Prin **parcurgerea** unui graf neorientat se înţelege examinarea în mod sistematic a vârfurilor, plecând dintr-un vârf dat start, astfel încât fiecare vârf accesibil din start pe muchii incidente două câte două să fie vizitat o singură dată. Trecerea de la un vârf x la altul se face prin examinarea, într-o anumită ordine a vecinilor săi.

Parcurgerea în lăţime (BF)

https://www.pbinfo.ro/articole/5512/parcurgerea-in-latime

Se parcurge vârful de start, apoi vecinii acestuia, apoi vecinii nevizitați ai acestora, etc, până când sunt vizitate toate vârfurile accesibile. Practic, pentru a stabili ordinea de vizitare se folosește o coadă (c), iar pentru a stabili dacă un vârf a fost sau nu vizitat se foloseşte un vector caracteristic (viz).

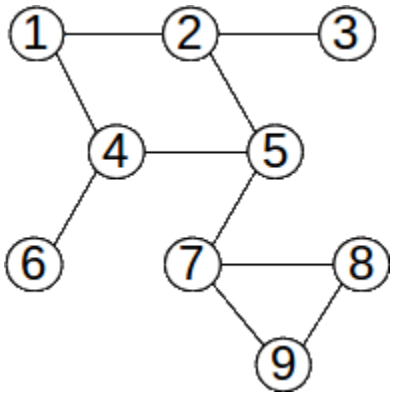
Algoritmul este:

- adaugăm în coadă vârful inițial și îl vizităm
- cât timp coada este nevidă
    - extragem un element din coadă
    - determinăm vecinii nevizitați ai vârfului extras, îi vizităm și îi adăugăm în coadă
    - eliminăm elementul din coadă

Observație

Dacă graful nu este conex, în urma parcurgerii nu se vor vizita toate vârfurile.

Exemplu



BF(5) : 5 2 4 7 1 3 6 8 9

BF(1): 1 2 4 3 5 6 7 8 9

Implementare C++

```
#include <iostream.h>
#include <fstream.h>
int a[100][100], c[100], viz[100], n;
void citire()    //citeste extremitatile muchiilor din fisier si creeaza matricea de adiacenta
{int i,j;
  ifstream f("graf.in");
  f>>n;
  while(f>>i>>j)
       a[i][j]=a[j][i]=1;
}
```

```
void BF(int nod)  //afiseaza nodurile in parcurgerea BF din vf. “nod”
{int p, u, v, i;
p=u=1;
c[p]=nod; //adauga in coada nodul de pornire
viz[nod]=1;   //marcheaza nodul de pornire ca vizitat
 while (p<=u) //cat timp mai exista elemente in coada
   { v=c[p];   //extrage nodul de la inceputul cozii
     for(i=1;i<=n;i++)  //parcurge toate cele n noduri
       if (a[v][i]==1 && !viz[i])  //daca nodul i este adiacent cu nodul v si este nevizitat inca
       {
         c[++u]=i;  //adauga nodul i la sfarsitul cozii
         viz[i]=1;  //marcheaza nodul i ca vizitat
       }
    p++;  //avanseaza la urmatorul nod de la inceputul cozii
   }
   for(i=1;i<=u;i++)  cout<<c[i]<< " "; // afiseaza nodurilor din coada
}

int main()
{int nod;
 citire();
 cin>>nod;  //citeste nodul de pornire
 BF(nod);
 return 0;
}
```

Parcurgerea în adâncime (DF)

https://www.pbinfo.ro/articole/5511/parcurgerea-in-adancime

Parcurgerea în adâncime reprezintă explorarea “naturală” a unui graf neorientat. Este foarte asemănătoare cu modul în care un turist vizitează un oraș în care sunt obiective turistice (vârfurile grafului) și căi de acces între obiective (muchiile). Vizitarea orașului va avea loc din aproape în aproape: se pleacă de la un obiectiv de pornire, se continuă cu un obiectiv învecinat cu acesta, apoi unul învecinat cu al doilea, etc.

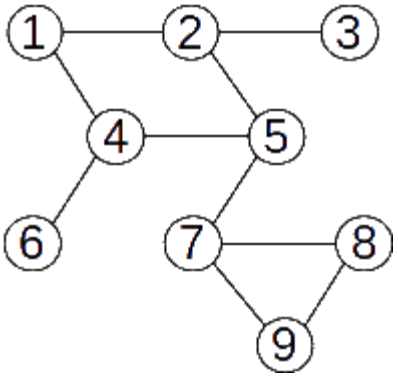
Parcurgerea în adâncime se face astfel:

- Se începe cu un vârf inițial x, care este în acest moment **vârf curent**.
- Vârful x se vizitează. Se determină primul său vecin nevizitat y al lui x, care devine vârf curent.
- Apoi se vizitează primul vecin nevizitat al lui y, și așa mai departe, mergând în adâncime, până când ajungem la un vârf care nu mai are vecini nevizitați. Când ajungem într-un astfel de vârf, ne întoarcem la “părintele” acestuia – vârful din care am ajuns în acesta.
- Dacă acest vârf mai are vecini nevizitați, alegem următorul vecin nevizitat al său și continuam parcurgerea în același mod.
- Dacă nici acest vârf nu mai are vecini nevizitați, revenim în vârful său părinte și continuăm în același mod, până când toate vârfurile accesibile din vârful de start sunt vizitate.

Observație

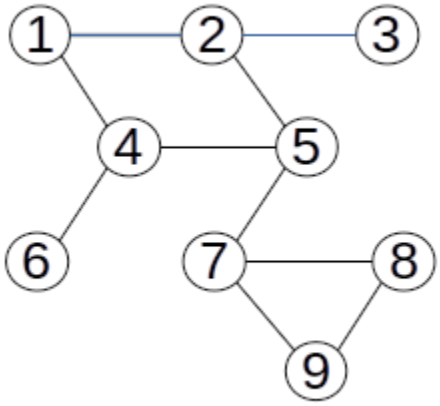
Dacă graful nu este conex, nu se vor vizita toate vârfurile.

Exemplu



DF(5): 5 2 1 4 6 3 7 8 9

Alte exemple de parcurgere pe acest graf:



- Parcurgerea din nodul 1: 1 2 3 5 4 6 7 8 9
- Parcurgerea din nodul 2: 2 1 4 5 7 8 9 6 3
- Parcurgerea din nodul 9: 9 7 5 2 1 4 6 3 8

Pentru implementarea algoritmului se foloseşte un vector caracteristic pentru memorarea faptului că un anume vârf a fost sau nu vizitat, la un anumit moment al parcurgerii:

- viz[i] = 0, vârful i nu a fost (încă) vizitat
- viz[i] = 1, vârful i a fost vizitat

Pentru a determina ordinea în care se parcurg nodurile care pot fi vizitate, se foloseşte o stivă:

- se analizează mereu nodurile adiacente cu nodul din vârful stivei
- dacă pentru nodul din vârful stivei găsim un vecin nevizitat, îl adăugăm în stivă
- dacă pentru nodul din vârful stivei nu mai găsim niciun vecin nevizitat, îl eliminăm din stivă

Implementare C++ (recursiva)

Presupunem că graful are n noduri și este prezentat prin matricea de adiacență a. Starea unui vârf (vizitat sau nu) este memorată în vectorul caracteristic viz. Toate aceste variabile sunt globale.

```
void df(int nod)
{
  viz[nod]=1; //viziteaza nodul curent
  cout<<nod<<" ";  // afiseaza nodul curent
  for(int i=1; i<=n; i++) // cauta primul vecin nevizitat al nodului curent “nod”
    if(a[nod][i]==1 && !viz[i])
        df(i); // continua parcurgerea cu primul vecin nevizitat
}
```

Parcurgerea grafurilor neorientate poate fi folosită în rezolvarea unei game largi de probleme: verificarea conexității unui graf, determinarea componentelor conexe ale unui graf, determinarea unor lanțuri în graf, verificarea faptului că un graf este bipartit, etc.