Chương 3:TÌM KIẾM TRÊN ĐỒ THỊ

- Giải thuật DFS và BFS

- Ứng dụng tìm kiếm trên đồ thị: tìm đường đi giữa các đỉnh trên đồ thị; tính liên thông đồ thị.

- Đồ thị Euler và Hamilton

Giới thiệu

Duyệt đồ thị theo chiều sâu

- Giới thiệu
  - Duyệt đồ thị là quá trình đi qua tất cả các đỉnh của đồ thị sao cho mỗi đỉnh của nó được viếng thăm đúng một lần.
  - Duyệt theo chiều sâu (Depth First Search – DFS)
  - Duyệt theo chiều rộng (Breadth First Search – BFS)

Giới thiệu

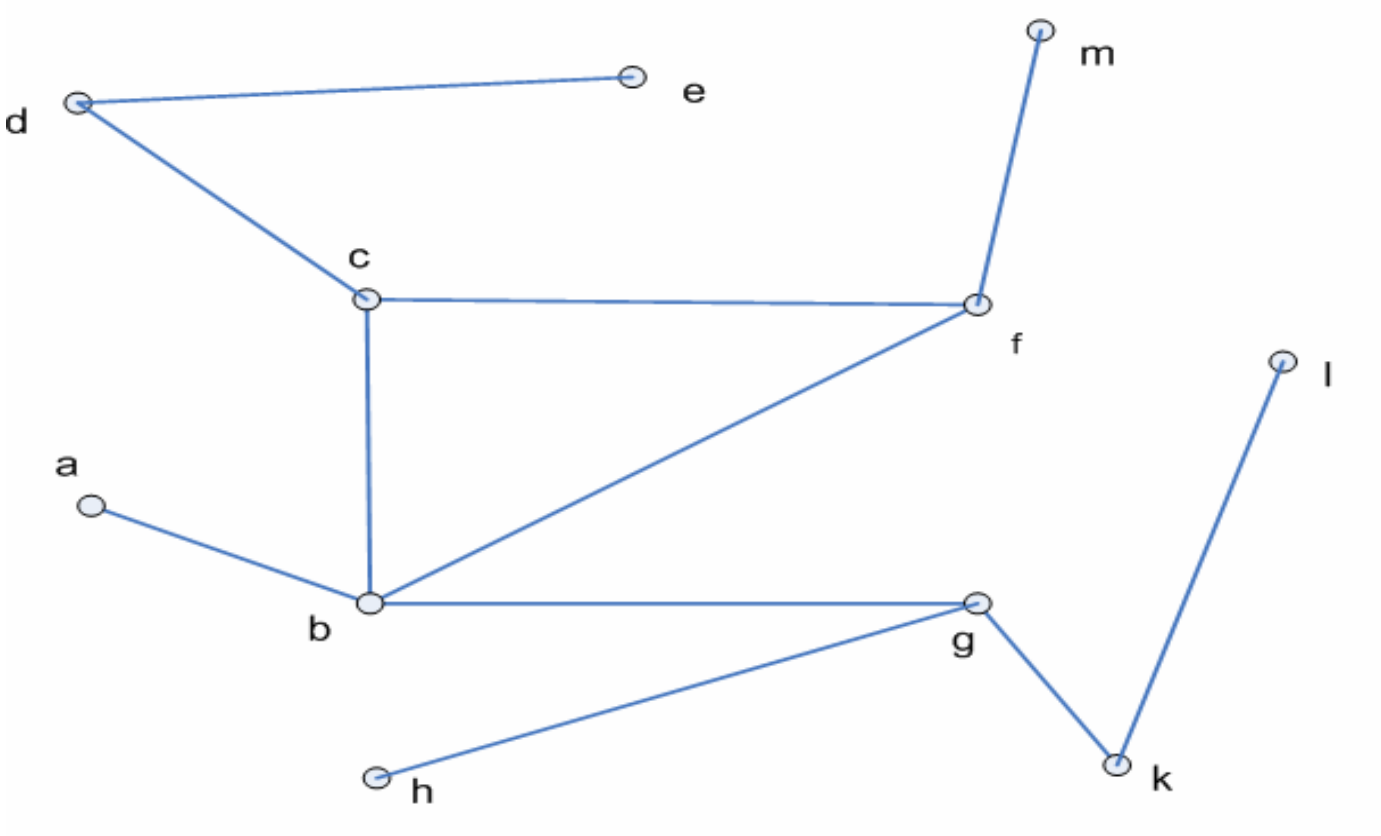
Duyệt đồ thị theo chiều sâu

Nguyên lý

- Bắt đầu tìm kiếm từ một đỉnh v nào đó của đồ thị.
- Sau đó chọn u là một đỉnh tùy ý kề với v (với đồ thị có hướng thì u là đỉnh sau, v là đỉnh đầu của cung uv)
- Lặp lại quá trình này với u cho đến khi không tìm được đỉnh kề tiếp theo nữa thì trở về đỉnh ngay trước đỉnh mà không thể đi tiếp để tìm qua nhánh khác.

➢ Giới thiệu

Duyệt đồ thị theo chiều sâu



Thứ tự duyệt:

d c b a

g k l

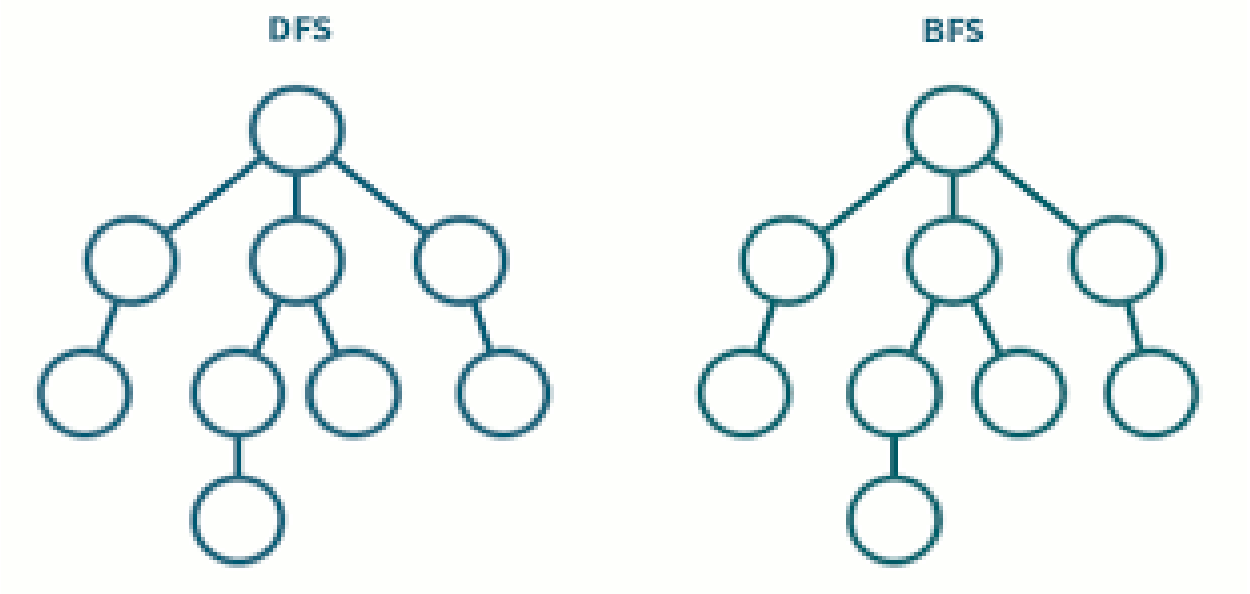
h

f m

e

➢ Giới thiệu

- Sự khác biệt giữa BFS và DFS: Tìm kiếm theo chiều rộng (BFS) là một kỹ thuật dựa trên đỉnh giúp chỉ ra đường đi ngắn nhất trong biểu đồ.
- DFS (Tìm kiếm theo chiều sâu) là một kỹ thuật dựa trên cạnh. DFS phụ thuộc vào cấu trúc dữ liệu **Stack (LIFO)**.
- BFS là một kỹ thuật phụ thuộc vào cấu trúc dữ liệu **hàng đợi (FIFO)**.



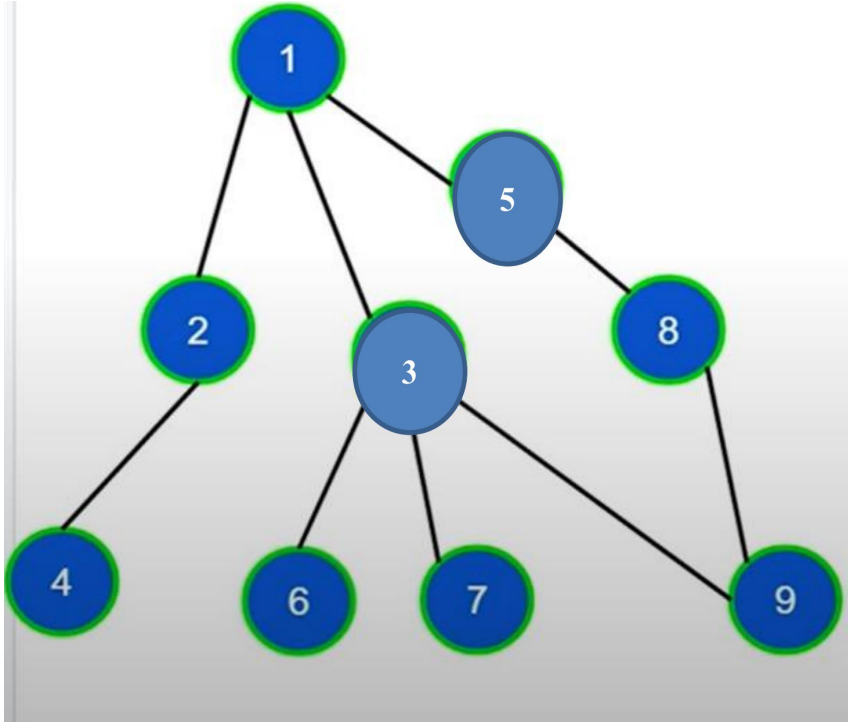
Tư tưởng giải thuật

DFS : Depth First Search

Thuật toán tìm kiếm theo chiều sâu : Ưu tiên duyệt xuống nhất có thể trước khi quay lại

Pseudocode :

```
//Bat dau duyet tu dinh u
DFS(u){
        <Tham Dinh u>;
        visited[u] = true; // Danh dau la u da duoc tham
        //Duyet cac dinh ke voi dinh u
        for(v : adj[u]){
                if(!visited[v]){ // Neu v chua duoc tham
                        DFS(v);
                }
        }
}
```



Độ phức tạp của thuật toán DFS phụ thuộc vào cách biểu diễn ma trận:

Đồ thị G = < V, E >

Biểu diễn bằng ma trận kề : O(V * V)

Biểu diễn bằng danh sách cạnh : O(V * E)

Biểu diễn bằng danh sách kề : O(V + E)

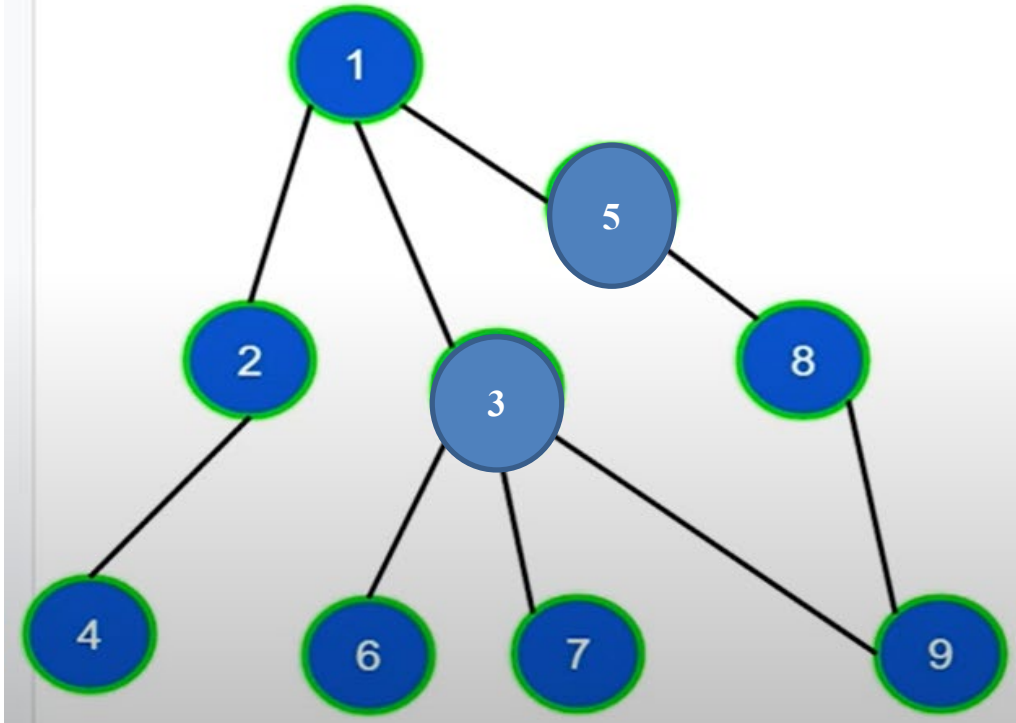
Biểu diễn thuật toán DFS

Ví dụ:

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1)



Stack

Đỉnh	Visited
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

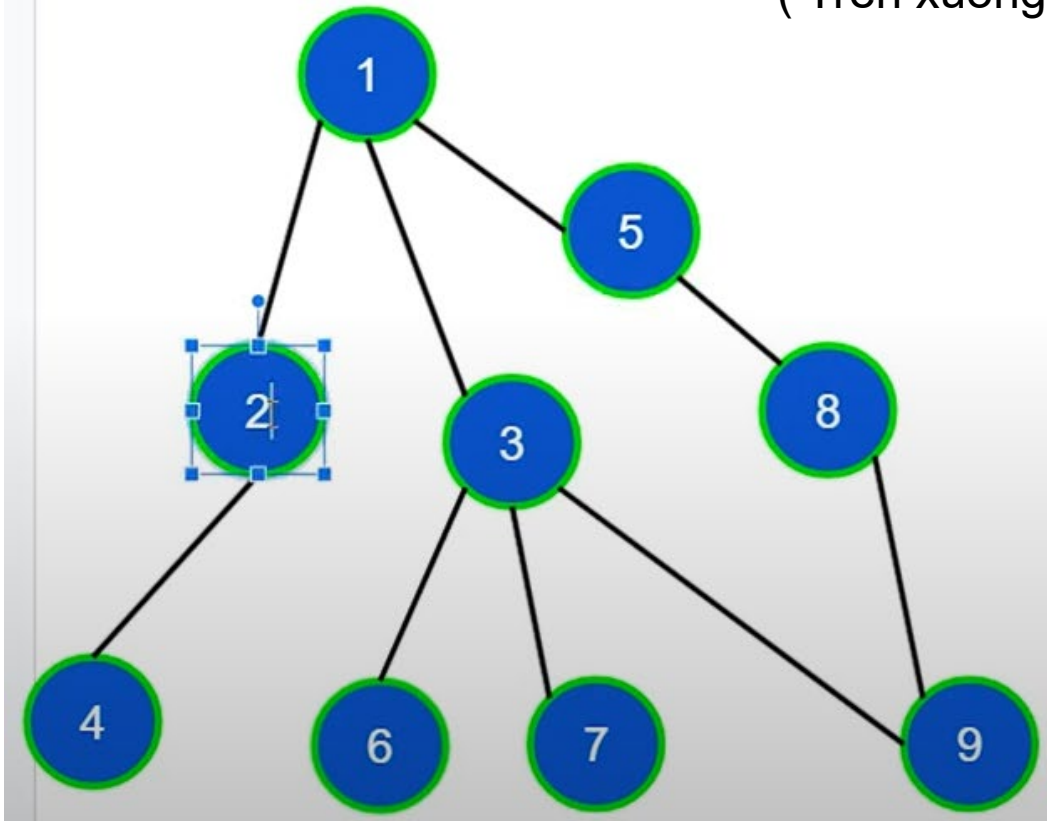
Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1,

1/ Bắt đầu duyệt đỉnh 1 (DFS 1) => đẩy 1 vào stack, Visited 1 sẽ true => tiếp theo duyệt các đỉnh kề với đỉnh 1 theo chiều sâu (Trên xuống, trái qua phải)



Stack

1

Đỉnh	Visited
1	True
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

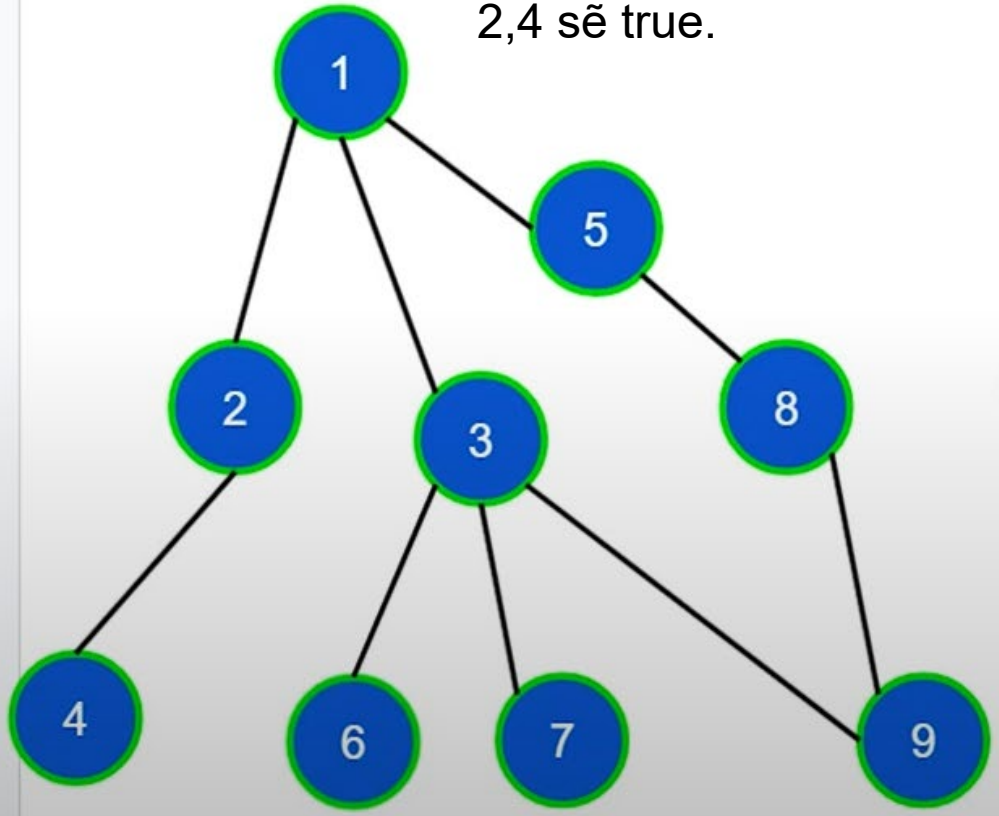
Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1, 2, 4

2/ Thăm đỉnh 2 => duyệt các đỉnh kề 2 (DFS2) -> đẩy 2 vào stack (có đỉnh 1 đã duyệt (true), còn lại 4), vị trí 2,4 sẽ true.



Stack

4
2
1

Đỉnh	Visited
1	True
2	True
3	False
4	True
5	False
6	False
7	False
8	False
9	False

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

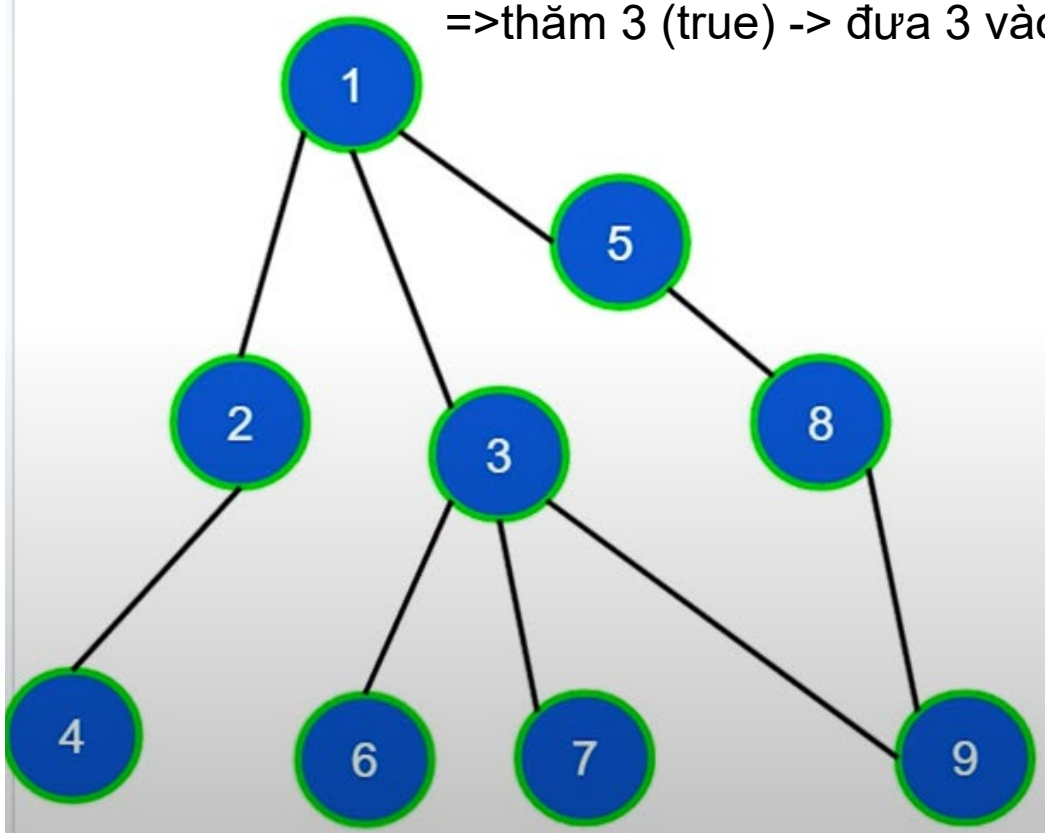
DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1, 2, 4, 3

3/ Sau khi Thăm đỉnh 4 => duyệt các đỉnh kề 4 (**DFS 4**) -> đẩy 4 vào stack. Đỉnh 4 có đỉnh 2 đã duyệt (true). -> Đỉnh 4 hết nhánh => (**OFF 4**), quay lại 2, kề 2 có 1 và 4 đã thăm =>(**OFF 2**),

4/ Quay lại 1 => kề 1 có 2 (đã thăm true); 3 và 5 (chưa thăm false) =>thăm 3 (true) -> đưa 3 vào stack (**DFS 3**) (do duyệt chiều sâu)



Stack

Đỉnh	Visited
1	True
2	True
3	True
4	True
5	False
6	False
7	False
8	False
9	False

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

5/ Duyệt tại 3 có liền kề 1 (đã thăm) còn 6,7,9 chưa thăm => đưa 6 vào stack => đánh dấu true => thăm 6 (**DFS 6**); xét 6 có thành phần liền kề là 3 (đã thăm (true) và 6 là đỉnh cuối => **OFF 6** khỏi stack => quay lại 3

6/ Đỉnh 3 có thành phần liền kề 1,6 (đã thăm true), còn lại 7 và 9 (chưa thăm (false) -> Thăm 7 (đánh dấu true) ->, (**DFS 7)** => 7 đỉnh cuối => **0FF 7** khỏi stack => quay lại 3

7/ Kề 3 có 9 chưa thăm (false) -> thăm 9 (true) -> đẩy 9 vào stack (**DFS 9**)

8/ Thăm 9, kề đỉnh 9 là 3 (true) và 8 Chưa thăm (false) -> thăm đỉnh 8 (true) => đẩy 8 vào stack **(DFS 8)**.

9/ Xét 8, kề 8 là 5 (chưa thăm false) và 9 (true) => thăm 5 (true) -> đẩy 5 vào stack **DFS 5**. Xét 5, không còn đỉnh duyệt (**OFF 5**). Quay lại 8 (**OFF 8**). Quay lại 9 **(OFF 9**). Quay lại 3 (**OFF 3**). Quay lại 1 (**OFF 1**)

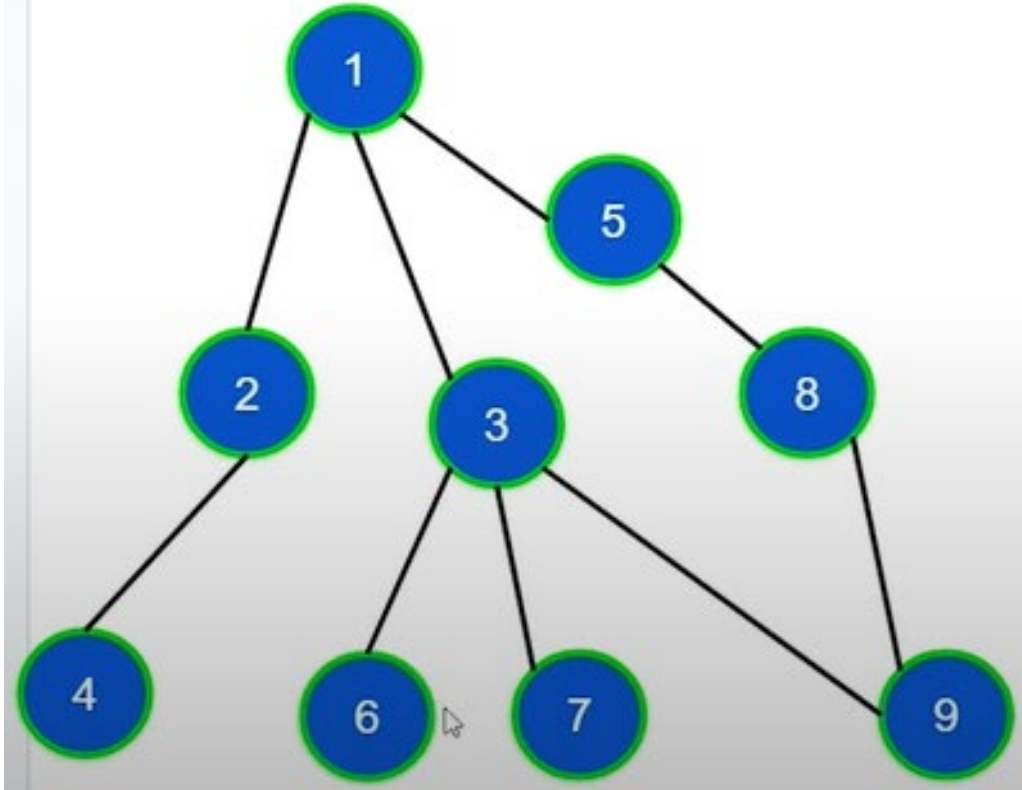
Thứ tự OFF khỏi stack: 4 => 2 => 6 =>7 => 5 => 8 => 9 =>3 => 1

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1, 2, 4, 3, 6



Stack

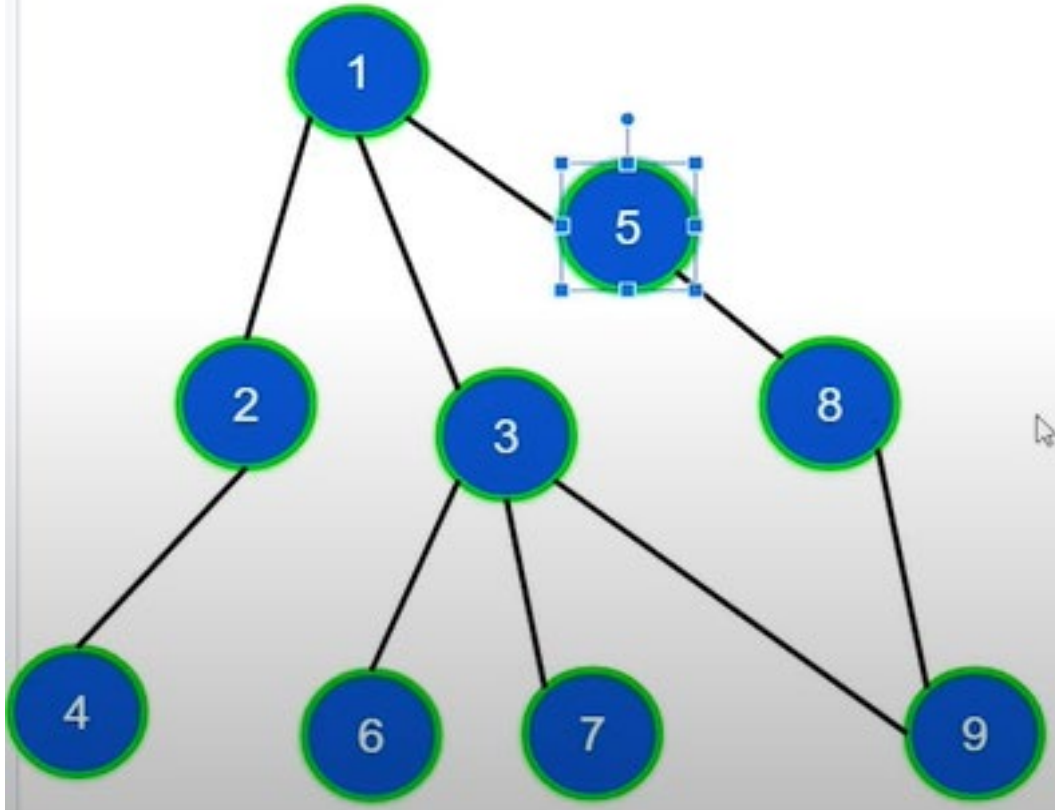
Đỉnh	Visited
1	True
2	True
3	True
4	True
5	False
6	True
7	False
8	False
9	False

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1, 2, 4, 3, 6, 7, 9, 8, 5



Stack

5
8
9
3
1

Đỉnh	Visited
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

```cpp
#include <bits/stdc++.h>
using namespace std;

//Input
//9 9
//1 2
//1 3
//1 5
//2 4
//3 6
//3 7
//3 9
//5 8
//8 9

int n, m;
vector<int> adj[1001];
bool visited[1001];

void inp(){
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int x, y; cin >> x >> y;
```

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

```
24          adj[x].push_back(y);
25          adj[y].push_back(x);
26      }
27      memset(visited, false, sizeof(visited));
28  }
29
30  void dfs(int u){
31      cout << u << " ";
32      //Danh dau la u da duoc tham
33      visited[u] = true;
34      for(int v : adj[u]){
35          //Neu dinh v chua duoc tham
36          if(!visited[v]){
37              dfs(v);
38          }
39      }
40  }
41
42  int main(){
43      inp();
44      dfs(1);
45  }
```

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

```
9 9
1 2
1 3
1 5
2 4
3 6
3 7
3 9
5 8
8 9
1 2 4 3 6 7 9 8 5
--------------------------------
Process exited after 1.726 seconds with return value 0
Press any key to continue . . .
```

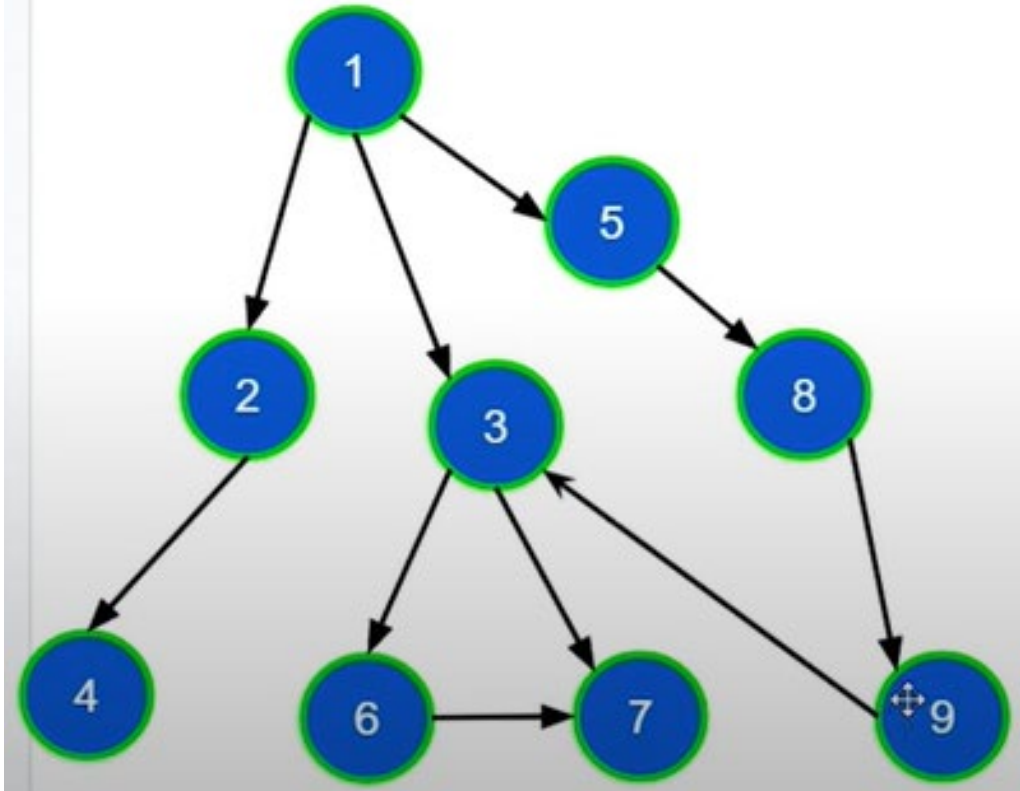
Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

Ví dụ: Thuật toán DFS cho đồ thị **có hướng**

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) =



Stack

Đỉnh	Visited
1	False
2	False
3	False
4	False
5	False
6	False
7	False
8	False
9	False

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

Ví dụ: Thuật toán DFS cho đồ thị **có hướng**

1/ Ban đầu tất cả các đỉnh đặt False (chưa thăm)

2/ Bắt đầu thăm đỉnh 1 -> đánh dấu đỉnh 1 (visited: True) -> đẩy đỉnh 1 vào stack bằng hàm **DFS (1)**.

3/ Xét đỉnh 1 có 2,3,5 kề nhưng do duyệt theo DFS nên thăm đỉnh 2 trước -> đỉnh 2 (true) -> đẩy 2 vào stack bằng hàm **DFS (2)**

4/ Xét đỉnh 2, kề 2 có 4 (do có hướng) -> thăm 4 (đánh dấu true) -> đẩy đỉnh 4 vào stack **DFS (4)**

5/ Xét đỉnh 4, không còn đỉnh nào kề 4 mà chưa thăm, nên **OFF (4)** khỏi stack -> Back track 2, xét đỉnh 2, kề 2 chỉ có đỉnh 1 (đã được thăm True), nên đỉnh 2 không còn đỉnh nào kề 2 mà chưa duyệt -> **OFF (2)** khỏi stack.

6/ Quay lại 1 -> còn 3 chưa thăm -> thăm 3 -> đánh dấu 3 (true) -> đẩy 3 vào stack -> **DFS (3)**.

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

Ví dụ: Thuật toán DFS cho đồ thị **có hướng**

7/ Duyệt 3, kề 3 có 6 và 7 -> Thăm 6 trước -> đánh dấu 6 (true) -> đẩy 6 vào stack -> **DFS (6)**

8/ Xét 6, kề 6 có 7 -> thăm 7 -> đánh dấu 7 true –> đẩy 7 vào Stack -> **DFS (7)**

9/ Xét 7, không còn đỉnh nào kề 7 mà chưa thăm -> **OFF (7)**

10/ Quay lại 6, không còn đỉnh kề nào chưa thăm **-> OFF (6).**

11/ Quay lại 3, kề 3 có 6,7 (đã thăm) -> nên 3 không còn đỉnh kề nào chưa thăm nên **OFF (3)** khỏi stack -> Quay lại đỉnh 1

12/ Xét đỉnh 1 có 2,3 đã thăm (true) -> còn lại đỉnh 5 chưa thăm -> thăm đỉnh 5 (đánh dấu true) -> đẩy đỉnh 5 vào stack **DFS (5)**.

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

Ví dụ: Thuật toán DFS cho đồ thị **có hướng**

13/ Duyệt đỉnh 5, kề đỉnh 5 còn đỉnh 8 chưa thăm (false) -> thăm đỉnh 8 (đánh dấu true) -> đẩy đỉnh 8 vào stack **DFS (8).**

14/ Duyệt đỉnh 8, kề đỉnh 8, có đỉnh 9 chưa duyệt (false) -> Thăm đỉnh 9 (đánh dấu true) –> đẩy đỉnh 9 vào stack (**DFS (9)**

15/ Xét đỉnh 9, kề đỉnh 9 là đỉnh 3, mà 3 được thăm (true), nên đỉnh 9 không còn đỉnh nào kề nên rút khỏi stack **OFF 9**.

16/ Quay về đỉnh 8, đỉnh 8 cũng không còn đỉnh kề nào chưa duyệt **OFF 8** khỏi stack.

17/ Tương tự, Quay về đỉnh 5, **OFF5**.

18/ Quay về đỉnh 1, **OFF 1**.

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

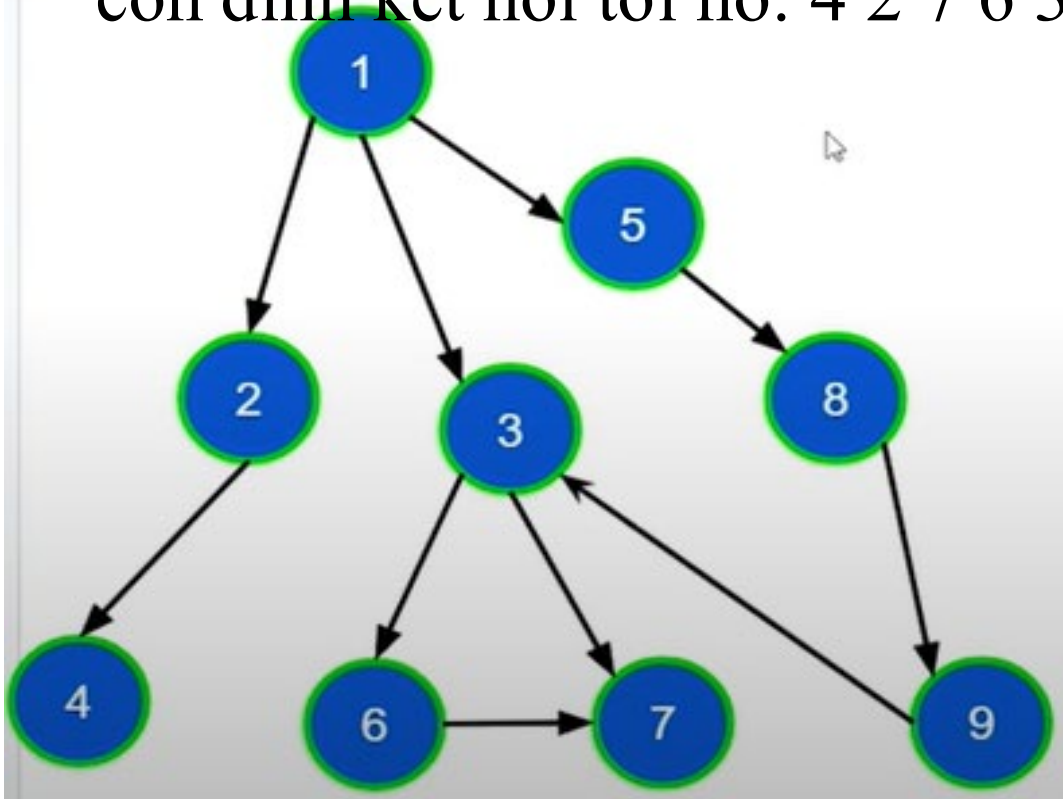
Kết quả DFS cho đồ thị trên:

DFS : Depth First Search

Kiểm nghiệm thuật toán DFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

DFS (1) = 1, 2, 4, 3, 6, 7, 5, 8, 9

Kết quả OFF khỏi stack khi duyệt xong 1 đỉnh mà không còn đỉnh kết nối tới nó: 4 2 7 6 3 9 8 5 1



Stack

Đỉnh	Visited
1	True
2	True
3	True
4	True
5	True
6	True
7	True
8	True
9	True

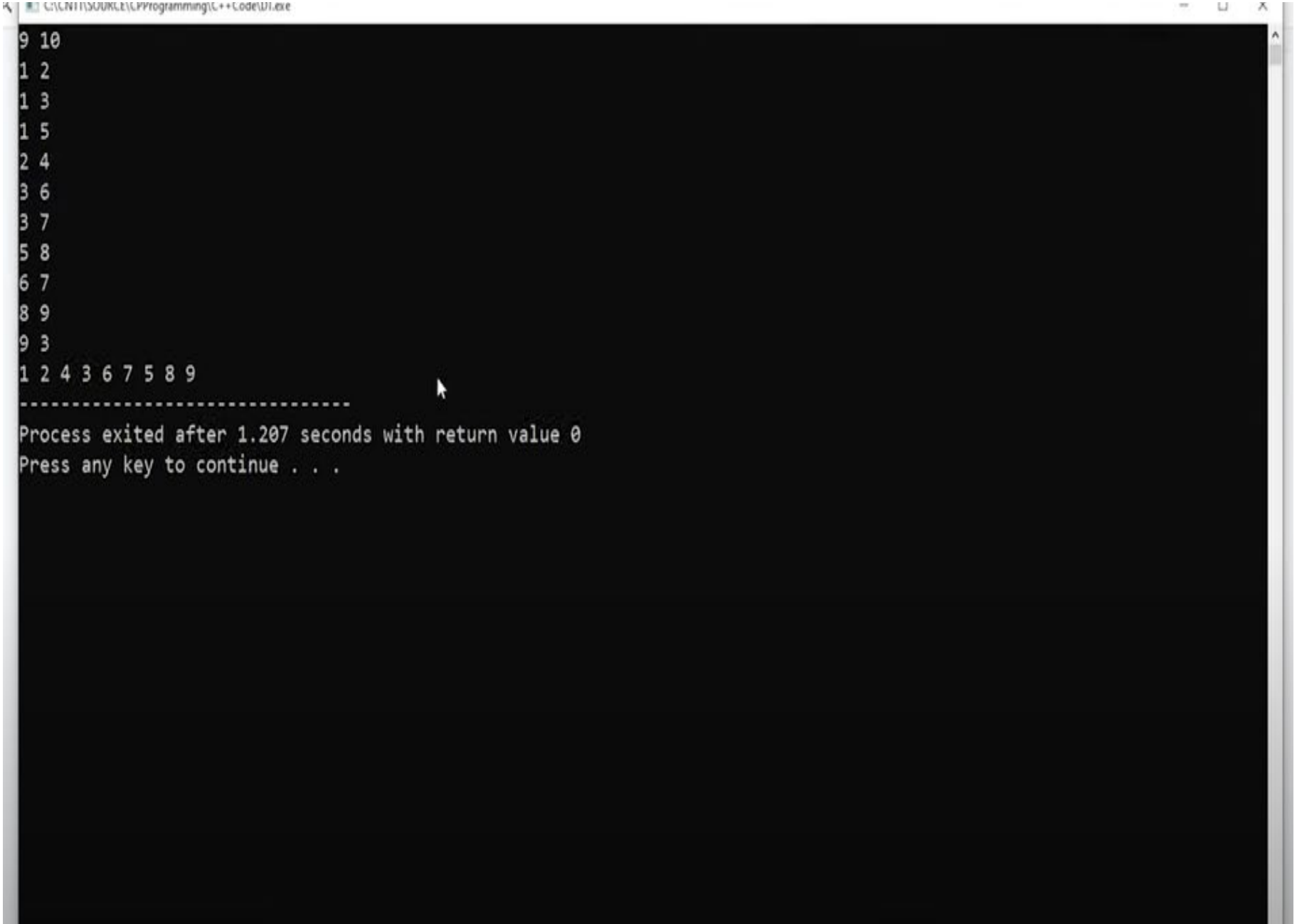
Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

```cpp
#include <bits/stdc++.h>
using namespace std;

//Input
//9 10
//1 2
//1 3
//1 5
//2 4
//3 6
//3 7
//5 8
//6 7
//8 9
//9 3
int n, m;
vector<int> adj[1001];
bool visited[1001];

void inp(){
    cin >> n >> m;
    for(int i = 0; i < m; i++){
        int x, y; cin >> x >> y;
        adj[x].push_back(y);
        //adj[y].push_back(x);
    }
```

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

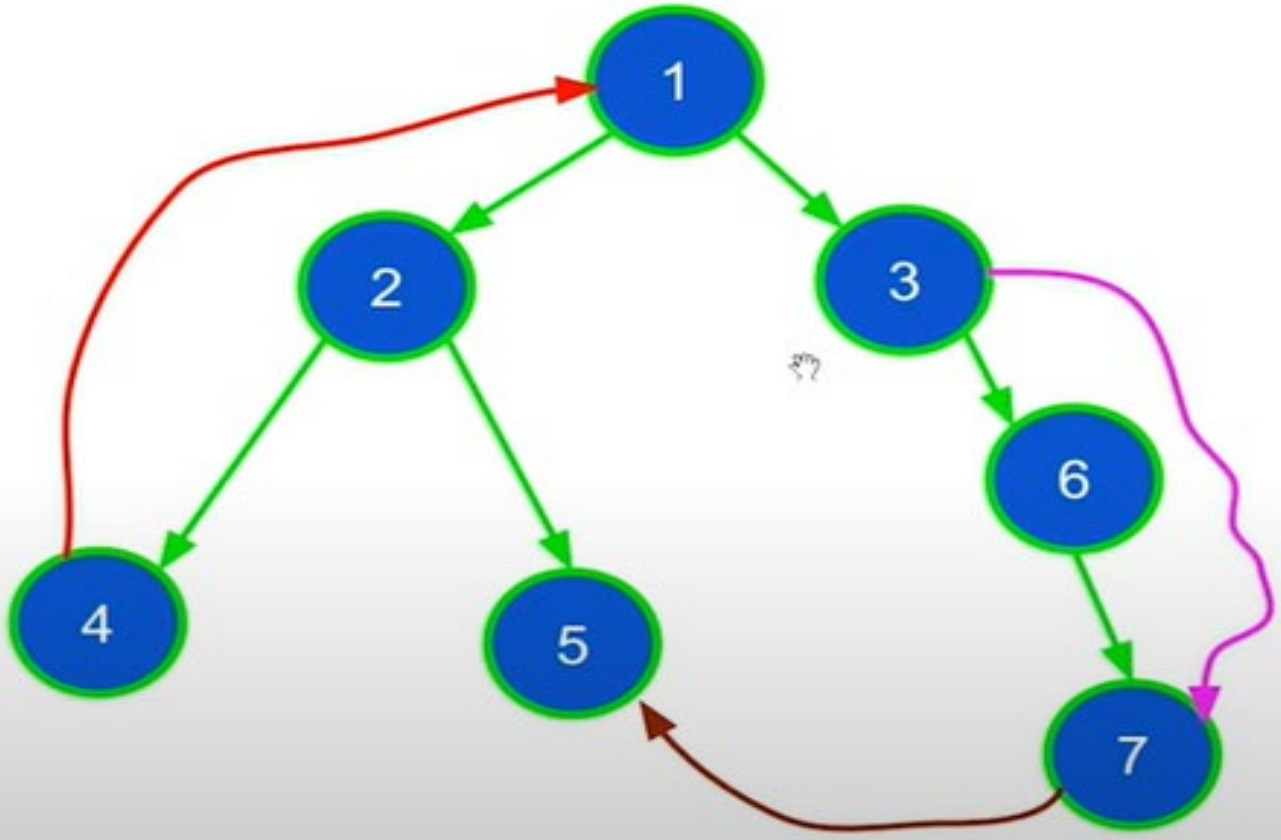
```
26      }
27      memset(visited, false, sizeof(visited));
28  }
29
30  void dfs(int u){
31      cout << u << " ";
32      //Danh dau la u da duoc tham
33      visited[u] = true;
34      for(int v : adj[u]){
35          //Neu dinh v chua duoc tham
36          if(!visited[v]){
37              dfs(v);
38          }
39      }
40  }
41
42  int main(){
43      inp();
44      dfs(1);
45  }
46
47
```

Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)



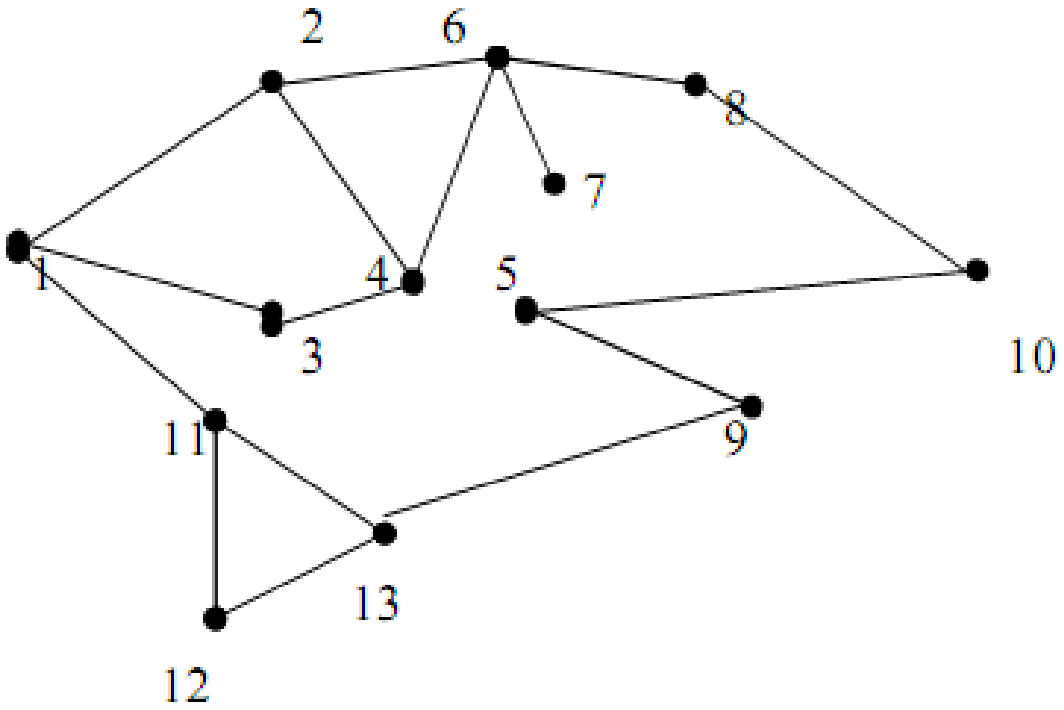
Thuật toán DFS(u) dùng ngăn xếp (khử đệ qui)

Các loại cạnh trong cây tìm kiếm DFS
Tree Edge : là cạnh mà theo đó từ một đỉnh ta đến thăm một đỉnh mới
Back Edge : Cạnh ngược là cạnh đi từ con cháu (descendant) đến tổ tiên (ancestor)
Forward Edge : Cạnh tới là cạnh đi từ tổ tiên tới hậu duệ
Cross Edge : Cạnh vòng là cạnh nối 2 đỉnh không có quan hệ họ hàng



VD: Kiểm nghiệm thuật toán DFS (1/3)

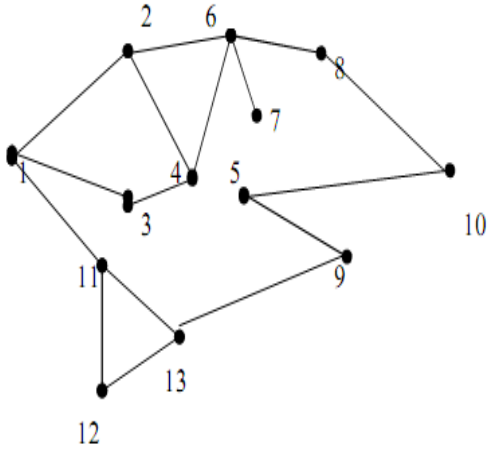
- **Ví dụ 1:** Cho đồ thị gồm 13 đỉnh như hình vẽ. Hãy kiểm nghiệm thuật toán DFS(1).



Kiểm nghiệm thuật toán DFS (2/3)

Đỉnh bắt đầu duyệt	Các đỉnh đã duyệt: chuaxet[u]=False	Các đỉnh chưa duyệt chuaxet[u]=True
DFS(1)	1	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(2)	1, 2	3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(4)	1, 2, 4	3, 5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(3)	1,2,4, 3	5, 6, 7, 8, 9, 10, 11, 12, 13
DFS(6)	1,2,4,3, 6	5, 7, 8, 9, 10, 11, 12, 13
DFS(7)	1,2,4,3, 6,7	5, 8, 9, 10, 11, 12, 13
DFS(8)	1,2,4,3, 6,7,8	5, 9, 10, 11, 12, 13
DFS(10)	1,2,4,3, 6,7,8,10	5, 9, 11, 12, 13
DFS(5)	1,2,4,3, 6,7,8,10,5	9, 11, 12, 13
DFS(9)	1,2,4,3, 6,7,8,10,5,9	11, 12, 13
DFS(13)	1,2,4,3, 6,7,8,10,5,9,13	11, 12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11	12
DFS(11)	1,2,4,3, 6,7,8,10,5,9,13,11,12	ϕ

Kết quả duyệt: ***1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12***



Kiểm nghiệm thuật toán DFS (3/3)

STT	Trạng thái ngăn xếp	Danh sách đỉnh được duyệt
1	1	1
2	1, 2	1, 2
3	1, 2, 4	1, 2, 4
4	1, 2, 4, 3	1, 2, 4, 3
5	1, 2, 4	1, 2, 4, 3
6	1, 2, 4, 6	1, 2, 4, 3, 6
7	1, 2, 4, 6, 7	1, 2, 4, 3, 6, 7
8	1, 2, 4, 6	1, 2, 4, 3, 6, 7
9	1, 2, 4, 6, 8	1, 2, 4, 3, 6, 7, 8
10	1, 2, 4, 6, 8, 10	1, 2, 4, 3, 6, 7, 8, 10
11	1, 2, 4, 6, 8, 10, 5	1, 2, 4, 3, 6, 7, 8, 10, 5
12	1, 2, 4, 6, 8, 10, 5, 9	1, 2, 4, 3, 6, 7, 8, 10, 5, 9
13	1, 2, 4, 6, 8, 10, 5, 9, 13	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13
14	1, 2, 4, 6, 8, 10, 5, 9, 13, 11	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11
15	1, 2, 4, 6, 8, 10, 5, 9, 13, 11, 12	1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12
16-	Lần lượt bỏ các đỉnh ra khỏi ngăn xếp	

Kết quả duyệt: 1, 2, 4, 3, 6, 7, 8, 10, 5, 9, 13, 11, 12

Ví dụ 2

- Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình bên phải. Hãy cho biết kết quả thực hiện thuật toán trong DFS bắt đầu tại đỉnh u=1? Chỉ rõ trạng thái của ngăn xếp và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

Ví dụ 2: Kiểm nghiệm thuật toán DFS(1)

STT	Trạng thái stack	Các đỉnh được duyệt
1	1	1
2	1, 2	1, 2
3	1, 2, 3	1, 2, 3
4	1, 2, 3, 4	1, 2, 3, 4
5	1, 2, 3, 4, 7	1, 2, 3, 4, 7
6	1, 2, 3, 4, 7, 5	1, 2, 3, 4, 7, 5
7	1, 2, 3, 4, 7, 5, 6	1, 2, 3, 4, 7, 5, 6
8	1, 2, 3, 4, 7, 5, 6, 12	1, 2, 3, 4, 7, 5, 6, 12
9	1, 2, 3, 4, 7, 5, 6, 12, 8	1, 2, 3, 4, 7, 5, 6, 12, 8
10	**1, 2, 3, 4, 7, 5, 6, 12, 10**	1, 2, 3, 4, 7, 5, 6, 12, 8, 10
11	1, 2, 3, 4, 7, 5, 6, 12, 10, 9	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9
12	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11
13	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11, 13	1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11, 13
14	∅	

Kết quả duyệt DFS(1) = { 1, 2, 3, 4, 7, 5, 6, 12, 8, 10, 9, 11, 13}.

❖ Cài đặt thuật toán

- Hàm Init(): đọc dữ liệu theo khuôn dạng từ file dothi.in và thiết lập mảng chuaxet[u] =True (u=1, 2,..,n).
- Hàm DFS_Dequi: Cài đặt thuật toán DFS(u) bằng đệ qui.
- Hàm DFS_Stack: Cài đặt thuật toán DFS(u) dựa vào stack.

dothi.in

```
10
0  1  1  1  0  0  0  0  0  0
1  0  0  1  1  0  0  0  0  0
1  0  0  1  0  1  0  0  0  0
1  1  1  0  1  1  0  0  1  0
0  1  0  1  0  0  0  1  0  0
0  0  1  1  0  0  1  0  0  0
0  0  0  0  0  1  0  0  1  1
0  0  0  0  1  0  0  0  1  1
0  0  0  1  0  0  1  1  0  1
0  0  0  0  0  0  1  1  1  0
```

DFS(3) = 3, 1, 2, 4, 5, 8, 9, 7, 6, 10.

Xem code minh họa

➢ Thuật toán tìm kiếm theo chiều rộng (Breadth First Search)

Tư tưởng:

- ❖ Trong quá trình tìm kiếm, ưu tiên "chiều rộng" hơn "chiều sâu"
- ❖ Tìm kiếm xung quanh trước khi đi xuống sâu hơn

Đỉnh được nạp vào hàng đợi đầu tiên là u.

- ❖ Các đỉnh kề với u là (v_1, v_2, . . ., v_k) **được nạp vào hàng đợi nếu như nó chưa được xét đến**.
- ❖ Quá trình duyệt tiếp theo được bắt đầu từ các đỉnh còn có mặt trong hàng đợi.
- ❖ **Thuật toán dừng khi hàng đợi rỗng.**

➢ Thuật toán BFS

BFS : Breadth First Search : Tư tưởng của thuật toán BFS là tìm kiếm ưu tiên chiều rộng hơn là chiều sâu

Thuật toán sẽ tìm kiếm xung quanh để mở rộng trước khi đi xuống sâu hơn

Pseudocode :

```
BFS(u){
    //Step 1 : Khởi tạo
    queue = Ø; // Tạo một hàng đợi rỗng
    push(queue, u); // Đẩy đỉnh u vào hàng đợi
    visited[u] = true; // Đánh dấu là đỉnh u đã được thăm
    //Step 2 : Lặp khi mà hàng đợi vẫn còn phần tử
    while( queue != Ø){
        v = queue.front(); // Lấy ra đỉnh ở đầu hàng đợi
        queue.pop(); // Xóa đỉnh khỏi đầu hàng đợi
        <Thăm đỉnh v >;
        //Duyệt các đỉnh kề với v mà chưa được thăm và đẩy vào hàng đợi
        for(int x : ke[v]){
            if( !visited[x]){ // Nếu x chưa được thăm
                push(queue, x);
                visited[x] = true
            }
        }
    }}
```

In ra đỉnh này (BFS [u]

In ra đỉnh này BFS [v]

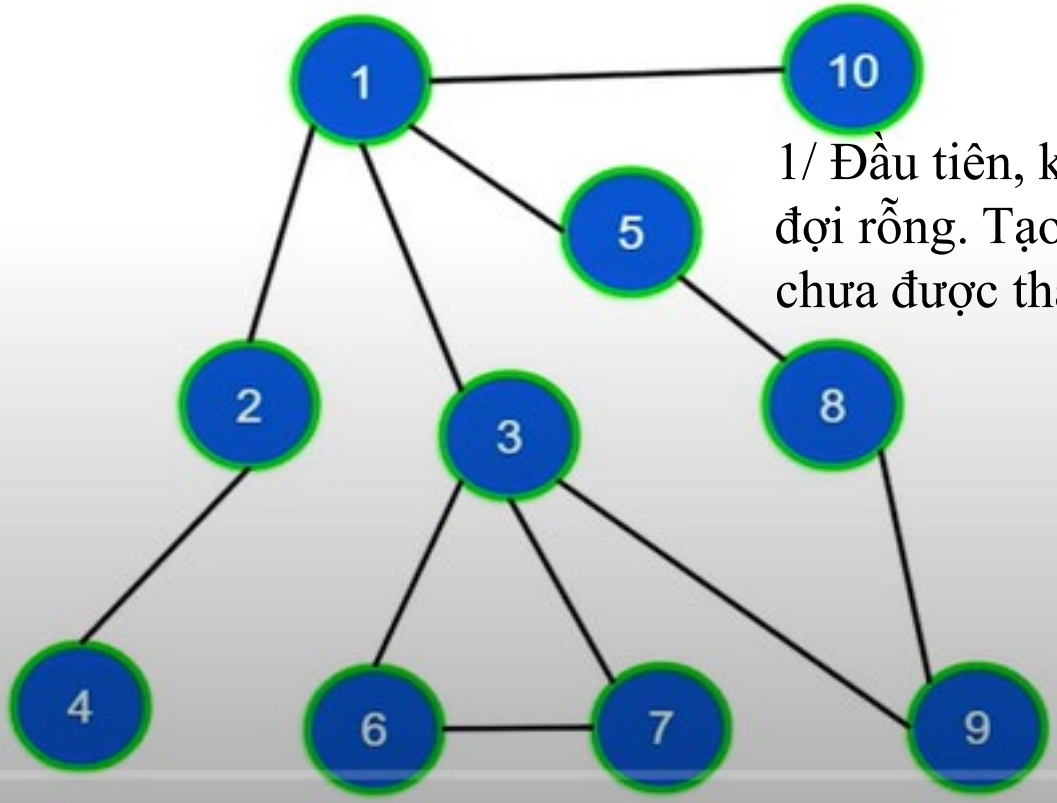
➢ Thuật toán BFS: ví dụ

BFS : Breadth First Search

Kiểm nghiệm thuật toán BFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

BFS(1) =

Back Hàng đợi Front



1/ Đầu tiên, khởi tạo hàng đợi rỗng. Tạo mãng visited chưa được thăm (False)

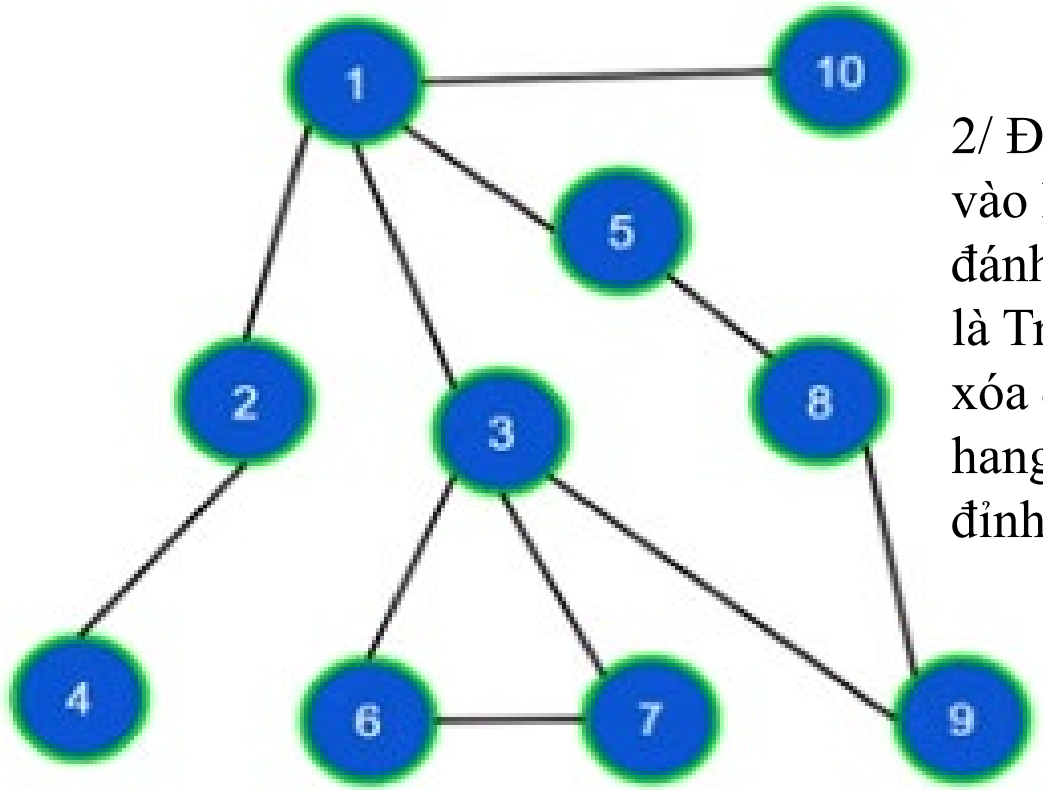
Đỉnh	Visited
1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F
10	F

➢ Thuật toán BFS

BFS : Breadth First Search

Kiểm nghiệm thuật toán BFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

BFS(1) =



2/ Đẩy đỉnh 1 vào hàng đợi -> đánh dấu đỉnh 1 là True (T) -> xóa đỉnh 1 trong hang đợi -> in ra đỉnh 1 **(BFS 1)**

Đỉnh	Visited
1	T
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F
10	F

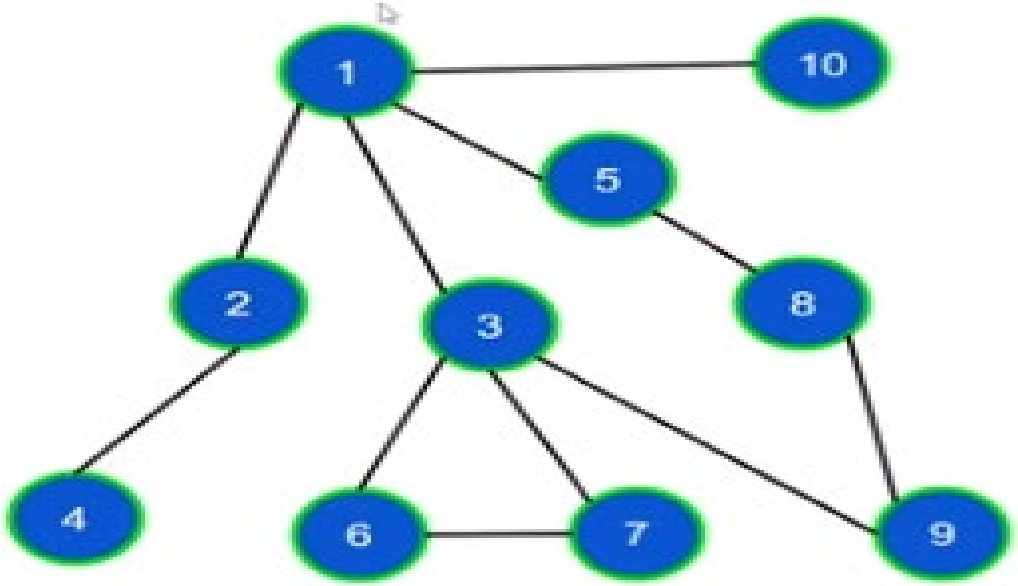
➢ Thuật toán BFS

3/ Duyệt danh sách kề đỉnh 1 gồm 2,3,5,10: đỉnh 2 chưa được thăm -> đẩy 2 vào hàng đợi -> đánh dấu True; tương tự 3,5,10 chưa được thăm -> đẩy 3,5,10 vào hàng đợi -> đánh dấu True **(đẩy cả 2,3,5,10 vào hang đợi**)

BFS : Breadth First Search

Kiểm nghiệm thuật toán BFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

BFS(1) = 1.

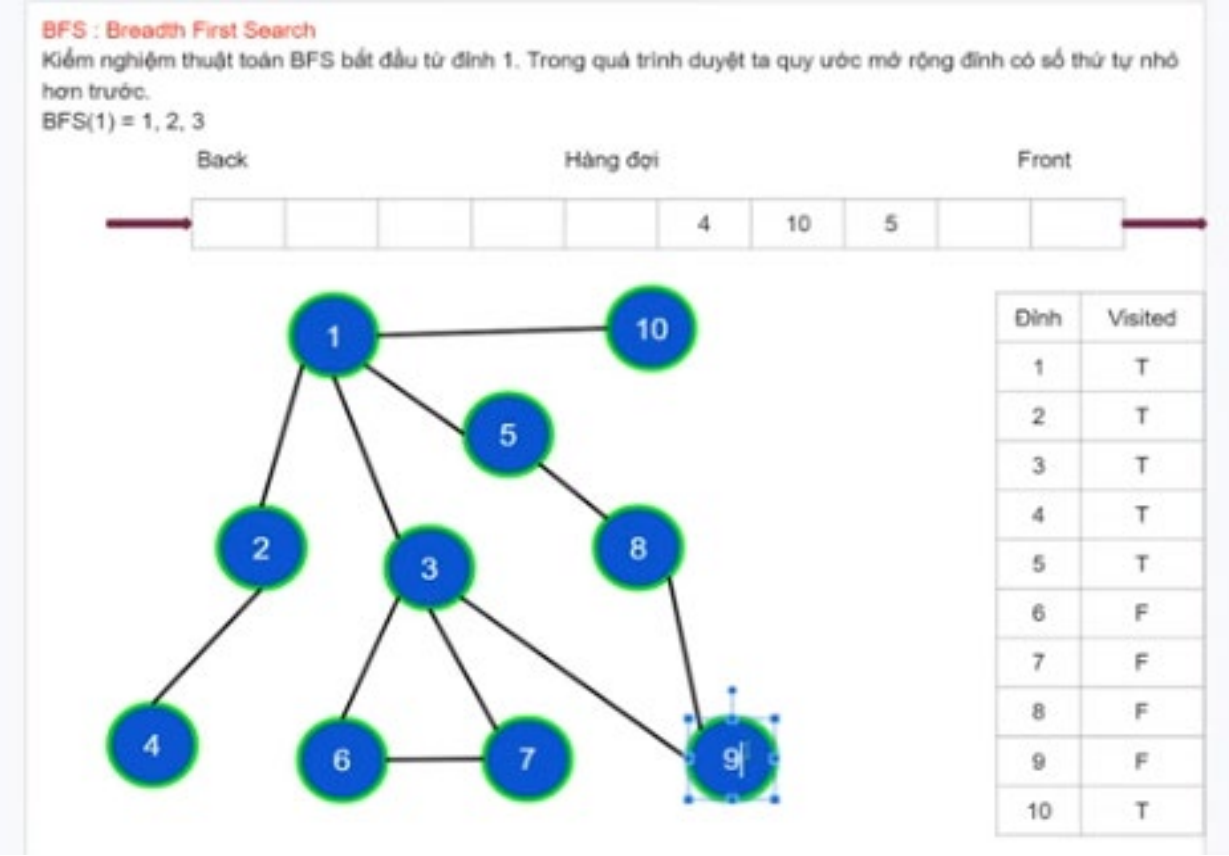


Đỉnh	Visited
1	T
2	T
3	T
4	F
5	T
6	F
7	F
8	F
9	F
10	T

➢ Thuật toán BFS

4/ Xóa đỉnh 2 tại hàng đợi -> in ra đỉnh 2 **BFS(2)** -> xét đỉnh 2, liền kề 2 có 1 (đã thăm True) và 4 (chưa thăm) -> đẩy 4 vào hàng đợi, đánh dấu đã thăm (True).

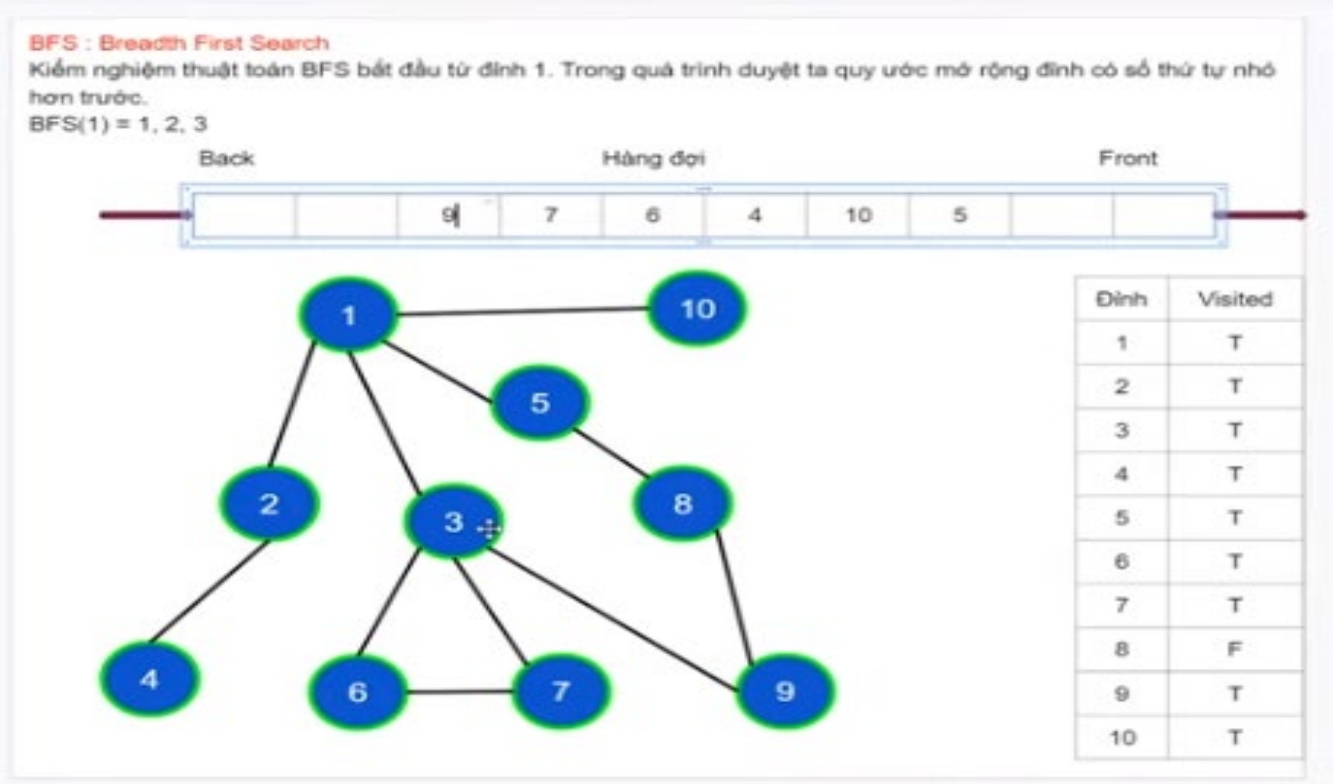
5/ Quay trở lại vòng lặp kiểm tra hàng đợi có rỗng không -> không rỗng -> xóa 3 khỏi hang đợi in ra đỉnh 3 **(BFS (3).**



➢ Thuật toán BFS

6/ Xét đỉnh 3, kề 3 có 1 (đã thăm True), còn 6,7,9 chưa thăm -> đẩy vào hàng đợi -> đánh dấu đã thăm True.

7/ Tiếp tục kiểm tra hàng đợi vẫn chưa rỗng -> xóa 5 khỏi hàng đợi -> In ra 5 **(BFS 5).** Xét đỉnh 5, kề 5 thì có 1 đã thăm (true), chỉ còn 8 -> đẩy 8 vào hàng đợi -> thăm 8, visited (true).



➢ Thuật toán BFS

7/ Tiếp tục kiểm tra hàng đợi vẫn chưa rỗng -> xóa 5 khỏi hàng đợi -> In ra 5 (**BFS 5**). Xét đỉnh 5, kề 5 thì có 1 đã thăm (true), chỉ còn 8 -> đẩy 8 vào hàng đợi -> thăm 8, visited (true).

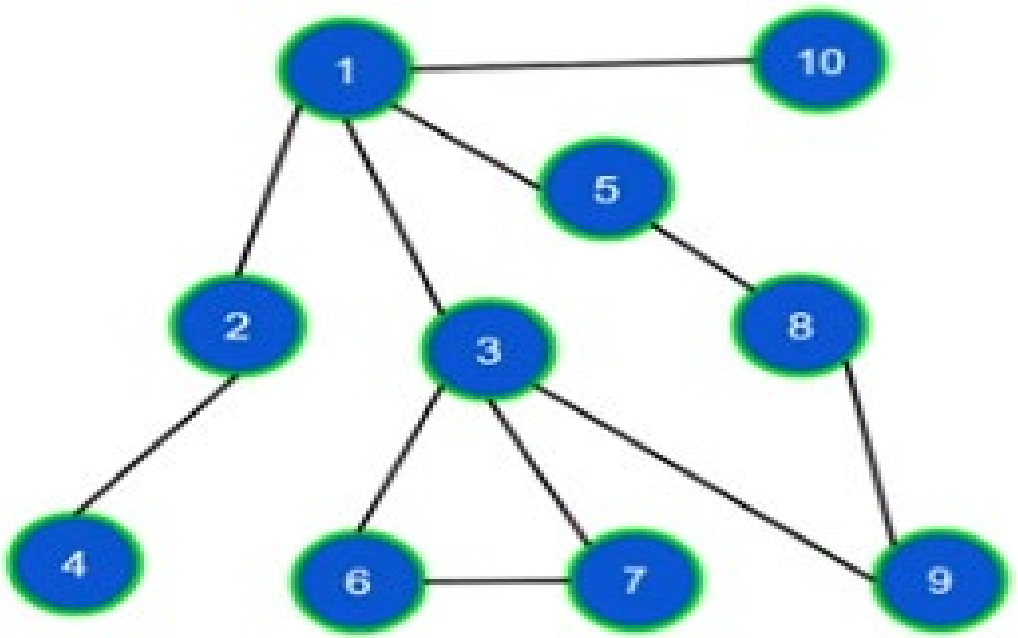
BFS : Breadth First Search

Kiểm nghiệm thuật toán BFS bắt đầu từ đỉnh 1. Trong quá trình duyệt ta quy ước mở rộng đỉnh có số thứ tự nhỏ hơn trước.

BFS(1) = 1, 2, 3, 5

Back | Hàng đợi | Front

	8	9	7	6	4	10			



Đỉnh	Visited
1	T
2	T
3	T
4	T
5	T
6	T
7	T
8	T
9	T
10	T

➢ Thuật toán BFS

8/ Tiếp tục kiểm tra hàng đợi chưa rỗng -> xóa 10 -> in ra 10 **BFS (10)** -> xét kề 10 có 1 đã thăm. Tiếp tục xóa 4 -> **BFS 4**; xét đỉnh 4, các đỉnh kề 4 chỉ có 2 (đã thăm). Xóa 6 -> (**BFS 6**); xét đỉnh 6, kề 6 có 3, 7 đã thăm. Xóa 7 -> **BFS 7**, xét kề 7 có 6,3 đã thăm. Xóa 9 -> **BFS 9**, xét kề 9 có 3, 8 đã thăm. Xóa 8 -> **BFS 8**, xét kề 8 có 3,9 đã thăm -> Hàng đợi rỗng -> kết thúc.

Kết quả BFS: 1,2,3,5,10,4,6,7,8,9

➢ Độ phức tạp thuật toán BFS

- Độ phức tạp thuật toán BFS(u) phụ thuộc vào phương pháp biểu diễn đồ thị
  - Độ phức tạp thuật toán là **$O(n^2)$** trong trường hợp đồ thị biểu diễn dưới dạng ma trận kề, với **n** là số đỉnh của đồ thị.
  - Độ phức tạp thuật toán là **O(n.m)** trong trường hợp đồ thị biểu diễn dưới dạng danh sách cạnh, với **n** là số đỉnh của đồ thị, **m** là số cạnh của đồ thị.
  - Độ phức tạp thuật toán là **O(max(n, m))** trong trường hợp đồ thị biểu diễn dưới dạng danh sách kề, với **n** là số đỉnh của đồ thị, **m** là số cạnh của đồ thị.

➢ Kiểm nghiệm thuật toán BFS (1/2)

- **Ví dụ 3.** Cho đồ thị gồm 13 đỉnh được biểu diễn dưới dạng ma trận kề như hình bên.
- Hãy cho biết kết quả thực hiện thuật toán BFS bắt đầu tại đỉnh **u=1**? Chỉ rõ trạng thái của hàng đợi và tập đỉnh được duyệt theo mỗi bước thực hiện của thuật toán?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

➢ Kiểm nghiệm thuật toán BFS (2/2)

STT	Trạng thái hàng đợi	Danh sách đỉnh được duyệt
1	1	Ø
2	2, 3, 4	1
3	3, 4, 6	1, 2
4	4, 6, 5	1, 2, 3
5	6, 5, 7	1, 2, 3, 4
6	5, 7, 12	1, 2, 3, 4, 6
7	7, 12, 8	1, 2, 3, 4, 6, 5
8	12, 8	1, 2, 3, 4, 6, 5, 7
9	8, 10	1, 2, 3, 4, 6, 5, 7, 12
10	10	1, 2, 3, 4, 6, 5, 7, 12, 8
11	9, 11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10
12	11, 13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9
13	13	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11
14	Ø	1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

Kết quả duyệt: 1, 2, 3, 4, 6, 5, 7, 12, 8, 10, 9, 11, 13

➢ Lưu ý: Nhắc lại

- Với đồ thị vô hướng
  - Nếu DFS(u) = V hoặc BFS(u) = V, ta có thể kết luận đồ thị liên thông
- Với đồ thị có hướng
  - Nếu DFS(u) = V hoặc BFS(u) = V, ta có thể kết luận đồ thị liên thông yếu

Ứng dụng của thuật toán DFS và BFS

➢ Các ứng dụng cơ bản của DFS và BFS

- **Duyệt tất cả các thành phần liên thông của đồ thị.**
- **Tìm đường đi từ đỉnh *s* đến đỉnh *t* trên đồ thị.**
- Duyệt các **đỉnh trụ** trên đồ thị vô hướng.
- Duyệt các **cạnh cầu** trên đồ thị vô hướng.
- **Định chiều** đồ thị vô hướng.
- Xác định tính **liên thông mạnh** trên đồ thị có hướng.
- Xác định tính **liên thông yếu** trên đồ thị có hướng.
- Liệt kê các **thành phần liên thông mạnh** của đồ thị có hướng.

➢ Lưu ý: Ứng dụng DFS và BFS vào xét tính liên thông đồ thị.

- Với đồ thị vô hướng
  - Nếu DFS(u) = V hoặc BFS(u) = V, ta có thể kết luận đồ thị liên thông
- Với đồ thị có hướng
  - Nếu DFS(u) = V hoặc BFS(u) = V, ta có thể kết luận đồ thị liên thông yếu

➢ Xác định thành phần liên thông của đồ thị

- Phát biểu bài toán:
  - Cho đồ thị vô hướng G=<V,E>, trong đó V là tập đỉnh, E là tập cạnh. Xác định các thành phần liên thông của G =<V,E>?
- Thuật toán:

Thuật toán Duyet-TPLT:

Bước 1 (Khởi tạo):

solt = 0; //Khởi tạo số thành phần liên thông ban đầu là 0

Bước 2 (Lặp):

```
for ( u =1; u≤ n; u++)  do //lặp trên tập đỉnh
    if (chuaxet[u] ) then
        solt = solt + 1; //Ghi nhận số thành phần liên thông
        <Ghi nhận các đỉnh thuộc TPLT>;
        BFS (u); //DFS(u);
    endif;
endfor;
```

Bước 3 (Trả lại kết quả):

Return(solt);

end.

➢ Kiểm nghiệm thuật toán

- Cho đồ thị được biểu diễn dưới dạng ma trận kề:

$$\begin{vmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{vmatrix}$$

- Kết quả:
  - Thành phần liên thông 1: BFS(1) = { 1, 3, 5, 7, 9, 11, 13}.
  - Thành phần liên thông 2: BFS(2) = {2, 4, 6, 8, 10, 12}.

➢ Tìm đường đi giữa các đỉnh trên đồ thị

- Phát biểu bài toán
  - Cho đồ thị G =<V, E> (vô hướng hoặc có hướng), trong đó V là tập đỉnh, E là tập cạnh. Tìm đường đi từ đỉnh $s \in V$ đến đỉnh $t \in V$?
- Mô tả thuật toán
  - Nếu $t \in$ DFS(s) hoặc $t \in$ BFS(s) thì ta có thể kết luận **có đường đi** từ ***s*** đến ***t*** trên đồ thị, ngược lại sẽ không có đường đi.
  - Để ghi nhận đường đi ta sử dụng mảng **truoc[]** gồm ***n*** phần tử (n= |V|)
    - Khởi tạo ban đầu **truoc[u]=0** với mọi ***u***.
    - Mỗi khi đưa ***v*∈Ke(u)** vào *ngăn xếp* (nếu sử dụng **DFS**) hoặc *hàng đợi* (nếu sử dụng **BFS**) ta ghi nhận **truoc[v]=*u***.
    - Nếu DFS hoặc BFS không duyệt được đến đỉnh ***t***, khi đó **truoc[t]=0** thì ta kết luận **không** có đường đi từ ***s*** đến ***t***.

➢ Tìm đường đi giữa các đỉnh trên đồ thị: DFS

Thuật toán DFS(s):
Begin
Bước 1 (Khởi tạo):
stack = ∅; *//Khởi tạo stack là ∅*
Push(stack, s); *//Đưa đỉnh s vào ngăn xếp*
chuaxet[s] = False; *//Xác nhận đỉnh s đã duyệt*
Bước 2 (Lặp) :
while (stack ≠ ∅) do
u = Pop(stack); *//Loại đỉnh ở đầu ngăn xếp*
for each v∈ Ke(u) do *//Lấy mỗi đỉnh v∈Ke(u)*
if (chuaxet[v]) then *//Nếu v đúng là chưa duyệt*
chuaxet[v] = False; *// Xác nhận đỉnh v đã duyệt*
Push(stack, u);*//Đưa u vào stack*
Push(stack, v); *//Đưa v vào stack*
truoc[v] = u; *//Ghi nhận truoc[v] là u*
break; *//Chỉ lấy một đỉnh t*
EndIf;
EndFor;
EndWhile;
Bước 3 (Trả lại kết quả):
Return(<Tập đỉnh đã duyệt>);
End.

➢ Tìm đường đi giữa các đỉnh trên đồ thị: BFS

Thuật toán BFS(s):

Bước 1(Khởi tạo):

Queue = ∅; Push(Queue,s); chuaxet[s] = False;

Bước 2 (Lặp):

```
while (Queue ≠ ∅ ) do
        u = Pop(Queue);
        for each v∈Ke(u) do
                if ( chuaxet[v] ) then
                        Push(Queue, v);chuaxet[v]=False;truoc[v]=u;
                EndIf ;
        EndFor ;
EndWhile ;
```

Bước 3 (Trả lại kết quả) :

Return(<Tập đỉnh được duyệt>) ;

End.

➢ Kiểm nghiệm thuật toán

- Xác định đường đi từ đỉnh **1** đến đỉnh **13** trên đồ thị được biểu diễn dưới dạng ma trận kề như hình bên:

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
0	0	1	0	0	1	1	1	0	0	0	1	0
0	1	0	0	1	0	1	0	0	0	0	1	0
0	0	0	1	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	1	1	0	0	1
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	1	1	0	0

➢ Kiểm nghiệm thuật toán DFS(1)

STT	Trạng thái stack	Truoc[s]=?
1	1	0
2	1, 2	truoc[2] =1
3	1, 2, 3	truoc[3] = 2
4	1, 2, 3, 4	truoc[4] =3
5	1, 2, 3, 4, 7	truoc[7] =4
6	1, 2, 3, 4, 7, 5	truoc[5] =7
7	1, 2, 3, 4, 7, 5, 6	truoc[6] =5
8	1, 2, 3, 4, 7, 5, 6, 12	truoc[12] =6
9	1, 2, 3, 4, 7, 5, 6, 12, 8	truoc[8] =12
10	**1, 2, 3, 4, 7, 5, 6, 12, 10**	truoc[10] =12
11	1, 2, 3, 4, 7, 5, 6, 12, 10, 9	truoc[9] =10
12	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11	truoc[11] =9
13	1, 2, 3, 4, 7, 5, 6, 12, 10, 9, 11, 13	truoc[13] =11
14	∅	

Kết quả đường đi từ đỉnh 1 đến đỉnh 13: *13->11-9-10-12-6-5-7-4-3-2-1.*

➢ Kiểm nghiệm thuật toán BFS(1)

STT	Trạng thái Queue	Truoc[s]=?
1	1	truoc[1]=0
2	2, 3, 4	truoc[2]=1; truoc[3]=1; truoc[4]=1;
3	3, 4, 6	truoc[6]= 2
4	4, 6, 5	truoc[5]=3
5	6, 5, 7	truoc[7]= 4
6	5, 7, 12	truoc[12]=6
7	7, 12, 8	truoc[8]= 5
8	12, 8	
9	8, 10	truoc[10]=12;
10	10	
11	9, 11, 13	truoc[9]=10; truoc[11]=10; truoc[13]=10;
12	11, 13	
13	13	
14	∅	

Kết quả đường đi: 13-10-12-6-2-1.

- **Lưu ý:**
  - Đường đi từ đỉnh ***s*** đến đỉnh ***t*** theo thuật toán BFS đi qua ít nhất các cạnh của đồ thị (có **độ dài nhỏ nhất**).

➢ Tính liên thông mạnh trên đồ thị có hướng

- Phát biểu bài toán:
  - Đồ thị có hướng G=<V,E> liên thông mạnh nếu giữa hai đỉnh bất kỳ của nó đều tồn tại đường đi.
  - Cho trước đồ thị có hướng G = <V,E>. Kiểm tra xem G có liên thông mạnh hay không?
- Thuật toán:

```
Boolean      Strong-Connective ( G =<V, E>) {
        ReInit(); //∀u∈V: chuaxet[u] = True;
        for each u∈V do {//Lấy mỗi đỉnh thuộc V
                if (DFS(u)≠V ) then //Nếu DFS(u) ≠V hoặc BFS(u) ≠V
                        return(False); //Đồ thị không liên thông mạnh
                endif;
                ReInit();//Khởi tạo lại các mảng chuaxet[]
        endfor;
        return(True);//Đồ thị liên thông mạnh
}
```

➢ Kiểm nghiệm thuật toán

- Xác định đồ thị có hướng G =<V,E> được biểu diễn dưới dạng ma trận kề bên có **liên thông mạnh** hay không?

0	0	0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	0	1	0
0	0	0	0	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	0
0	0	0	0	1	0	1	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	0	0

➢ Kiểm nghiệm thuật toán kiểm tra tính liên thông mạnh

Đỉnh u∈V	DFS(u)=?//BFS(u)=?	DFS(u) =V?
1∈V	DFS(1) = 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 4, 12, 13	Yes
2∈V	DFS(2) = 2, 3, 9, 5, 7, 11, 8, 4, 1, 6, 10, 12, 13	Yes
3∈V	DFS(3) = 3, 9, 5, 7, 11, 2, 8, 4, 1, 6, 10, 12, 13	Yes
4∈V	DFS(4) = 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 12, 13	Yes
5∈V	DFS(5) = 5, 7, 11, 2, 3, 9, 13, 8, 4, 1, 6, 10, 12	Yes
6∈V	DFS(6) = 6, 10, 2, 3, 9, 5, 7, 11, 8, 4, 1, 12, 13	Yes
7∈V	DFS(7) = 7, 11, 2, 3, 9, 5, 13, 8, 4, 1, 6, 10, 12	Yes
8∈V	DFS(8) = 8, 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 13, 12	Yes
9∈V	DFS(8) = 9, 5, 7, 11, 2, 3, 13, 8, 4, 1, 6, 10, 12	Yes
10∈V	DFS(10) = 10, 2, 3, 9, 5, 7, 11, 8, 4, 1, 6, 12, 13	Yes
11∈V	DFS(11) = 11, 2, 3, 9, 5, 7, 13, 8, 4, 1, 6, 10, 12	Yes
12∈V	DFS(12) = 12, 4, 1, 6, 10, 2, 3, 9, 5, 7, 11, 8, 13	Yes
13∈V	DFS(13) = 13, 9, 5, 7, 11, 2, 3, 8, 4, 1, 6, 10, 12	Yes

➢ Cài đặt thuật toán: Các hàm

- Thủ tục **Read-Data():** Đọc ma trận kề biểu diễn đồ thị trong file **dothi.in**.
- Thủ tục **ReInit():** Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục **DFS(u):** Thuật toán DFS bắt đầu tại đỉnh ***u***.
- Thủ tục **BFS(u):** Thuật toán BFS bắt đầu tại đỉnh ***u***.
- Hàm **Strong-Connective():** Kiểm tra tính liên thông mạnh của đồ thị.

➢ Duyệt các **đỉnh trụ**

- Phát biểu bài toán
  - Cho đồ thị vô hướng G =<V, E>. Đỉnh ***u*** ∈ V được gọi **trụ** nếu loại bỏ đỉnh ***u*** cùng với các cạnh nối với ***u*** làm **tăng** thành phần liên thông của đồ thị.
  - Cho đồ thị vô hướng G =<V, E> , tìm các đỉnh trụ của G?
- Mô tả thuật toán
  - Trong DFS() hoặc BFS(), thiết lập giá trị chuaxet[u] = False.
  - Quá trình duyệt sẽ được thực hiện tại một đỉnh bất kỳ **v≠u**.
    - Nếu DFS(v) = V\{u} hoặc BFS(v) = V\{u}:
      - Đồ thị mới nhận được cũng chỉ có **1** thành phần liên thông
      - Kết luận ***u*** *không là trụ*.
    - Nếu DFS(v) ≠ V\{u} hoặc BFS(v) ≠ V\{u}:
      - Khi đó **u** chính là *trụ* vì số thành phần liên thông của đồ thị đã tăng lên.

➢ Thuật toán duyệt các **đỉnh trụ** của đồ thị

```
Duyet_Tru (G =< V, E >){
        ReInit( ); // ∀u ∈ V: chuaxet[u] = true;
        for(u ∈ V){ //lấy mỗi đỉnh u
                chuaxet[u] = false; // cấm BFS hoặc DFS duyệt u
                if(BFS(v) ≠ V\{u}) // có thể kiểm tra DFS(v) ≠ V\{u}
                        <Ghi nhận u là trụ>;
                ReInit(); // khởi tạo lại mảng chuaxet[]
        }
}
```

➢ Kiểm nghiệm thuật toán

- Xác định đồ thị vô hướng G=<V,E> được biểu diễn dưới dạng ma trận kề bên có những **đỉnh trụ** nào?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

➢ Kiểm nghiệm thuật toán duyệt các đỉnh trụ của đồ thị

Đỉnh u∈V	DFS(v)=?//BFS(v)=? v≠u.	DFS(v) ≠V\u?
1∈V	DFS(2) = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
2∈V	DFS(1) = 1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
3∈V	DFS(1) = 1, 2, 4	Yes
4∈V	DFS(1) = 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
5∈V	DFS(1) = 1, 2, 3, 4	Yes
6∈V	DFS(1) = 1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13	No
7∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 9, 8, 10, 11, 12, 13	No
8∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13	No
9∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8	Yes
10∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9	Yes
11∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13	No
12∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13	No
13∈V	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13	No

➢ Cài đặt thuật toán: Các hàm

- Thủ tục **Read-Data():** Đọc ma trận kề biểu diễn đồ thị trong file **dothi.in.**
- Thủ tục **ReInit():** Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục **DFS(v):** Thuật toán DFS bắt đầu tại đỉnh ***v***.
- Thủ tục **BFS(v):** Thuật toán BFS bắt đầu tại đỉnh ***v***.

➢ Duyệt các **cạnh cầu**

- **Phát biểu bài toán**
  - Cho đồ thị vô hướng G =<V,E>. Cạnh ***e***∈E được gọi là **cầu** nếu loại bỏ ***e*** làm **tăng** thành phần liên thông của đồ thị.
  - Cho trước đồ thị vô hướng G = <V,E>, tìm tất cả các cạnh cầu của đồ thị.
- **Mô tả thuật toán**
  - Đối với đồ thị được biểu diễn dưới dạng ma trận kề:
    - Loại bỏ cạnh e=(u,v)∈E ra khỏi đồ thị ta thực hiện bằng cách cho các phần **tử A[u][v]=0** và **A[v][u]=0**.
  - Đối với đồ thị được biểu diễn dưới dạng danh sách kề:
    - Danh sách kề của đỉnh ***u*** ta bớt đi đỉnh ***v***, Ke(u) = Ke(u)\{**v**}
    - Danh sách kề của đỉnh ***v*** ta bớt đi đỉnh ***u***, Ke(v) = Ke(v)\{**u**}.

➢ Thuật toán duyệt các cạnh cầu của đồ thi

```
Thuật toán   Duyet-Cau ( G =<V, E>) {
        ReInit(); //∀u∈V: chuaxet[u] = True;
        for each e∈E do {//Lấy mỗi cạnh thuộc đồ thị
                E = E\{e}; //Loại bỏ cạnh e ra khỏi đồ thị
                if (DFS(1) ≠ V ) then //Nếu tăng thành phần liên thông của đồ thị
                        <Ghi nhận e là cầu>;
                endif ;
                E = E∪{e} ; //Hoàn trả lại cạnh e
                ReInit();//Khởi tạo lại các mảng chuaxet[]
        endfor;
}
```

❖ Kiểm nghiệm thuật toán

- Xác định đồ thị vô hướng G=<V,E> được biểu diễn dưới dạng ma trận kề bên có những **cạnh cầu** nào?

0	1	1	1	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

❖ Kiểm nghiệm thuật toán duyệt các cạnh cầu của đồ thị

Cạnh e∈E	DFS(u)=?//BFS(u)=?	DFS(u) ≠V?
(1,2)∈E	DFS(1) = 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(1,3)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(1,4)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(2,3)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(2,4)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(3,4)∈E	DFS(1) = 1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 4	No
(3,5)∈E	DFS(1) = 1, 2, 3, 4	Yes
(5,6)∈E	DFS(1) = 1, 2, 3, 4, 5, 7, 6, 8, 9, 10, 11, 12, 13	No
(5,7)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(5,8)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(5,9)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(6,7)∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 9, 8, 7, 10, 11, 12, 13	No
(6,9) ∈E	DFS(1) = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(7,8) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 12, 13	No
(8,9) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(9,10) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9	Yes
(10,11) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11, 13	No
(10,12) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(10,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(11,12) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 12	No
(11,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No
(12,13) ∈E	DFS(1) =1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13	No

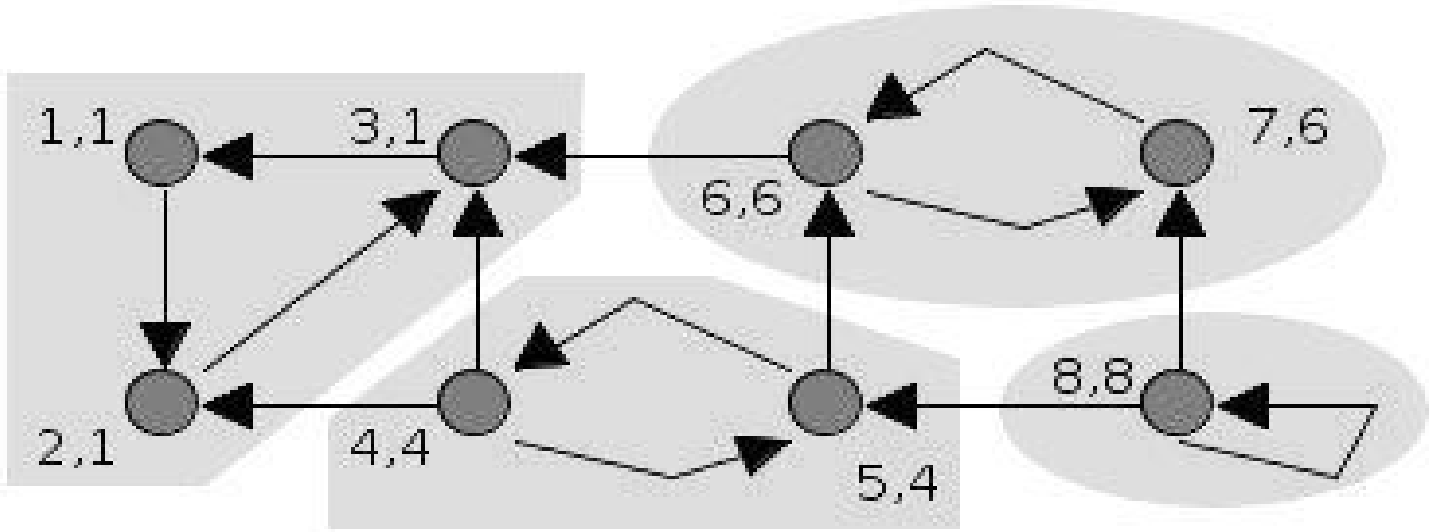
Kết luận: cạnh (3,5), (9,10) là cầu

❖ Cài đặt thuật toán

- Thủ tục **Read-Data():** Đọc ma trận kề biểu diễn đồ thị trong file **dothi.in**.
- Thủ tục **ReInit():** Khởi tạo lại giá trị cho mảng chuaxet[].
- Thủ tục **DFS(u):** Thuật toán DFS bắt đầu tại đỉnh ***u***.
- Thủ tục **BFS(u):** Thuật toán BFS bắt đầu tại đỉnh ***u***.

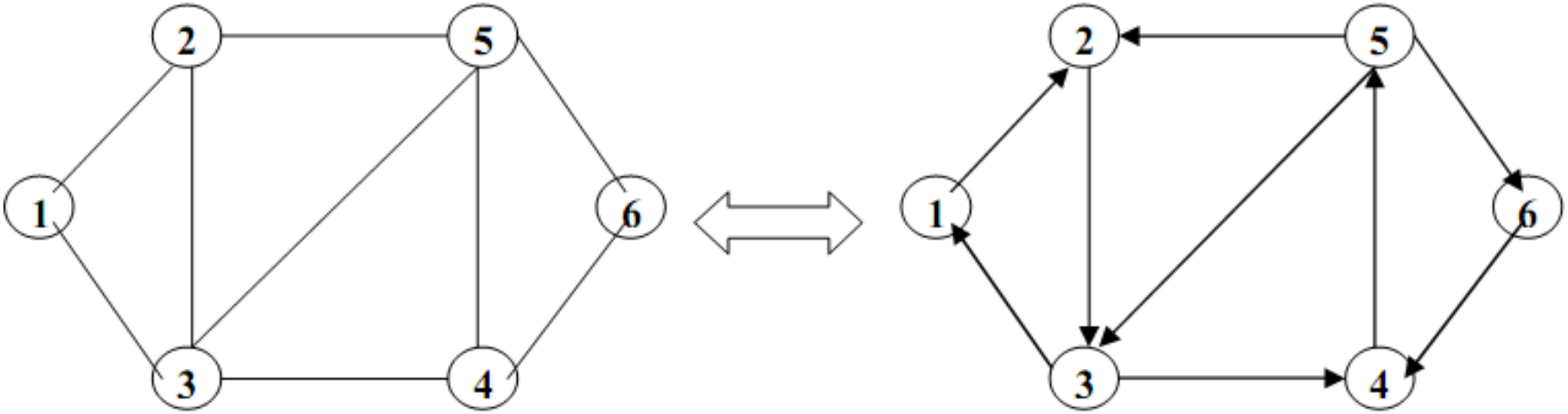
➢ Duyệt các thành phần liên thông mạnh của đồ thị

- Mỗi thành phần **liên thông mạnh** của đồ thị là một đồ thị con của G mà *giữa hai đỉnh bất kỳ của đồ thị con đều có đường đi.*
- **Bài toán:**
  - Liệt kê tất cả các thành phần liên thông mạnh của đồ thị có hướng G=<V,E>.



➢ Bài toán định chiều đồ thị (1/2)

- Định nghĩa
  - Phép định chiều đồ thị vô hướng liên thông là phép biến đổi đồ thị vô hướng liên thông thành đồ thị có hướng liên thông mạnh.
  - Đồ thị vô hướng G =<V,E> có thể dịch chuyển được thành **đồ thị có hướng liên thông mạnh** bằng cách **định chiều mỗi cạnh vô hướng thành một cung có hướng** được gọi là đồ thị định chiều được.



➢ Bài toán định chiều đồ thị (2/2)

- **Định lý**
  - Đồ thị vô hướng liên thông G =<V, E> ***định chiều được*** khi và chỉ khi **tất cả các cạnh e∈E của G đều không phải là cầu**.
- **Bài toán**
  - Cho đồ thị vô hướng liên thông G = <V,E>. Hãy định chiều đồ thị G sao cho ta có thể nhận được đồ thị có hướng **liên thông mạnh**.
- **Một số vấn đề cần quan tâm**
  - *Chứng minh* một đồ thị vô hướng là *định chiều được*.
  - Chỉ ra một *phép định chiều* trên một đồ thị vô hướng.
  - *Viết chương trình* kiểm tra một đồ thị vô hướng có *định chiều được hay không*?

❖ Vấn đề 1: Chứng minh đồ thị định chiều được

- Cách 1: Chứng minh đồ thị vô hướng định chiều được nếu đồ thị có hướng liên thông mạnh.
  - Tức là ta sử dụng thuật toán duyệt BFS hoặc DFS duyệt đồ thị từ mọi đỉnh của đồ thị có hướng.
  - Nếu như tại mọi đỉnh ta có đồ thị liên thông yếu (tức đồ thị liên thông mạnh) thì kết luận: đồ thị định chiều được
- Cách 2: Đồ thị vô hướng định chiều được nếu mọi cạnh của nó không phải là cạnh cầu.
  - Như vậy ta chỉ cần kiểm tra tất cả các cạnh của đồ thị xem có phải là cạnh cầu hay không rồi kết luận.

❖ Vấn đề 2: Hãy định chiều đồ thị

- Định chiều xuôi và ngược cho đồ thị.
  - Chiều xuôi là chiều cạnh đi từ đỉnh u tới các đỉnh khác.
  - Chiều ngược là chiều đi của các đỉnh còn lại thuộc đồ thị tới u.

❖ Ví dụ minh họa

- Cho đồ thị như bên cạnh, chứng minh đồ thị định chiều được:
- Ta có bảng:

0	1	1	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0
0	1	1	0	0

Bước	Cạnh	BFS(i)	Cạnh cầu
1	(1,2)	1.3.4.2.5	No
2	(1,3)	1.2.4.3.5	No
3	(1,4)	1.2.3.4.5	No
4	(2,4)	1.2.3.4.5	No
5	(2,3)	1.2.3.4.5	No
6	(2,5)	1.2.3.4.5	No
7	(3,4)	1.2.3.4.5	No
8	(3,5)	1.2.3.4.5	No

- **Vấn đề 1**: Đồ thị không có cạnh cầu nên nó có thể định chiều được.
- **Vấn đề 2**: Định chiều đồ thị bằng cách chỉ ra cung thuận và cung ngược:
  - Cung thuận: (từ đỉnh 1) có (1>2),(2>3).(3>4),(3>5)
  - Cung ngược: (4.1) (3.1) (4.2) (5.2)

➢ Ứng dụng: BFS và DFS trò chơi dò mìn của Microsoft

- Thuật toán DFS và BFS với một trò chơi Dò mìn (Minesweeper) của Microsoft là một trò chơi nổi tiếng. Phiên bản ban đầu của Dò mìn là một bản đồ hình chữ nhật N x M. Mỗi ô có hoặc không có mìn.
- Ban đầu các ô đều chưa đựơc lật, mỗi lượt ta click chuột để lật. Nếu ta click vào mìn thì thua. Ta thắng khi lật hết tất cả các ô không có mìn.
- Khi lật các ô không có mìn, ta sẽ thấy các số từ 1 đến 8, là số mìn xung quanh ô ta vừa lật.
- Tuy nhiên, khi lật các ô mà xung quanh không có mìn (có 0 quả), ta sẽ cùng một lúc lật tất cả các ô xung quanh cho tới khi ta chạm đến các ô mà xung quanh nó có mìn. Các ô này sẽ đựơc bỏ trống, không đánh số.

➢ Ứng dụng: BFS và DFS trò chơi dò mìn của Microsoft

- Thuật toán DFS và BFS với một trò chơi Dò mìn (Minesweeper) của Microsoft là một trò chơi nổi tiếng. Phiên bản ban đầu của Dò mìn là một bản đồ hình chữ nhật N x M. Mỗi ô có hoặc không có mìn.
- Ban đầu các ô đều chưa được lật, mỗi lượt ta click chuột để lật. Nếu ta click vào mìn thì thua. Ta thắng khi lật hết tất cả các ô không có mìn.
- Khi lật các ô không có mìn, ta sẽ thấy các số từ 1 đến 8, là số mìn xung quanh ô ta vừa lật.
- Tuy nhiên, khi lật các ô mà xung quanh không có mìn (có 0 quả), ta sẽ cùng một lúc lật tất cả các ô xung quanh cho tới khi ta chạm đến các ô mà xung quanh nó có mìn. Các ô này sẽ được bỏ trống, không đánh số.

❖ Ứng dụng: BFS và DFS trò chơi dò mìn của Microsoft

•Nếu coi map của trò chơi là một đồ thị, thì mỗi ô là một đỉnh, với mỗi đỉnh kết nối với các đỉnh ở trên, dưới, trái, phải của nó. Làm thế nào viết thuật toán mở một ô không có mìn, khi được mở sẽ tự động mở tất cả các ô xung quanh? Đây là chỗ dùng đến DFS và BFS.

•Khi mở một ô không có mìn, ta có thể dùng DFS hoặc BFS để loang ra xung quanh cho tới khi chạm phải các ô có mìn xung quanh. Ví dụ:

❖ Ứng dụng: BFS và DFS trò chơi dò mìn của Microsoft : chương trình máy tính

```
# v la dinh hien tai dang xet
function open(graph, v):
        states[v] = 1 # state = 1 la da tham, 0 la
chua tham
        # neu xung quanh v khong co min, ta bat
dau dung DFS de mo? tiep cac o xung quanh
        if (numMinesArround(v) == 0):
                for u in neighbor(v):
                        if states[u] == 0:
                                open(graph, u)
```

Bài tập 1

- Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình bên. Hãy:
  a) Trình bày thuật toán **BFS(u)**?
  b) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh ***u=1***? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.
  c) Kiểm nghiệm thuật toán BFS(u) bắt đầu tại đỉnh ***u=7***? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bài tập 2

- Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình bên. Hãy:
  a) Trình bày thuật toán **DFS(u)**?
  b) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh ***u=1***? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.
  c) Kiểm nghiệm thuật toán DFS(u) bắt đầu tại đỉnh ***u=7***? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bài tập 3

- Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình bên. Hãy:
  a) Trình bày thuật toán duyệt các *thành phần liên thông* của đồ thị?
  b) Kiểm nghiệm thuật toán trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

0	0	1	0	1	0	1	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	0	0
1	0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	0	0	1	0	1	0	0	0
1	0	1	0	1	0	0	0	1	0	0	0	0
0	0	0	1	0	1	0	0	0	1	0	1	0
0	0	0	0	1	0	1	0	0	0	1	0	1
0	0	0	1	0	1	0	1	0	0	0	1	0
0	0	1	0	1	0	0	0	1	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	1	0	0	0	1	0	1	0	0

Bài tập 4

- Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình bên. Hãy:
  a) Dựa vào thuật toán **BFS**, xây dựng thuật toán tìm đường đi từ đỉnh ***s*** đến đỉnh ***t*** trên đồ thị?
  b) Tìm đường đi từ đỉnh ***s=1*** đến đỉnh ***t =13*** trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.
  c) Viết chương trình tìm đường đi từ ***s*** đến ***t*** dựa vào biểu diễn đồ thị dưới dạng ma trận kề.

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

Bài tập 5

- Cho đồ thị vô hướng được biểu diễn dưới dạng ma trận kề như hình bên. Hãy:

0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1	1	0	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	1	0	0	0
0	0	0	0	1	0	1	0	1	0	0	0	0
0	0	0	0	1	1	0	1	0	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	0	1	0	1	1
0	0	0	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	0	0	0	0	1	1	1	0

a) Dựa vào thuật toán **DFS**, xây dựng thuật toán tìm đường đi từ đỉnh ***s*** đến đỉnh ***t*** trên đồ thị?

b) Tìm đường đi từ đỉnh ***s=1*** đến đỉnh ***t =13*** trên đồ thị đã cho? Chỉ rõ kết quả trung gian theo mỗi bước thực hiện của thuật toán.

c) Viết chương trình tìm đường đi từ ***s*** đến ***t*** dựa vào biểu diễn đồ thị dưới dạng ma trận kề.