**PARCURGEREA ÎN ADÂNCIME**

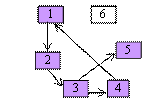
Parcurgerea unui graf in adancime se face prin utilizarea stivei (alocate implicit prin subprograme recursive).

Pentru fiecare varf se parcurge primul dintre vecinii lui neparcursi inca

Dupa vizitarea varfului initial x1, se exploreaza primul varf adiacent lui, fie acesta x2 , se trece apoi la primul varf adiacent cu x2 si care nu a fost parcurs inca , s.a.m.d.

Fiecare varf se parcurge cel mult odata

De exemplul pentru garful din figura de mai jos, se va proceda in felul urmator:

****SE obtine : **1, 2, 3, 4 , 5**.

Varful 6 ramane nevizitat.

**Algoritmul**

-Graful se va memora utilizand matricea de adiacenta a[10][10]

-pentru a nu parcurge un varf de doua ori se va folosi un vector boolean viz care va retine :

- viz[k]=0 daca varful k nu a fost vizitat inca

- viz[k]=1 daca varful k a fost vizitat

-ca si la parcurgerea in latime vecinii unui varf se « cauta » pe linia acestui varf : daca a[nod][k]=1 inseamna ca varfurile nod si k sunt adiacente. Pentru ca varful k sa fie fie parcurs trebuie ca varful sa nu fi fost vizitat : viz[k]=0

#include<fstream.h>

#include<conio.h>

int a[50][50],n,m,viz[100];

 void adanc(int nod)

{

cout<<nod<<" ";

viz[nod]=1;

for(int k=1;k<=n;k++)

if(a[nod][k]==1&&viz[k]==0)

adanc(k);

}

 void main()

{int x,y,j;

clrscr();

cin>>n;

for(int i=1;i<=n;i++)

for(int j=1;j<=n;j++)

cin>>a[i][j] ;

cout<<endl<<"matricea de adiacenta"<<endl;

for(i=1;i<=n;i++)

{for(j=1;j<=n;j++)

cout<<a[i][j]<<" ";

cout<<endl;}

cout<<endl<<"parcurgere in adancime incepand de la varful 1"<<endl;

adanc(1);

getch();}

**Aplicatii :**

1.Sa se parcurga graful in adancime pornind pe rand de la toate varfurile

2. Sa se determine daca pornind de la varful x se poate ajunge la varful y

3. Sa se determine daca se poate parcurge pornind de la un varf tot graful

Intr-un parc de distractii exista « Casa Labirint ». Aceasta este formata din n incaperi intre care se gasesc culoare cu un singur sens de parcurgere. Se stie stie incaperea de la care se porneste.

a)      Sa se determine care sunt incaperile cu cele mai multe coridoare de legatura

b)      Care sunt incaperile in care traseul se blocheaza (fara iesire)

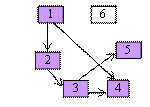
c)      Exista vreo incapere care sa aiba acces direct la toate celelalte

Rezolvarea multor probleme de grafuri, presupune parcurgerea lor de la un anumit nod. Pentru explorarea grafurilor, exista doua tipuri de algoritmi: de explorarea in latime si de explorare in adancime. Parcurgerea grafurilor orientate este similara cu a grafurilor neorientate, se tine cont insa de orientare.

**PARCURGEREA GRAFURILOR ÎN LĂȚIME**

La explorarea in latime, dupa vizitarea varfului initial, se exploreaza toate varfurile adiacente lui, se trece apoi la primul varf adiacent si se exploreaza toate varfurile adiacente acestuia si neparcurse inca, s.a.m.d.

Fiecare varf se parcurge cel mult odata.

**se obtine 1, 2, 4, 3, 5**

Varful 6 ramane neparcurs

Daca se parcurge graful incepand de la varful 2, solutia este : 2,3,4,5, in timp ce parcurgerea incepand cu 4 va retine doar varful 4

**Algoritmul**

Se va folosi o coada in care se inscriu nodurile in forma in care sunt parcurse: nodul initial *varf* (de la care se porneste), apoi varfurilr a,b,..., adiacente lui *varf*, apoi cele adiacente lui a, cele adiacente lui b,... ,s.a.m.d.

Coada este folosita astfel:

- se incarca primul varf in coada;

- se afla toate varfurile adiacente cu primul nod si se introduc dupa primul varf

- se ia urmatorul nod si i se afla nodurile adiacente

- procesul se repeta pana cand se ajunge la sfarsitul cozii

-Graful se va memora utilizand matricea de adiacenta a[50][50]

-pentru memorarea succesiunii varfurilor parcurse se va folosi un vector c[50] care va functiona ca o coada

-pentru a nu parcurge un varf de doua ori se va folosi un vector boolean viz[50] care va retine :

- viz[k]=0 daca varful k nu a fost vizitat inca

- viz[k]=1 daca varful k a fost vizitat

-doua variabile : prim si ultim vor retine doua pozitii din vectorul c si anume :

- p este indicele componentei pentru care se parcurg vecinii (indexul componentelor marcate cu rosu in sirurile parcurse anterior ). Prin urmare Varf=c[p], este elementul pentru care se determina vecinii (varfurile adiacente)

-u este pozitia in vector pe care se va face o noua inserare in vectorul c (evident, de fiecare data cand se realizeaza o noua inserare se mareste vectorul)

-vecinii unui varf se « cauta » pe linia acestui varf : daca a[varf][k]=1 inseamna ca varf si k sunt adiacente. Pentru ca varful k sa fie adaugat in coada trebuie ca varful sa nu fi fost vizitat : viz[k]=0

**void latime(int x)**

**{int c[50], viz[50],u,p,v ;**

**for (i=1;i<=n;i++) viz[i]=0;**

**c[1]=x;viz[x]=1;**

**p=1;u=1;**

**while (p<=u)**

**{v=c[p];**

**for (i=1;i<=n;i++)**

**if (a[v][i]==1&&viz[i]==0)**

**{u++ ;**

**c[u]=i ;**

**viz[i]=1;}**

**p=p+1;**

**}**

**for (i=1;i<=p;i++)**

**cout<<c[i] ;**

**}**