



# ỨNG DỤNG DFS BFS



## NỘI DUNG BÀI HỌC

01

Đếm số thành phần  
liên thông của đồ thị

...

02

Tìm đường đi giữa  
hai đỉnh trên đồ thị

...

03

Đỉnh trụ, cạnh cầu

...

04

Kiểm tra chu trình  
trên đồ thị

...

05

DFS, BFS trên  
mảng hai chiều

...

# 1. Đếm số thành phần liên thông của đồ thị:



Ứng dụng đầu tiên và đơn giản nhất của **DFS, BFS** là dùng để kiểm tra đồ thị liên thông hoặc đếm số thành phần liên thông của đồ thị.



Số thành phần liên thông của đồ thị chính là số lần gọi DFS hoặc BFS để có thể thăm hết mọi đỉnh trên đồ thị.

## Code

```
int n, m;
vector<int> adj[1005];
bool visited[1005];

int tplt(){
    int count = 0;
    for(int i = 1; i <= n; i++){
        if(!visited[i]){
            ++count;
            DFS(i); // BFS(i);
        }
    }
    return count;
}
```



## 2. Tìm đường đi giữa 2 đỉnh trên đồ thị:

**Đường đi giữa 2 đỉnh trên đồ thị** sẽ khác nhau tùy vào thuật toán được sử dụng là BFS hay DFS, nếu trên đồ thị không có trọng số thì đường đi giữa 2 đỉnh sử dụng thuật toán BFS sẽ là đường đi ngắn nhất. Để truy vết đường đi ta sử dụng thêm mảng `parent[ ]`



## 2. Tìm đường đi giữa 2 đỉnh trên đồ thị:

### Code

```
int n, m;
vector<int> adj[1005];
bool visited[1005];
int parent[1005];

void DFS(int u){
    visited[u] = true;
    for(int v : adj[u]){
        if(!visited[v]){
            DFS(v);
            parent[v] = u;
        }
    }
}

void BFS(int u){
    queue<int> q;
    q.push(u); visited[u] = true;
    while(!q.empty()){
        int u = q.front(); q.pop();
        for(int v : adj[u]){
            if(!visited[v]){
                q.push(v);
                parent[v] = u;
                visited[v] = true;
            }
        }
    }
}

void path(int s, int t){
    DFS(s); // BFS(s);
    if(!visited[t]){
        cout << "Khong ton tai duong di !\n";
    }
    else{
        vector<int> res;
        while(t != s){
            res.push_back(t);
            t = parent[t];
        }
        res.push_back(s);
        reverse(res.begin(), res.end());
        for(int x : res) cout << x << ' ';
    }
}
```

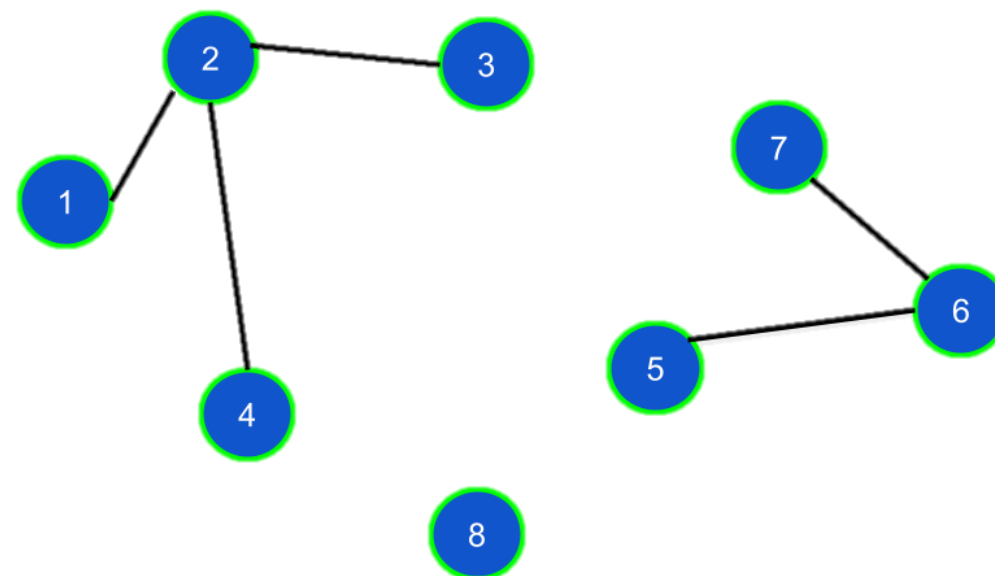
### 3. Đỉnh trụ, cạnh cầu:



**Đỉnh trụ:** Là đỉnh khi loại bỏ đỉnh này và các cạnh liên thuộc với đỉnh này sẽ làm tăng số thành phần liên thông của đồ thị.



Để kiểm tra đỉnh trụ bằng thuật toán DFS hoặc BFS cần độ phức tạp là  $O(V * (E + V))$ , bạn cần gọi V lần thuật toán DFS hoặc BFS.



**Ví dụ:** Các đỉnh trụ : 2, 6



### 3. Định trụ, cạnh cầu:

#### Code

```
int n, m;
vector<int> adj[1005];
bool visited[1005];

int tplt(){
    int count = 0;
    for(int i = 1; i <= n; i++){
        if(!visited[i]){
            ++count;
            DFS(i); // BFS(i)
        }
    }
    return count;
}
```

```
void dinh_tru(){
    int cc = tplt(); // so thanh phan lien thong ban dau
    for(int i = 1; i <= n; i++){
        memset(visited, false, sizeof(visited));
        visited[i] = true; // loai bo i khoi do thi
        if(cc < tplt()){
            cout << i << ' ';
        }
    }
}
```

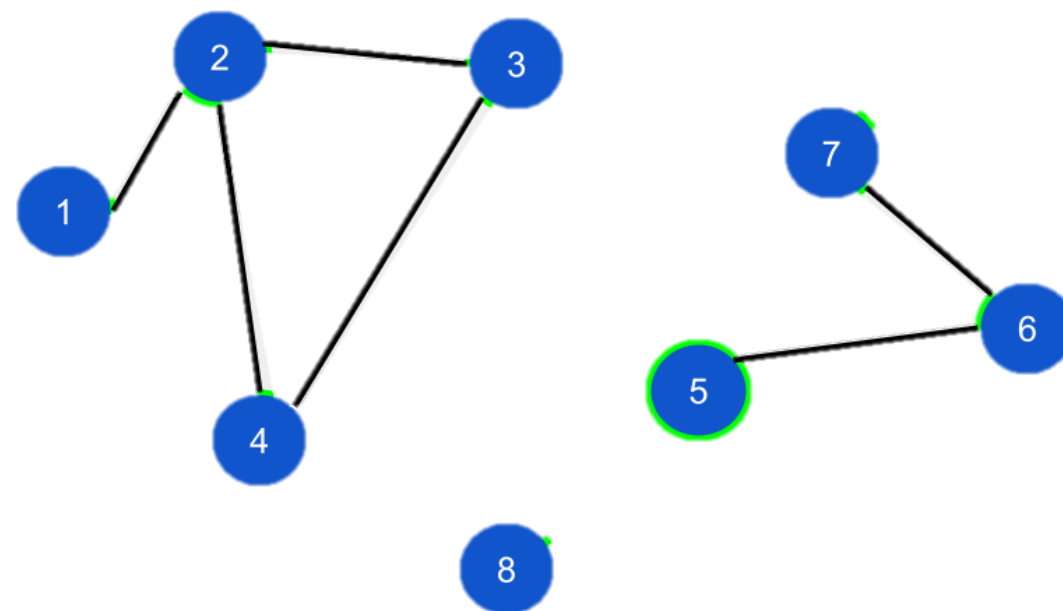
### 3. Đỉnh trụ, cạnh cầu:



**Cạnh cầu:** Những cạnh khi loại bỏ nó (chỉ loại bỏ cạnh, không loại bỏ 2 đỉnh) làm tăng số thành phần liên thông của đồ thị.



Để kiểm tra cạnh cầu bằng thuật toán DFS hoặc BFS cần độ phức tạp là  $O(E * (E + V))$ , bạn cần gọi E lần thuật toán DFS hoặc BFS.



**Ví dụ:** Cạnh cầu (1, 2), (5, 6), (6, 7)





### 3. Đỉnh trụ, cạnh cầu:

#### Code DFS không duyệt vào cạnh đã xóa

```
int n, m;
vector<int> adj[1005];
bool visited[1005];
vector<pair<int, int>> edge;

void DFS(int u, int s, int t){
    visited[u] = true;
    for(int v : adj[u]){
        //Neu xet phai canh muon loai bo thi khong xet
        if((u == s && v == t) || (u == t && v == s)){
            continue;
        }
        if(!visited[v]){
            DFS(v, s, t);
        }
    }
}

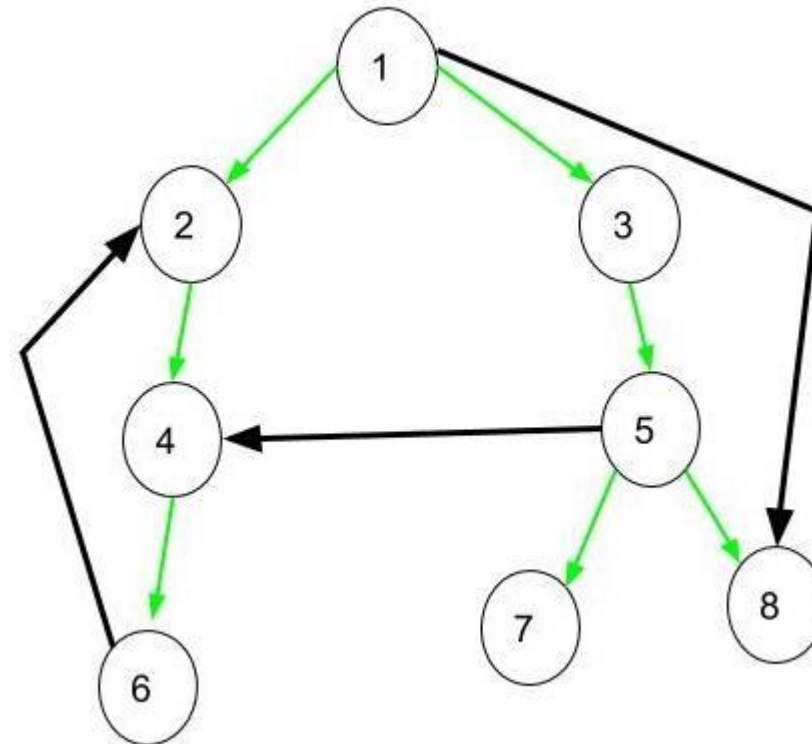
void canh_cau(){
    int cc = tplt();
    for(pair<int, int> e : edge){
        int s = e.first, t = e.second;
        memset(visited, false, sizeof(visited));
        if(cc < tplt()){
            cout << s << ' ' << t << endl;
        }
    }
}
```

Đối với khi xét cạnh cầu, việc loại bỏ cạnh ra khỏi đồ thị sẽ phức tạp hơn, bạn có thể sử dụng danh sách kề là set để có thể xóa nhanh hơn thay vì dùng vector, hoặc đánh dấu cạnh cần xóa và không duyệt vào cạnh này trong thuật toán DFS, BFS.

## 4. Kiểm tra chu trình trên đồ thị:



Để kiểm tra chu trình trên đồ thị ta kiểm tra trên đồ thị có cạnh ngược hay không.



Trên hình thì cạnh (6, 2) là cạnh ngược.



## 4. Kiểm tra chu trình trên đồ thị:

### Kiểm tra đồ thị vô hướng có chu trình bằng DFS

```
int n, m;
vector<int> adj[1005];
bool visited[1005];
int parent[1005];
bool DFS(int u){
    visited[u] = true;
    for(int v : adj[u]){
        if(!visited[v]){
            parent[v] = u;
            if(DFS(v))
                return true;
        }
        else if(v != parent[u]){
            return true;
        }
    }
    return false;
}
```

### Kiểm tra đồ thị vô hướng có chu trình bằng BFS

```
bool BFS(int u){
    queue<int> q;
    q.push(u);
    visited[u] = true;
    while(!q.empty()){
        int x = q.front(); q.pop();
        for(int y : adj[x]){
            if(!visited[y]){
                q.push(y);
                visited[y] = true;
                parent[y] = x;
            }
            else if(y != parent[x])
                return true;
        }
    }
    return false;
}
```



## 4. Kiểm tra chu trình trên đồ thị:

### Kiểm tra chu trình trên đồ thị có hướng bằng DFS

```
int n, m;
vector<int> adj[1005];
int color[1005];

bool DFS(int u){
    color[u] = 1;
    for(int v : adj[u]){
        if(color[v] == 0){
            if(DFS(v))
                return true;
        }
        else if(color[v] == 1){
            return true;
        }
    }
    color[u] = 2;
    return false;
}
```

### Kiểm tra chu trình trên đồ thị có hướng bằng BFS (Kahn)

```
int n, m;
vector<int> adj[1005];
int degree[1005];

bool Kahn(){
    for(int i = 1; i <= n; i++){
        for(int x : adj[i]){
            degree[x]++; // ban bac vao
        }
    }
    queue<int> q;
    for(int i = 1; i <= n; i++){
        if(degree[i] == 0){
            q.push(i);
        }
    }

    int cnt = 0;
    while(!q.empty()){
        int u = q.front(); q.pop();
        ++cnt;
        for(int v : adj[u]){
            --degree[v];
            if(degree[v] == 0){
                q.push(v);
            }
        }
    }
    if(cnt != n) return true;
    else return false;
}
```

## 5. DFS, BFS trên mảng hai chiều:



**DFS và BFS** có thể áp dụng được trên mảng 2 chiều, coi mỗi ô trên mảng 2 chiều là một đỉnh của đồ thị. 2 ví dụ dưới đây xét 8 ô chung đỉnh với ô hiện tại tương tự như là đỉnh kề.

### DFS trên mảng 2 chiều

```
int n, m;
int a[1005][1005];
bool visited[1005][1005];
int dx[8] = {-1, -1, -1, 0, 0, 1, 1, 1};
int dy[8] = {-1, 0, 1, -1, 1, -1, 0, 1};

void DFS(int i, int j){
    visited[i][j] = true;
    for(int k = 0; k < 8; k++){
        int i1 = i + dx[k], j1 = j + dx[k];
        if(i1 >= 1 && i1 <= n && j1 >= 1 && j1 <= m && !visited[i1][j1]){
            DFS(i1, j1);
        }
    }
}
```



## 5. DFS, BFS trên mảng hai chiều:

### BFS trên mảng 2 chiều

```
void BFS(int i, int j){
    queue<pair<int, int>> q;
    q.push({i, j});
    visited[i][j] = true;
    while(!q.empty()){
        pair<int, int> x = q.front(); q.pop();
        for(int k = 0; k < 8; k++){
            int i1 = x.first + dx[k], j1 = x.second + dx[k];
            if(i1 >= 1 && i1 <= n && j1 >= 1 && j1 <= m && !visited[i1][j1]){
                q.push({i1, j1});
                visited[i1][j1] = true;
            }
        }
    }
}
```