Phần 2

LÝ THUYẾT ĐỒ THỊ

Graph Theory

GV: Nguyễn Huy Đức

Bộ môn: Khoa học Máy tính – ĐH Thủy lợi

 0903 402 655

 ducnghuy@gmail.com

Nội dung

- Chương 1: Các khái niệm cơ bản
- Chương 2: Biểu diễn đồ thị
- **Chương 3: Các thuật toán tìm kiếm trên đồ thị**
- Chương 4: Đồ thị Euler và đồ thị Hamilton
- Chương 5: Cây và cây khung của đồ thị
- Chương 6: Bài toán đường đi ngắn nhất

Chương 3

CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Các thuật toán duyệt đồ thị

- Một bài toán quan trọng trong lý thuyết đồ thị là bài toán *duyệt tất cả các đỉnh* có thể đến được từ một đỉnh xuất phát nào đó.
- Vấn đề này đưa về một bài toán liệt kê mà yêu cầu của nó là *không được bỏ sót hay lặp lại bất kỳ đỉnh nào*.
- Cần phải xây dựng những thuật toán cho phép *duyệt một cách hệ thống các đỉnh*, những thuật toán như vậy gọi là những thuật toán tìm kiếm trên đồ thị.
- *Hai thuật toán cơ bản nhất*:
  - Tìm kiếm theo chiều sâu (Depth First Search – DFS)
  - Tìm kiếm theo chiều rộng (Breadth First Search – BFS)

Chương 3: CÁC THUẬT TOÁN TÌM KIẾM TRÊN ĐỒ THỊ

Nội dung chương gồm:

- **Các thuật toán duyệt đồ thị**
  - ✓ Thuật toán tìm kiếm theo chiều sâu (Depth First Search - DFS)
  - ✓ Thuật toán tìm kiếm theo chiều rộng (Breadth First Search – BFS)

- **Ứng dụng:**
  - ✓ Tìm đường đi trên đồ thị.
  - ✓ Kiểm tra tính liên thông của đồ thị.

3.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH - DFS)

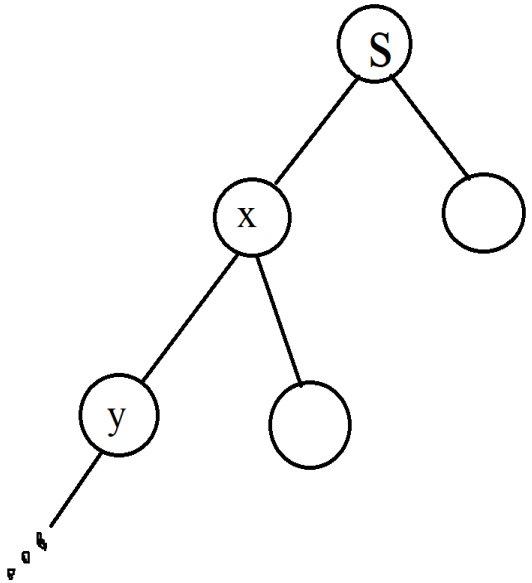
Ý tưởng của thuật toán:

- Bắt đầu tại một đỉnh S nào đó, chọn một đỉnh x bất kỳ kề với S và lấy nó làm đỉnh duyệt tiếp theo. Cách duyệt tiếp theo được thực hiện tương tự như đối với đỉnh S với đỉnh bắt đầu là x.
- Điều đó gợi ý cho ta viết một thủ tục đệ quy DFS(u) mô tả việc duyệt từ đỉnh u bằng cách thông báo thăm đỉnh u và tiếp tục quá trình duyệt DFS(v) với v là một đỉnh chưa thăm kề với u.
- Để kiểm tra việc duyệt mỗi đỉnh đúng một lần, chúng ta sử dụng một mảng *cx[]* gồm *n* phần tử (tương ứng với *n* đỉnh) để đánh dấu các đỉnh đã duyệt.

  Ban đầu *cx[u]=0*, nếu đỉnh *u* đã được duyệt thì đánh dấu *cx[u]=1*.

3.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH - DFS)

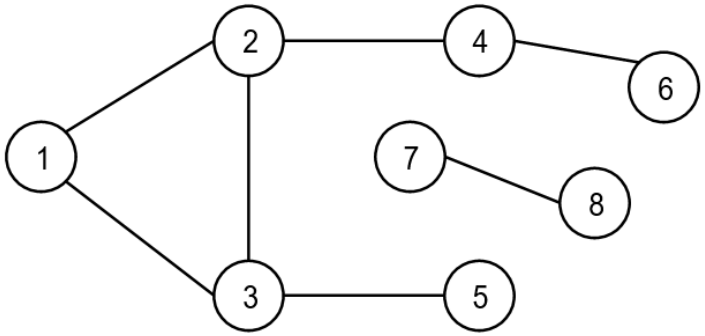
- Như vậy, từ đỉnh S ban đầu, duyệt đi xa nhất theo từng nhánh.
  - ✓ Khi nhánh đã duyệt hết, lùi về từng đỉnh để tìm và duyệt những nhánh tiếp theo.
- Quá trình duyệt chỉ dừng lại khi tìm thấy đỉnh cần tìm hoặc tất cả đỉnh đều đã được duyệt qua.

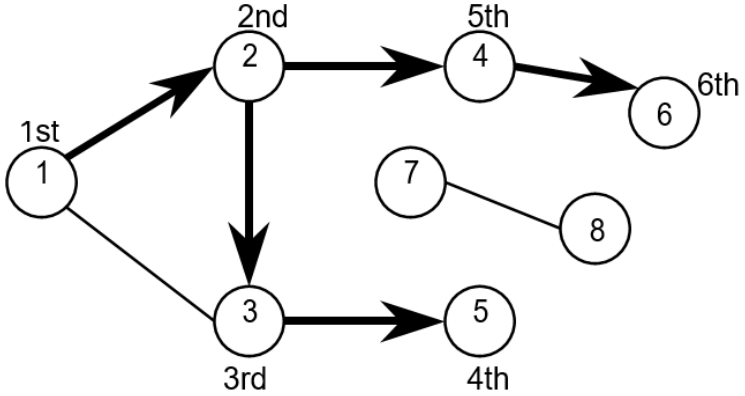


3.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH - DFS)

- **Ví dụ 1:** Với đồ thị sau đây, đỉnh xuất phát S = 1: quá trình duyệt đệ quy có thể vẽ trên cây tìm kiếm DFS sau

  (Mũi tên u→v chỉ thao tác đệ quy: DFS(u) gọi DFS(v)).

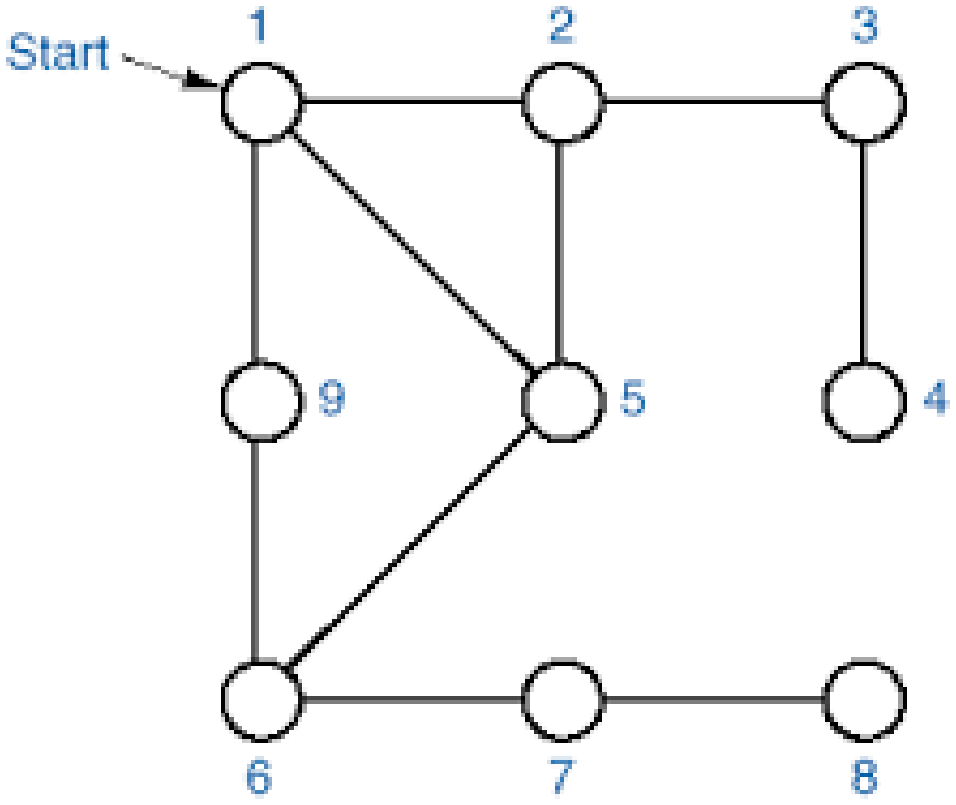




- Đỉnh 2 và 3 đều kề với 1, nhưng DFS(1) sẽ tìm thấy đỉnh 2 trước và gọi DFS(2). Trong DFS(2) sẽ xét tất cả các đỉnh kề với 2 mà chưa đánh dấu thì dĩ nhiên trước hết nó tìm thấy 3 và gọi DFS(3). Trong DFS(3) sẽ xét đỉnh 5 chưa đánh dấu và gọi DFS(5).
- Trong DFS(5): không thấy đỉnh nào kề với 5 nên TT lùi lại từng bước đến đỉnh 2 thấy có đỉnh 4 kề với đỉnh 2 và chưa đánh dấu nên gọi DFS(4),…

3.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU (DEPTH FIRST SEARCH - DFS)

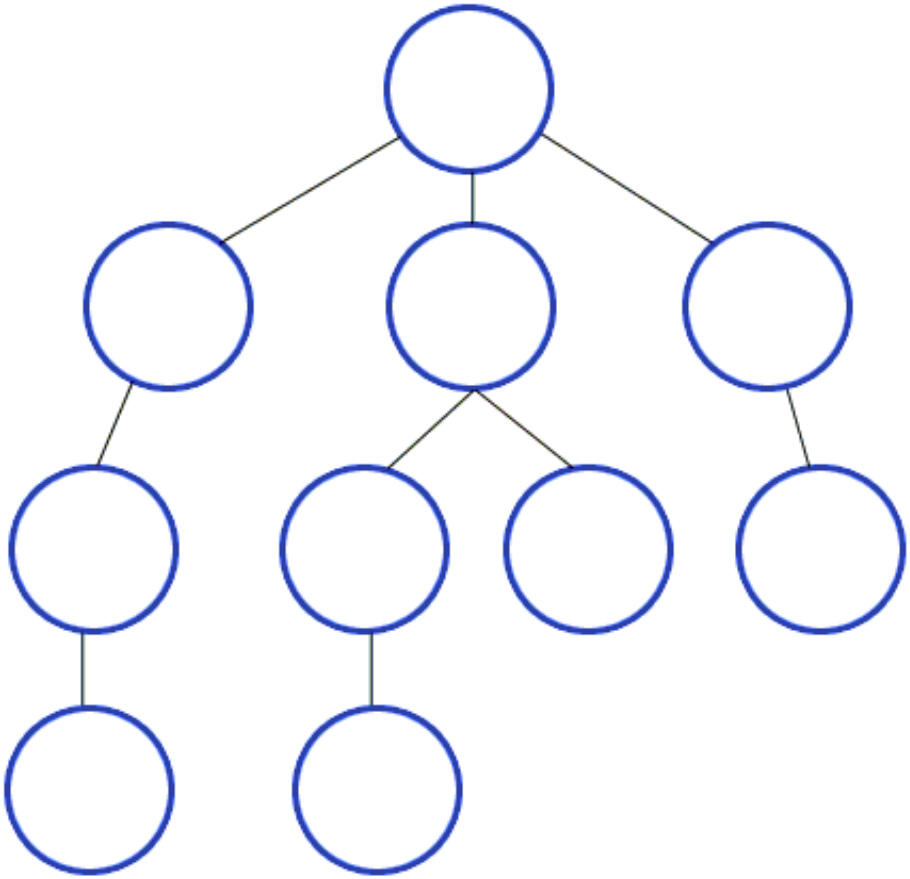
- **Ví dụ 2:** thứ tự duyệt các đỉnh theo thuật toán DFS



Depth-first traversal

3.1 THUẬT TOÁN TÌM KIẾM THEO CHIỀU SÂU
(DEPTH FIRST SEARCH - DFS)

- **Ví dụ 3:**



Chương trình trên C++

```
#include <bits/stdc++.h>
using namespace std;
const char ginp[]="DT.INP";
int a[101][101]; //Ma trận kề biểu diễn đồ thị
int n, v, s;
int cx[101]; //Mảng đánh dấu các đỉnh chưa xét
void init()
{  freopen(ginp, "r", stdin);
    //Doc du lieu vao
    cin>>n;
    for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++) cin>>a[i][j];
   //Khoi tao các dinh deu chua tham
     for (int i = 1; i <= n; i++) cx[i] = 0;
```

Chương trình trên C++

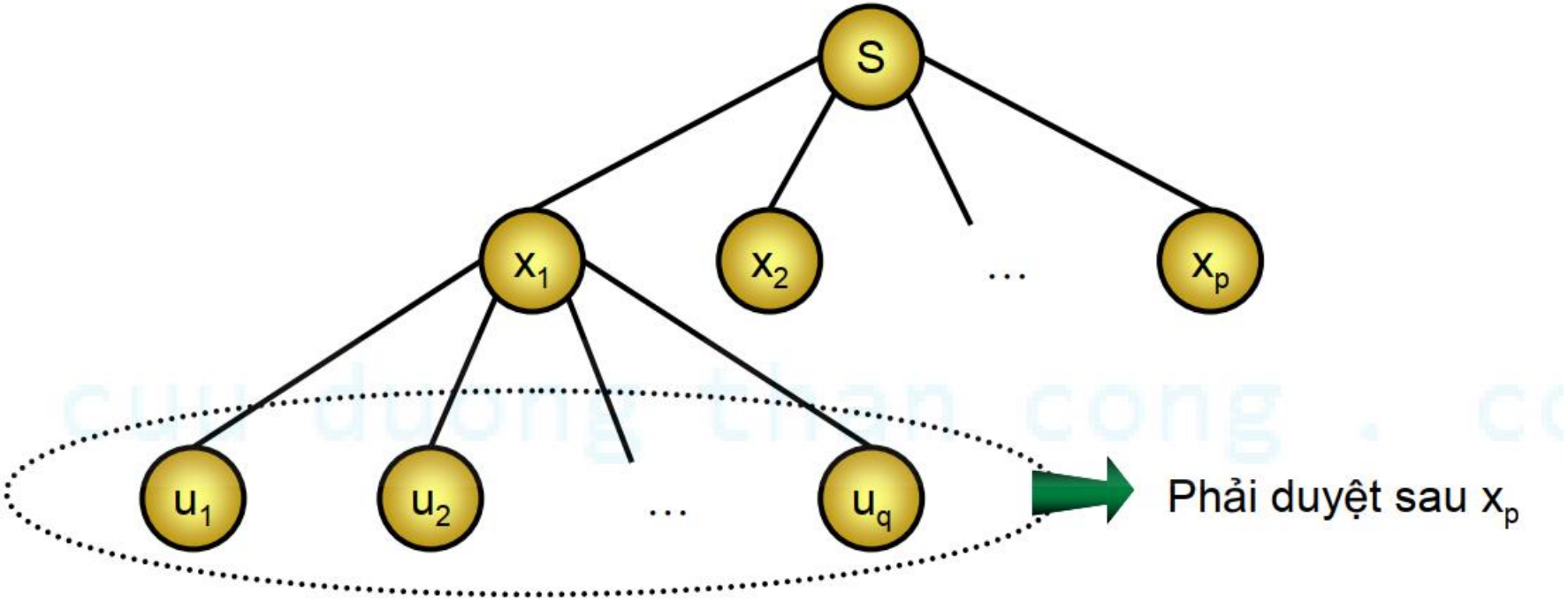
```
cout<<"So dinh: "<<n<<"\n";
cout<<"Ma tran ke cua do thi:"<<"\n";
for (int i=1; i<=n; i++)
  { for (int j=1; j<=n; j++) cout<<a[i][j]<<" ";
     cout<<"\n"; }
}
void DFS(int u)
{ cout << u <<" "; cx[u] = 1; //đánh dấu đỉnh u đã thăm.
   for (int v = 1; v <= n; v++)
       if (a[u][v] == 1 && cx[v]==0)  DFS(v); }//nếu v kề u và v chưa thăm thì gọi DFS(v)
int main()
{  init(); s=1;  cout<<"Duyet theo chieu sau bat dau tu dinh: "<<s<<"\n";
    DFS(s);  return 0; }
```

3.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG
(Breadth First Search – BFS)

Ý tưởng của thuật toán:

- Cơ sở của phương pháp cài đặt này là "lập lịch" duyệt các đỉnh. Việc thăm một đỉnh sẽ lên lịch duyệt các đỉnh kề nó sao cho thứ tự duyệt là ưu tiên chiều rộng (*đỉnh nào gần S hơn sẽ được duyệt trước*).
- Ví dụ: Bắt đầu ta thăm đỉnh S. Việc thăm đỉnh S sẽ phát sinh thứ tự duyệt những đỉnh (x_1, x_2, ..., x_p) kề với S (những đỉnh gần S nhất). Khi thăm đỉnh x_1 sẽ lại phát sinh yêu cầu duyệt những đỉnh (u_1, u_2 ..., u_q) kề với x_1. Nhưng rõ ràng các đỉnh u này "xa" S hơn những đỉnh x nên chúng chỉ được duyệt khi tất cả những đỉnh x đã duyệt xong. Tức là thứ tự duyệt đỉnh sau khi đã thăm x_1 sẽ là: (x_2, x_3..., x_p, u_1, u_2, ..., u_q).

3.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG
(Breadth First Search – BFS)



Giả sử ta có một danh sách chứa những đỉnh đang "chờ" thăm. Tại mỗi bước, ta thăm một đỉnh đầu danh sách và cho những đỉnh chưa "xếp hàng" kề với nó xếp hàng thêm vào cuối danh sách. Vì nguyên tắc đó nên danh sách chứa những đỉnh đang chờ sẽ được tổ chức dưới dạng hàng đợi (Queue).

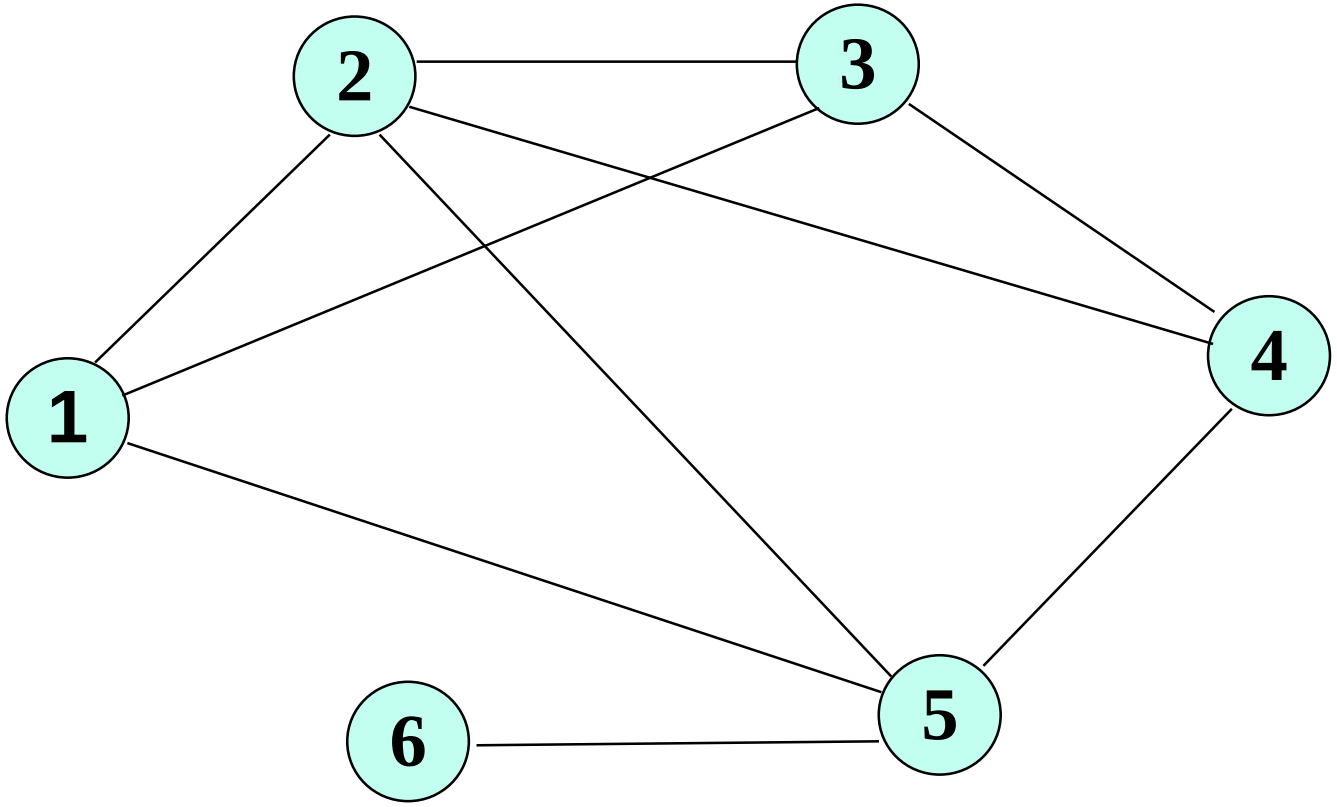
3.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG
(Breadth First Search – BFS)

Ta sẽ dựng giải thuật như sau:

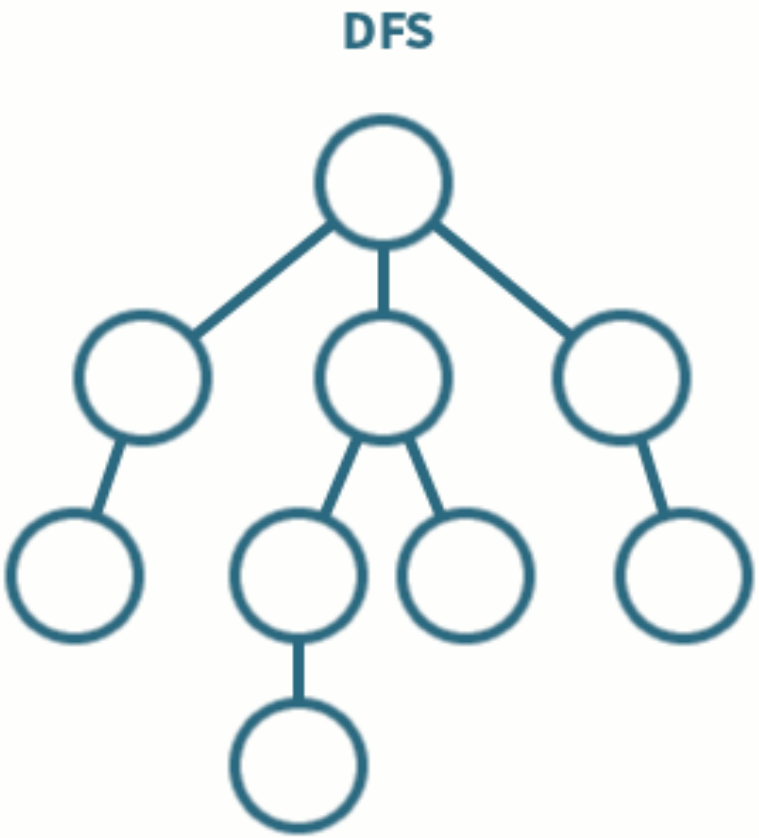
- **Bước 1:** Khởi tạo:
  - ✓ Các đỉnh đều ở trạng thái chưa đánh dấu, ngoại trừ đỉnh xuất phát S là đã đánh dấu.
  - ✓ Hàng đợi (Queue) dùng để chứa các đỉnh sẽ được duyệt theo thứ tự ưu tiên chiều rộng, ban đầu chỉ có một phần tử là S.
- **Bước 2:** Lặp các bước sau đến khi hàng đợi rỗng:
  - ✓ Lấy u khỏi hàng đợi, thông báo thăm u (Bắt đầu việc duyệt đỉnh u)
  - ✓ Xét tất cả những đỉnh v kề với u mà chưa được đánh dấu, với mỗi đỉnh v đó:
    1. Đánh dấu v.
    2. Đẩy v vào hàng đợi (v sẽ chờ được duyệt tại những bước sau)

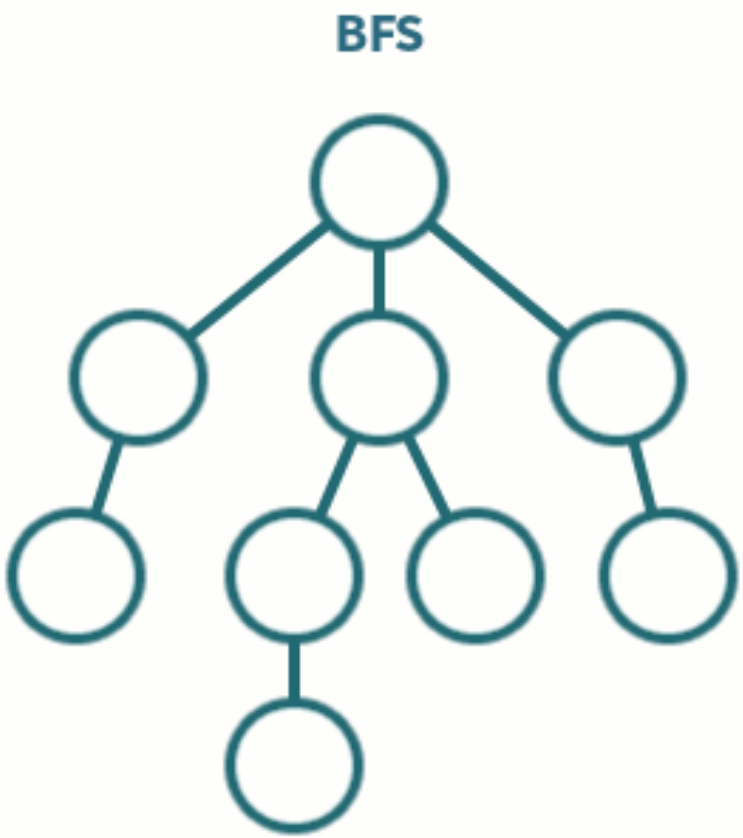
3.2 THUẬT TOÁN TÌM KIẾM THEO CHIỀU RỘNG
(Breadth First Search – BFS)

- **Ví dụ:** Duyệt theo TT BFS bắt đầu từ đỉnh 1



DFS với BFS





Chương trình trên C++

```
void BFS(int u)
{       int v, dau=1, cuoi=1;
        Q[cuoi]=u; cx[u]=1;
        while (dau<=cuoi)
        {       v=Q[dau]; dau++;
                cout << v <<" ";
                for (int j=1; j<=n; j++)
                      if (a[v][j]==1 && cx[j]==0)
                            {cuoi++; Q[cuoi]=j; cx[j]=1; }
        }
}
```

Chương trình duyệt cây theo DFS và BFS trên C++

```
#include <bits/stdc++.h>
using namespace std;
const char ginp[]="DT.IN1";
int a[101][101];
int n, cx[101], Q[101],v, s;
void init()
{  freopen(ginp, "r", stdin);  //Mo tep de doc du lieu vao
    cin>>n;
    for (int i=1; i<=n; i++)
            for (int j=1; j<=n; j++) cin>>a[i][j];
   //in ma tran ke
    cout<<"So dinh: "<<n<<"\n";
    cout<<"Ma tran ke cua do thi:"<<"\n";
    for (int i=1; i<=n; i++)
            {  for (int j=1; j<=n; j++) cout<<a[i][j]<<" ";
               cout<<"\n";
            }
}
```

Chương trình duyệt cây theo DFS và BFS trên C++

```
void DFS(int u)
{  cout << u <<" ";
   cx[u] = 1;
   for (int v = 1; v <= n; v++)
      if (a[u][v] == 1 && cx[v]==0) DFS(v);
}

void BFS(int u)
{    int v, dau=1, cuoi=1;
     Q[cuoi]=u; cx[u]=1;
     while (dau<=cuoi)
          {   v=Q[dau]; dau++;
              cout << v <<" ";
              for (int j=1; j<=n; j++)
                    if (a[v][j]==1 && cx[j]==0)  {cuoi++; Q[cuoi]=j; cx[j]=1; }
          }
}
```

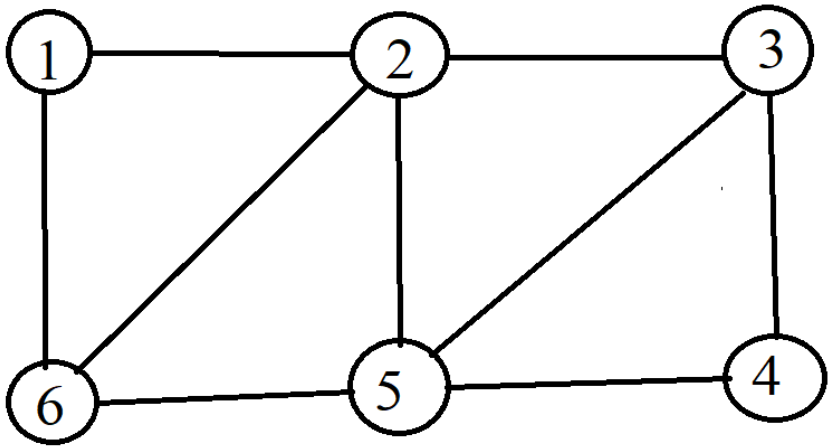
Chương trình duyệt cây theo DFS và BFS trên C++

```
int main()
{
    init();
    s=1; //dinh bat dau
    //Khoi tao các dinh deu chua tham
            for (int i = 1; i <= n; i++) cx[i] = 0;
    cout<<"Duyet theo chieu sau bat dau tu dinh "<<s<<": \n";
    DFS(s);
    cout<<"\n";
    //Khoi tao các dinh deu chua tham
            for (int i = 1; i <= n; i++) cx[i] = 0;
    cout<<"Duyet theo chieu rong bat dau tu dinh "<<s<<": \n";
    BFS(s);
    return 0;
}
```

Ví dụ duyệt đồ thị theo DFS và BFS

- Tệp dữ liệu DT.INP:

```
6
0 1 0 0 0 1
1 0 1 0 1 1
0 1 0 1 1 0
0 0 1 0 1 0
0 1 1 1 0 1
1 1 0 0 1 0
```



Đồ thị G

- Kết quả duyệt:

```
Duyet theo chieu sau bat dau tu dinh: 1
1 2 3 4 5 6
Duyet theo chieu rong bat dau tu dinh: 1
1 2 6 3 5 4
```

3.3 Tìm đường đi và kiểm tra tính liên thông

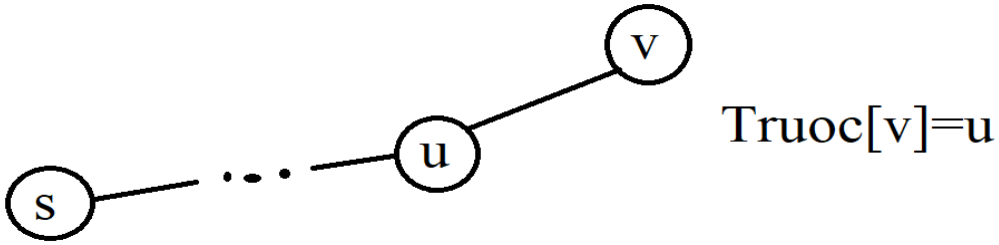
1. Bài toán tìm đường đi giữa 2 đỉnh:

Cho s và t là 2 đỉnh của đồ thị. Tìm đường đi từ s đến t ?

- Duyệt các đỉnh của đồ thị bắt đầu từ đỉnh s (thực hiện DFS(s) hoặc BFS(s)) sẽ cho phép thăm tất cả các đỉnh thuộc cùng thành phần liên thông với s.
- Có 2 khả năng:
  - ✓ Đỉnh t không cùng thành phần liên thông với s: khi đó đỉnh t không được đánh dấu là đã thăm (cx[t]= 0), khi đó, không có đường đi từ s đến t.
  - ✓ Đỉnh t cùng thành phần liên thông với s: đỉnh t được đánh dấu là đã thăm (cx[t]= 1), trường hợp này có đường đi từ s đến t.

3.3 Tìm đường đi và kiểm tra tính liên thông

- Để ghi nhận đường đi, dùng thêm biến Truoc[v] để lưu đỉnh đi trước đỉnh v trong đường đi từ s đến v.



- Đặt thêm câu lệnh này vào hàm DFS(u) và BFS(u):

✓ ***Trong DFS(u):*** sửa câu lệnh `if (a[u][v] == 1 && cx[v]==0) {DFS(v); }`
thành câu lệnh `if (a[u][v] == 1 && cx[v]==0) {Truoc[v]=u; DFS(v);}`

✓ ***Trong BFS(u):***
sửa câu lệnh `if (a[v][j]==1 && cx[j]==0) {cuoi++; Q[cuoi]=j; cx[j]=1; }`
thành
`if (a[v][j]==1 && cx[j]==0) {cuoi++; Q[cuoi]=j; cx[j]=1; Truoc[j]=v; }`

3.3 Tìm đường đi và kiểm tra tính liên thông

- Đường đi từ s đến t được khôi phục theo quy tắc:

$$t \leftarrow p1 = Truoc[t] \leftarrow p2 = Truoc[p1] \leftarrow ... \leftarrow s.$$

- *Nhận xét:* đường đi tìm được theo thuật toán tìm kiếm theo chiều rộng từ s tới t sẽ là đường đi ngắn nhất (theo nghĩa qua ít cạnh nhất) – điều này là do thứ tự thăm đỉnh của thuật toán tìm kiếm.

3.3 Tìm đường đi và kiểm tra tính liên thông

2. Tìm các thành phần liên thông của đồ thị:

Cho biết đồ thị có bao nhiêu thành phần liên thông, từng thành phần liên thông gồm những đỉnh nào ?

- Dùng biến solt để đếm số thành phần liên thông, lúc đầu solt=0, tăng solt lên 1 trước mỗi lần gọi DFS(u) (hoặc BFS(u)).
- Số thành phần liên thông bằng số lần gọi DFS(u) (hoặc BFS(u)).
- Dùng biến đánh dấu đỉnh đã thăm cx[v] để ghi nhận đỉnh đó đã thăm và ở thành phần liên thông nào:

  Lúc đầu cx[v]=0, khi đã thăm đỉnh v, thay cho gán cx[v]=1 thì gán **cx[v]= solt**.

Chương trình tìm đường đi và kiểm tra liên thông

```
#include <iostream>
using  namespace std;
const char ginp[]="DT.IN1";
int a[101][101];
int n, cx[101], Q[101], truoc[101], solt=0;
void init()
{  freopen(ginp, "r", stdin); //Doc du lieu vao
    cin>>n;
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++) cin>>a[i][j];
   cout<<"So dinh: "<<n<<"\n"; //in ma tran ke
   cout<<"Ma tran ke cua do thi:"<<"\n";
   for (int i=1; i<=n; i++)
      {  for (int j=1; j<=n; j++) cout<<a[i][j]<<" ";
         cout<<"\n";  }
}
```

Chương trình tìm đường đi và kiểm tra liên thông

```
void DFS(int u)
{  //cout << u <<" "; //Tim duong di va kiem tra lien thong khong can den
   cx[u] = solt;
   for (int v = 1; v <= n; v++)
      if (a[u][v] == 1 && cx[v]==0)
         {truoc[v]=u; DFS(v); }
}

void BFS(int u)
{   int v, dau=1, cuoi=1;
    Q[cuoi]=u; cx[u]=solt;
    while (dau<=cuoi)
          {  v=Q[dau]; dau++;
            //cout << v <<" "; //Tim duong di va kiem tra lien thong khong can den
            for (int j=1; j<=n; j++)
                  if (a[v][j]==1 && cx[j]==0)
                        cuoi++; Q[cuoi]=j; cx[j]=solt; truoc[j]=v; }
           }
}
```

Chương trình tìm đường đi và kiểm tra liên thông

```
void Duyet()
{
     int s=1; //duyet bat dau tu s
   //Khoi tao các dinh deu chua tham
     for (int i = 1; i <= n; i++)  cx[i] = 0;
     cout<<"Duyet theo chieu sau bat dau tu dinh: "<<s<<"\n";
     solt=1; DFS(s);
     cout<<"\n";

   //Khoi tao các dinh deu chua tham
     for (int i = 1; i <= n; i++)   cx[i] = 0;
     cout<<"Duyet theo chieu rong bat dau tu dinh: "<<s<<"\n";
     solt=1; BFS(s);
     cout<<"\n";
}
```

Chương trình tìm đường đi và kiểm tra liên thông

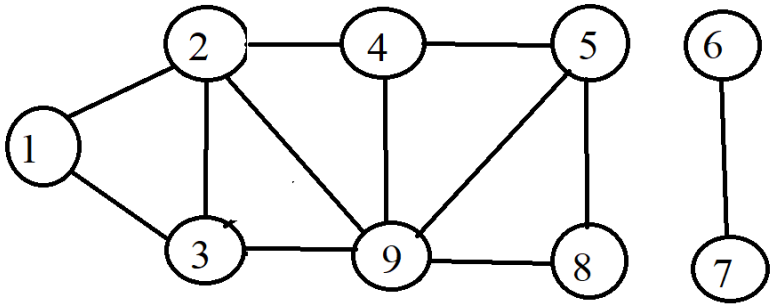
```
void Duongdi()
{
   int s=1, t=5; //tìm duong di tu dinh s den dinh t
   //Khoi tao các dinh deu chua tham
   for (int v = 1; v <= n; v++) {truoc[v]=0; cx[v] = 0; }
   solt=1;
   DFS(s);
   //BFS(s);
   if (cx[t]==0)
        cout<<"Khong co duong di tu "<<s<<" den "<<t<<endl;
   else  {
            cout<<"Duong di tu "<<s<<" den "<<t<<": "<<endl;
            cout<<t;
            int p=t;
            while (truoc[p]!= 0)
                  {  p= truoc[p]; cout<<" <-- "<<p;  }
         }
    cout<<endl;
}
```

Chương trình tìm đường đi và kiểm tra liên thông

```
void lienthong()
{   //Khoi tao các dinh deu chua tham
     for (int v = 1; v <= n; v++) cx[v] = 0;
     solt=0;
     for (int v = 1; v <= n; v++)
            if (cx[v] == 0) { solt++;
                              DFS(v);   //BFS(v);
                             }
      if (solt==1)
            cout<<"Do thi lien thong"<<endl;
      else
            {  cout<<"Do thi co "<<solt<<" thanh phan lien thong"<<endl;
               for (int i= 1; i<=solt; i++)
                     {  cout<<"Thanh phan lien thong "<<i<<" gom cac dinh:"<<endl;
                        for (int v=1; v<=n; v++)
                              if (cx[v]==i) cout<<v<<" ";
                        cout<<endl;  }
            }
}
```

Chương trình tìm đường đi và kiểm tra liên thông

```
int main()
{
        init();
         //Duyet();
         Duongdi();
         lienthong();
         return 0;
}
```



Đồ thị DT.IN2

File DT.IN2

```
9
0 1 1 0 0 0 0 0 0
1 0 1 1 0 0 0 0 1
1 1 0 0 0 0 0 0 1
0 1 0 0 1 0 0 0 1
0 0 0 1 0 0 0 1 1
0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 0 1
0 1 1 1 1 0 0 1 0
```

Kết quả chạy với đồ thị DT.IN2

Duong di tu 1 den 5:
5 <-- 4 <-- 9 <-- 3 <-- 2 <-- 1
Do thi co 2 thanh phan lien thong
Thanh phan lien thong 1 gom cac dinh:
1 2 3 4 5 8 9
Thanh phan lien thong 2 gom cac dinh:
6 7