ ++NPoz ] = Nod;
    Nod = Father[ Nod ];
  }

  if(NPoz)
  {
    printf("%ld -> ", Start);
```

```
      while(NPoz > 0)
      {
        if(NPoz > 1) printf("%ld -> ", Complet[ NPoz ]);
      else printf("%ld\n", Complet[ NPoz ]);
      --NPoz;
      }
    }
  }
  else
    printf("Nu se poate ajunge de la nodul %ld la nodul %ld\n", Start,
Final);

  printf( "\n");
}

int main()
{
  /* Ecran de Start-UP*/

  VALID_UNU = 1;
  long St;
  int Optiune, OP;

  while(VALID_UNU){

  printf("\n\n\nPROIECT: Grafuri tratate matematic si informatic\n");
  printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n\n");
```

```
  printf("Alegeti una din optiunile de mai jos:\n");


  printf("1. Foloseste primul graf predefinit (N = 6, M = 10) \n");
  printf("2. Foloseste al doilea graf predefinit (N = 10421, M =
21313) \n");
  printf("3. Introdu de la tastatura un nou graf \n");
  printf("4. Introdu numele unui fisier care contine un graf nou \n");
  printf("5. Paraseste aplicatia \n\n\n");


  printf("Optiunea = ");
  scanf("%d", &Optiune);
  printf("\n");


  switch(Optiune){
    case 1: ReadSel1(); break;
    case 2: ReadSel2(); break;
    case 3: ReadSel3(); break;
    case 4: ReadSel4(); break;
    default: VALID_UNU = 0; break;
  }


  VALID_DOI = 1;


  while(VALID_DOI && VALID_UNU)
  {
    printf("Start = ");
    scanf("%ld", &St);
```

```
    printf("\nFinal = ");
    scanf("%ld", &Final);
    printf("\n");

    SolveFunction(St);

    printf("\nAlegeti una din urmatoarele:\n");
    printf("1. Alege noduri noi pentru Start si Final\n");
    printf("2. Inapoi la meniul principal\n");
    printf("3. Paraseste aplicatia\n");

    printf("Optiune: ");
    scanf("%d", &OP);
    switch(OP)
    {
      case 1: VALID_DOI = 1; break;
      case 2: VALID_DOI = 0; break;
      default: VALID_DOI = 0, VALID_UNU = 0; break;
    }
  }
  }

  return 0;
}
```

6. Bibliografie

1. *Thomas H. Cormen, Leisserson, Rivest – Introducere in algoritmi*
2. *Infoarena.ro*
3. *Wikipedia.com*
4. *Librarie Lectii AeL 5.02*
5. *Brin W. Kernighan, Dennis M. Ritchie – Limbajul C*