

# CÀI ĐẶT ĐỒ THỊ (TT)

## Các thuật toán tìm kiếm trên đồ thị

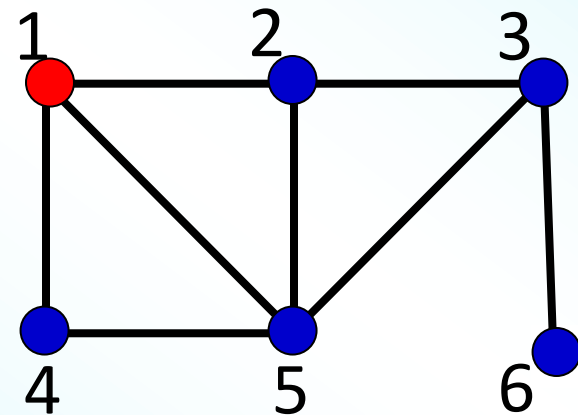
# **1. Tìm kiếm theo chiều sâu (Depth First Search – DFS)**

# Ý tưởng

- B1. Xuất phát từ 1 đỉnh cho trước nào đó.
- B2. Xử lý đỉnh này và đánh dấu để không xử lý lần sau.
- B3. Đưa tất cả các đỉnh kề với nó vào danh sách xử lý và chọn 1 đỉnh để xử lý tiếp theo.
- B4. Quay lại B2 cho đến khi không còn đỉnh trong danh sách.

## VD:

- Bắt đầu từ 1. Đưa các đỉnh kề với 1 vào DS: 2, 4, 5
  - Chọn 2 để xử lý. Đưa các đỉnh kề với 2 vào DS: 3, 4, 5, ...
- Thứ tự: 1 2 3 5 4 6



# Cài đặt DFS

## ■ Phân tích:

- ◆ Dùng cấu trúc Stack
- ◆ Sử dụng mảng đánh dấu là mảng 1 chiều:
  - `int danh dau[maxV];`
  - Quy ước:
    - `danh dau[i] = 0`; đỉnh *i* chưa được xét
    - `danh dau[i] = 1`; đỉnh *i* đã được xét

# Khai báo CTDL

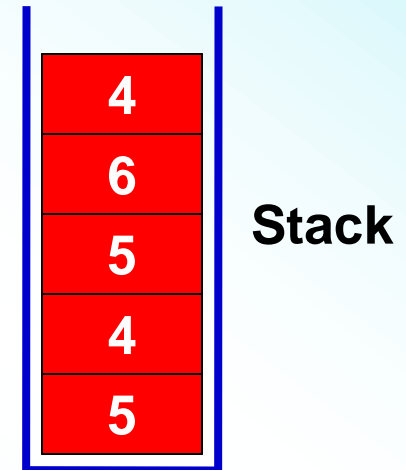
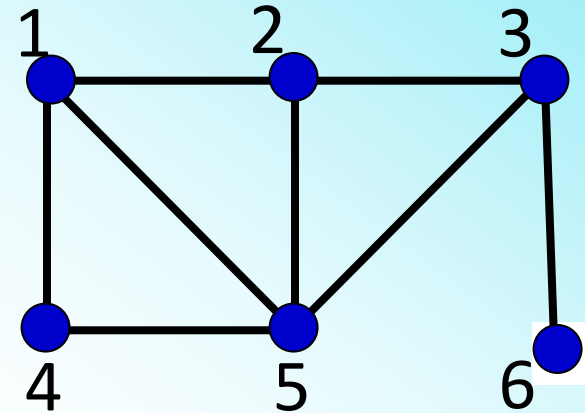
```
typedef struct DOTHI
{
    int nV;           // số đỉnh
    int nE;           // số cạnh
    int type;         // 0: vô hướng, 1: có hướng
    int mtke[maxV][maxV]; // ma trận kề
};
```

# Cài đặt DFS (tt)

```
void DFS(DOTHI g, int s)           // s là đỉnh xuất phát
{
    int danhdao[maxV]; Stack st;
    //Khoi tao
    for (int i = 1; i<=g.nV; i++)
        danhdao[i] = 0;           // chua co dinh nao duoc xet
    Khoitao(st);                   // Khoi tao Stack
    Push(st,s);                    // Dua s vao Stack
    while (!isEmpty(st))           //Trong khi Stack chua rong
    {
        int v = Pop (st);          // Lay v ra khoi Stack
        if (danhdao[v] != 1)       // Neu v chua xet
        {
            cout<<v<<" ";        danhdao[v] = 1;
            for (i=g.nV; i>=1; i--)
                if (!danhdao[i] && g.mtke[v][i] != 0)
                    Push(st,v);
        }
    }
}
```

# Cài đặt DFS (tt)

- Đưa 1 vào Stack
- Lấy 1 ra xử lý, đưa 5, 4, 2 vào Stack
- Lấy 2 ra xử lý, đưa 5, 3 vào Stack
- Lấy 3 ra xử lý, đưa 6, 3 vào Stack
- Lấy 5 ra xử lý, đưa 4 vào Stack
- Lấy 4 ra xử lý. Không đưa gì vào Stack
- Lấy 6 ra xử lý. Không đưa gì vào Stack
- Lấy 5 ra. Không xử lý (vì đã xử lý rồi)
- Lấy 4 ra. Không xử lý
- Lấy 5 ra. Không xử lý

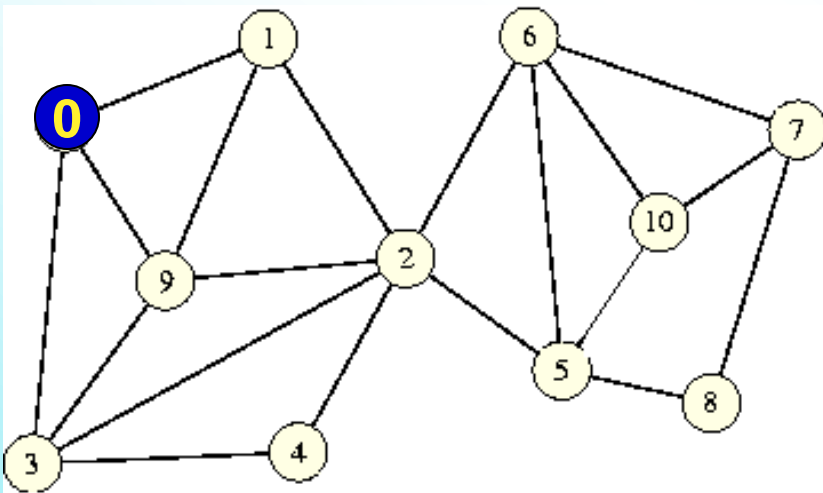


**Thứ tự duyệt:**

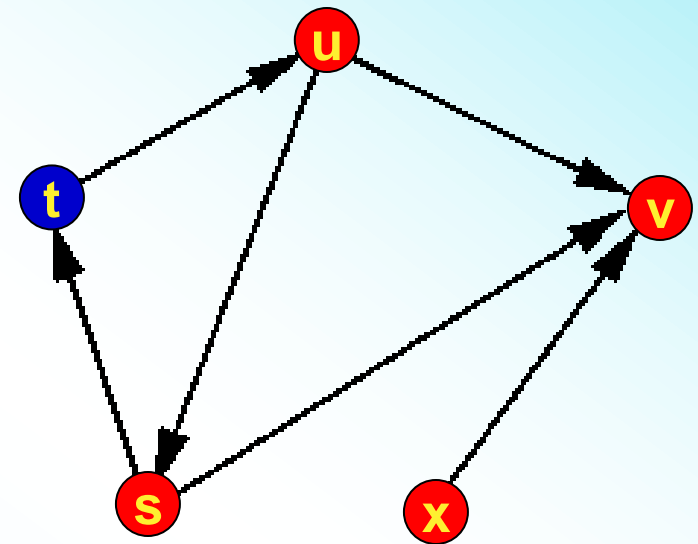


# Ví dụ về DFS

- Áp dụng DFS, hãy thể hiện thứ tự duyệt các đỉnh trong đồ thị sau:



**Đáp án: 0 1 2 3 4 9 5 6 7 8 10**



**Đáp án: t u s v**

**Đỉnh x không được duyệt**



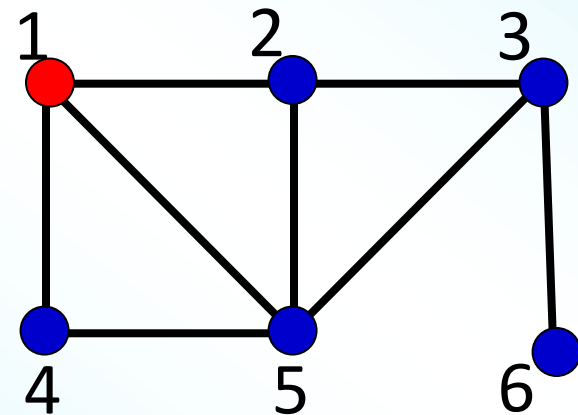
## **2. Tìm kiếm theo chiều rộng (Breadth First Search - BFS)**

# Ý tưởng

- B1. Xuất phát từ 1 đỉnh cho trước nào đó.
- B2. Xử lý đỉnh này và đánh dấu để không xử lý lần sau.
- B3. Đưa tất cả các đỉnh kề với nó vào danh sách xử lý và lần lượt xử lý các đỉnh kề với đỉnh đang xét
- B4. Quay lại B2 cho đến khi không còn đỉnh trong danh sách.

## VD:

- Bắt đầu từ 1. Đưa các đỉnh kề với 1 vào DS: 2, 4, 5
  - Chọn 2 để xử lý. Đưa các đỉnh kề với 2 vào DS: 3, 4, 5, ...
- Thứ tự: 1 2 4 5 3 6



# Cài đặt BFS

## ■ Phân tích:

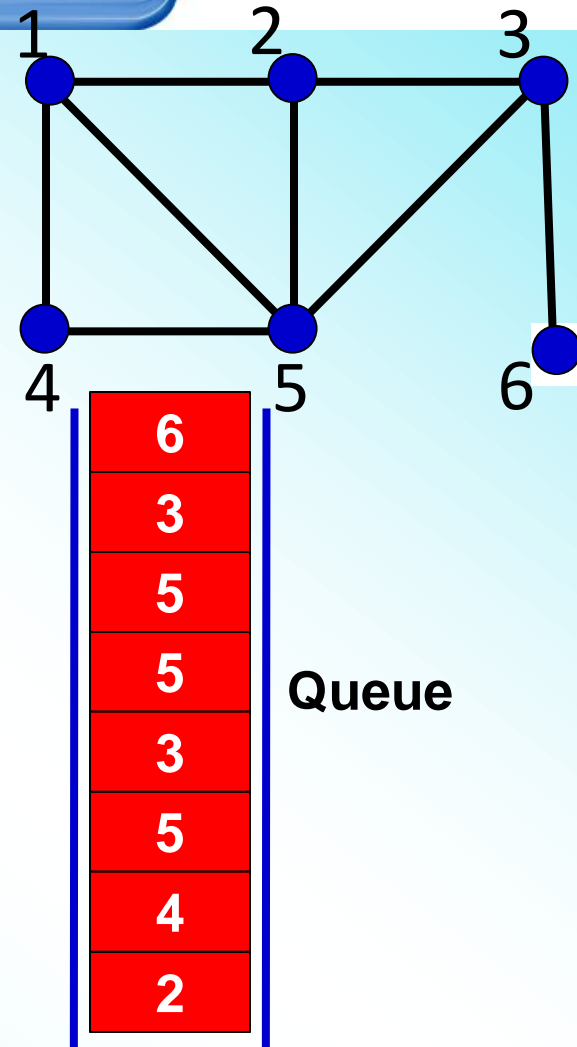
- ◆ Dùng cấu trúc Queue
- ◆ Sử dụng mảng đánh dấu là mảng 1 chiều:
  - `int danh dau[maxV];`
  - Quy ước:
    - `danh dau[i] = 0;`      đỉnh *i* chưa được xét
    - `danh dau[i] = 1;`      đỉnh *i* đã được xét

# Cài đặt BFS (tt)

```
void BFS(DOTHI g, int s)           // s là đỉnh xuất phát
{
    int danhdao[maxV];           Queue q;
    //Khoi tao
    for (int i = 1; i<=g.nV; i++)
        danhdao[i] = 0;         // chua co dinh nao duoc xet
    Khoitao(q);                  // Khoi tao Queue
    // Bat dau
    Push(q,s);                   // Dua s vao Queue
    while (!isEmpty(q))          //Trong khi Queue chua rong
    {
        int v = Pop (q);         // Lay v ra khoi Queue
        if (danhdao[v] != 1)// Neu v chua xet
        {
            cout<<v<<" ";      danhdao[v] = 1;
            for (i=1; i<=g.nV; i++)
                if (!danhdao[v] && g.mtke[v][i] != 0)
                    Push(q,v);
        }
    }
}
```

# Cài đặt BFS (tt)

- Đưa 1 vào Queue
- Lấy 1 ra xử lý, đưa 5, 4, 2 vào Queue
- Lấy 2 ra xử lý, đưa 5, 3 vào Queue
- Lấy 4 ra xử lý, đưa 5 vào Queue
- Lấy 5 ra xử lý, đưa 3 vào Queue
- Lấy 3 ra xử lý. Đưa 6 vào Queue
- Lấy 5 ra. Không xử lý (vì đã xử lý rồi)
- Lấy 5 ra. Không xử lý
- Lấy 3 ra. Không xử lý
- Lấy 6 ra xử lý. Không đưa gì vào Queue

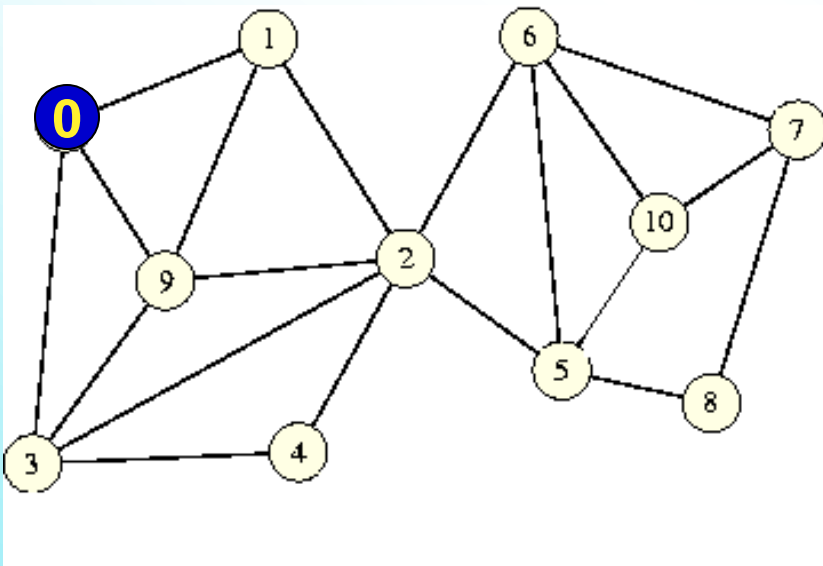


**Thứ tự duyệt:**

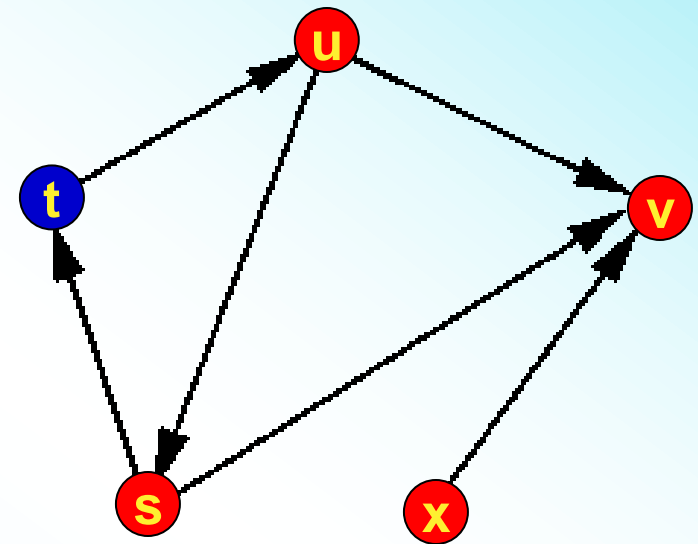


# Ví dụ về BFS

- Áp dụng BFS, hãy thể hiện thứ tự duyệt các đỉnh trong đồ thị sau:



**Đáp án: 0 1 3 9 2 4 5 6 8 10 7**



**Đáp án: t u s v**

**Đỉnh x không được duyệt**

### **3. Ứng dụng các thuật toán tìm kiếm trên đồ thị**

# Hàm DFS bằng đệ quy

- Do nguyên tắc gọi hàm đệ quy cũng giống như nguyên tắc hoạt động của Stack nên ta có thể dùng đệ quy thay cho Stack để viết hàm DFS
- Chú ý:
  - ◆ Mảng danhđau bắt buộc phải khai báo bên ngoài hàm đệ quy
  - ◆ Phần khởi tạo mảng danhđau cũng vẫn được thực hiện nhưng phải để ở bên ngoài hàm đệ quy (thường khởi tạo ở trong hàm main).

```
int danhđau[maxV]
void DFS(DOTHI g, int s)           // s là đỉnh xuất phát
{
    if (danhđau[s] == 1) return;
    cout<<s<<" đã được duyệt \n";           danhđau[s] = 1;
    for (int v = 1; v<=g.nV; v++)
        if (danhđau[v] == 0 && g.mtke[s][v] != 0)
            DFS(g,v);
}
```



# Áp dụng DFS để kiểm tra liên thông

## ■ Ý tưởng:

- ◆ Áp dụng cho đồ thị vô hướng
- ◆ Áp dụng DFS, bắt đầu từ đỉnh bất kỳ, nếu duyệt qua được tất cả các đỉnh thì đồ thị là liên thông
- ◆ Cụ thể:
  - Sử dụng thêm biến đếm để đếm số đỉnh được duyệt
  - Nếu duyệt xong mà đếm bằng  $g.nV$  (số đỉnh của đồ thị) thì có nghĩa là tất cả các đỉnh được duyệt

# Áp dụng DFS để kiểm tra liên thông (tt)

```
int danhdao[maxV]; int dem = 0
void DFS(DOTHI g, int s)    // s là đỉnh xuất phát
{
    if (danhdao[s] == 1) return;
    cout<<s<<" đã được duyệt \n";
    danhdao[s] = 1;
    for (int v = 1; v<=g.nV; v++)
        if (danhdao[v] == 0 && g.mtke[s][v] != 0)
        {
            ++dem;
            DFS(g,v);
        }
}
```

# Áp dụng DFS để kiểm tra liên thông (tt)

```
int isLienThong(DOTHI g)
{
    if (g.type == 1)
        return 0;           // không xét đồ thị có hướng
    dem = 1;
    for (int v = 1; v <= g.nV; v++)
        danh dau[v] = 0;
    DFS (g, 1, dem);
    if (dem == g.nV) return 1; // đồ thị liên thông
    return 0;                // đồ thị không liên thông
}
```