**Các thuật toán tìm kiếm trên đồ thị**

**1. Tìm kiếm theo chiều rộng (DFS - Depth First Search)**

**DFS** là thuật toán được sử dụng để tìm kiếm hay duyệt qua (thăm) tất cả các đỉnh  
thuộc các thành phần liên thông của đồ thị một cách có hệ thống với kỹ thuật gọi là ưu  
tiên độ sâu. Thuật toán có nhiều ứng dụng giải các bài toán về liên thông, cây khung,  
đường đi...Thuật toán có thể được cài đặt bằng phương pháp đệ qui hay phương pháp  
lặp sử dụng **cấu trúc stack** để khử đệ qui. Độ phức tạp của thuật toán là **O(n+m)** với n  
là số đỉnh và m là số cạnh của đồ thị.

**Ý tưởng :**

- Xuất phát từ một đỉnh **u** cho trước chưa thăm

- Thăm đỉnh **u** và tìm đỉnh kề **v** chưa được thăm. Nếu có tiếp tục duyệt từ **v** lặp lại quá trình này cho đến khi không còn đỉnh kề nào chưa được thăm.

- Nếu tại một đỉnh **u** nào đó không còn đỉnh kề chưa thăm quay trở lại đỉnh trước của **u** (đỉnh kề của u trên đường đến u) và tìm các đỉnh kề chưa thăm của đỉnh này

**1.1. Cài đặt thuật toán**

//duyet thuat toan DFS Danhsachke

void DFS(int u) {

temp.push\_back(u);

tham[u] = true;

for (int v : adj[u])

{

if (!tham[v])

DFS(v);

}

}

- Đầu tiên, đỉnh u được thêm vào danh sách kết quả **temp**, đánh dấu đỉnh này đã được thăm.

tham[u] = **true**: Đánh dấu đỉnh u là đã được thăm để tránh việc duyệt lại.

- Duyệt qua các đỉnh kề của u:

Với mỗi đỉnh v kề với u (trong danh sách kề adj[u]):

Nếu đỉnh v chưa được thăm

**Gọi đệ quy DFS**(v): Tiến hành duyệt theo chiều sâu từ đỉnh v.

- Thuật toán sẽ tiếp tục duyệt sâu vào từng đỉnh kề của mỗi đỉnh đệ quy cho đến khi không còn đỉnh nào để duyệt nữa.

**Lưu ý:**

- Ở đây hàm DFS được triển khai bằng cách sử dụng để quy để duyệt qua các đỉnh kề. Dẫn đến việc với đồ thị có **độ sâu lớn hoặc quá sâu** có thể gây tràn bộ nhớ stack có thể thay bằng cách sử vòng lặp hoặc cấu trúc dữ liệu stack tự tạo.

- DFS cũng cần được gọi từ mỗi đỉnh chưa được thăm để xác định tất cả các thành phần liên thông của đồ thị. Nếu không, chỉ một phần của đồ thị có thể được duyệt và các thành phần khác nhau sẽ không được xét.

- DFS theo chiều sâu, duyệt hết một nhánh trước khi chuyển sang nhánh khác. Điều này có thể dẫn đến việc đầu tiên tìm được đường đi sâu nhất trước khi đi theo nhánh khác.

- Duyệt qua các đỉnh kề của một đỉnh chưa được thăm trước khi di chuyển đến đỉnh kề khác. Điều này đảm bảo việc duyệt hết một nhánh trước khi quay lại để xét các nhánh khác.

- DFS có thể được áp dụng cho cả đồ thị có hướng và vô hướng, nhưng cần xác định cách duyệt đỉnh kề phù hợp với từng loại đồ thị.

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

Thuật toán **BFS** dùng để duyệt và tìm kiếm trên đồ thị cũng tương tự như thuật toán DFS nhưng thay vì dùng cấu trúc stack thì BFS dùng cấu trúc hàng đợi để thực hiện,  
kỹ thuật này gọi là tìm kiếm ưu tiên chiều rộng. Thuật toán có độ phức tạp là **O(m+n).**

**Ý tưởng:**

**-** Bắt đầu từ một đỉnh **u** cho trước chưa được thăm.

- Thăm đỉnh **u** và tìm tất cả các đỉnh kề chưa được thăm sau đó thêm chúng vào **hàng đợi**.

- Lặp lại quá trình này cho đến khi không còn đỉnh nào trong hàng đợi(**hàng đợi rỗng**).

- Duyệt qua các đỉnh kề chưa thăm của mỗi đỉnh trong **hàng đợi**, đánh dấu và thêm chúng vào hàng đợi và tiếp tục thăm các đỉnh kề chưa được thăm của đỉnh đó.

**2.1. Cài đặt thuật toán**

//duyet thuat toan BFS Danhsachke

void BFS(int u) {

queue<int> q;

q.push(u);

tham[u] = true;

while (!q.empty())

{

int v = q.front();

q.pop();

temp.push\_back(v);

for (int x : adj[v])

{

if (tham[x] == false)

{

q.push(x);

tham[x] = true;

}

}

}

}

- Đầu tiên, khởi tạo một **hàng đợi** q và thêm đỉnh u vào hàng đợi. Đỉnh này được coi là đỉnh bắt đầu.

queue<int> q;

q.push(u);

tham[u] = **true:** Đánh dấu đỉnh u là đã được thăm để tránh việc lặp lại.

- **Duyệt qua hàng đợi:**

Khi hàng đợi không rỗng:

Lấy đỉnh đầu tiên từ hàng đợi ( **int v = q.front()** ).

Sau đó loại bỏ đỉnh vừa lấy ra khỏi hàng đợi.

Thêm đỉnh v vào danh sách kết quả **temp**.

**- Duyệt qua các đỉnh kề của v:**

Với mỗi đỉnh x kề với v (trong danh sách kề adj[v]):

Nếu đỉnh x chưa được thăm (!tham[x]):

Thêm đỉnh x vào hàng đợi.

Đánh dấu đỉnh x là đã được thăm.

- Thuật toán BFS này duyệt qua từng đỉnh kề của các đỉnh đã được thăm, lưu trữ các đỉnh đã duyệt theo thứ tự rộng trước khi tiến hành duyệt đỉnh tiếp theo.

**Lưu ý:**

- Ở đây BFS được triển khai sử dụng một hàng đợi (**queue**) để duyệt qua các đỉnh kề theo chiều rộng. Việc này đảm bảo các đỉnh được thăm theo cấp bậc và không lặp lại.

- Nếu đồ thị không liên thông, cần gọi thuật toán BFS (**hoặc DFS**) từ mỗi đỉnh chưa được thăm để xác định các thành phần liên thông khác nhau của đồ thị.

- BFS có thể được áp dụng cho cả đồ thị có hướng và vô hướng, nhưng cần xác định cách duyệt đỉnh kề phù hợp với từng loại đồ thị.

- BFS duyệt theo chiều rộng, nghĩa là các đỉnh cùng cấp bậc sẽ được duyệt trước, sau đó mới duyệt các đỉnh cấp bậc cao hơn. Điều này tạo ra việc duyệt theo cấp bậc của đồ thị, từ đỉnh gốc đến các đỉnh xa hơn theo từng tầng.

**3. Ứng dụng 2 thuật toán BFS và DFS để kiểm tra tính liên thông của đồ thị**

//ktr tính lien thong

int ktrTinhLienThong(int n, int m) {

int cnt = 0;

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

{

if (!tham[i]) {

++cnt;

BFS(i);

}

}

return cnt;

}

**Ở đây,** ta sẽ sử dụng BFS để kiểm tra tính liên thông.

- Đầu tiên khởi tạo biến cnt để đếm số lượng thành phần liên thông trong đồ thị.

- Duyệt các đỉnh từ 1 đến n

* + Nếu đỉnh i chưa được thăm (!tham[i]) tức là chưa thuộc bất kỳ thành phần liên thông nào.
  + Tăng biến cnt để đếm số lượng thành phần liên thông
  + Gọi hàm BFS(i) để duyệt từ đỉnh i.Mục tiêu là tìm tất cả các đỉnh thuộc cùng một thành phần liên thông với i.
  + Cuối cùng là trả về kết quả cuối là số lượng thành phần liên thông của đồ thị.

- Nếu số lượng thành phần liên thông là 1, tức là chỉ cần một lần duyệt từ một đỉnh bất kỳ để có thể duyệt hết các đỉnh khác, đồ thị là liên thông.

- Ngược lại, nếu có nhiều hơn một thành phần liên thông, đồ thị không liên thông và sẽ có số thành phần liên thông được xuất ra để thể hiện.