

LECTURE 07

DEPTH FIRST SEARCH ALGORITHM



Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Thuật toán DFS là gì?

Thuật toán **Depth First Search** (DFS) tìm kiếm theo chiều sâu là thuật toán tìm kiếm trong đồ thị **vô hướng** hoặc **có hướng**, không trọng số.

Thuật toán DFS luôn tìm kiếm được đường đi từ một đỉnh bất kỳ cho trước tới một đỉnh đích (nếu 2 đỉnh thuộc cùng thành phần liên thông với nhau). Nhưng **không chắc chắn** đường đi sẽ là đường đi ngắn nhất.

Độ phức tạp thuật toán DFS

Độ phức tạp: $O(V + E)$

- Tập hợp V (Vertices) những phần tử gọi là đỉnh của đồ thị.
- Tập hợp E (Edges) những phần tử gọi là cạnh của đồ thị.

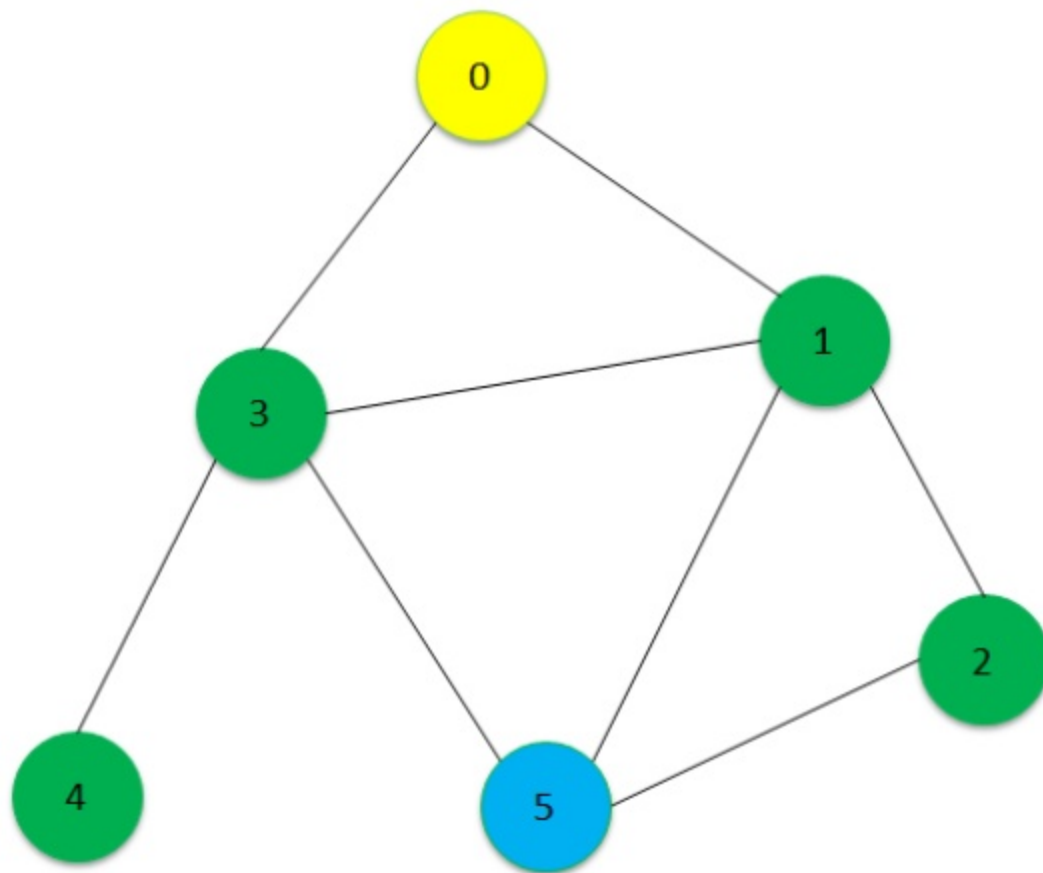
Ý tưởng thuật toán

Mô tả: Xuất phát từ 1 đỉnh bất kỳ, đi tới tất các đỉnh kề của nó, lưu các đỉnh này lại. Sau đó tiếp tục lấy đỉnh **mới nhất vừa được lưu** (từ tập đỉnh đã được lưu) ra xét và đi cho đến khi không còn đỉnh nào có thể đi.

Trong quá trình đi từ đỉnh này sang đỉnh kia, tiến hành lưu lại đỉnh cha của đỉnh kề, để khi đi ngược lại từ đỉnh **Kết Thúc** đến đỉnh **Xuất Phát**, ta có được đường đi cần tìm.

Bài toán minh họa

Cho đồ thị vô hướng như hình vẽ. Tìm **đường đi** từ đỉnh **0** đến đỉnh **5**.



Hướng cài đặt

Có 2 cách để chạy bài toán này có thể dùng **đệ quy** hoặc **không đệ quy**. Nếu chạy không đệ quy thì cần sử dụng cấu trúc dữ liệu ngăn xếp (**stack**) để lưu các đỉnh đang xét. Phương pháp trình bày dưới đây sẽ sử dụng phương pháp khử đệ quy dùng ngăn xếp.

Bước 0: Chuẩn bị dữ liệu

Từ dữ liệu đầu vào là ma trận kề, danh sách kề hoặc định dạng dữ liệu khác.

Adjacency Matrix

6						
0	1	0	1	0	0	
1	0	1	1	0	1	
0	1	0	0	0	1	
1	1	0	0	1	1	
0	0	0	1	0	0	
0	1	1	1	0	0	

Adjacency List

6	8
0	1
0	3
1	2
1	3
1	5
2	5
3	4
3	5

Bước 0: Chuẩn bị dữ liệu

Chuyển danh sách cạnh kề vào graph.

	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

Mảng đánh dấu các đỉnh đã xét.

	0	1	2	3	4	5
visited	false	false	false	false	false	false

Mảng lưu vết đường đi.

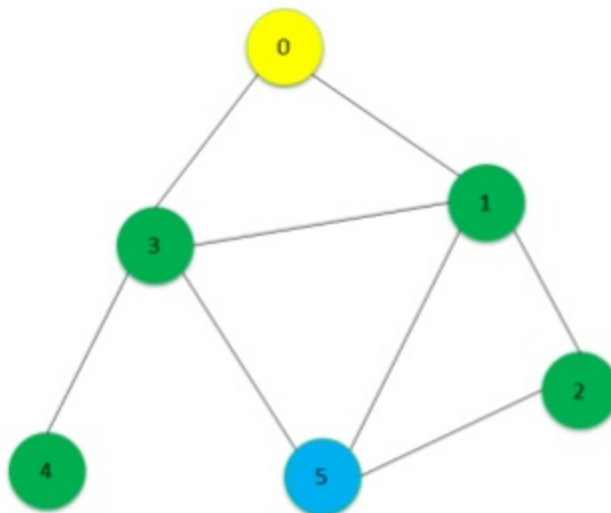
	0	1	2	3	4	5
path	-1	-1	-1	-1	-1	-1

Ngăn xếp lưu các đỉnh đang xét.

stack	...
-------	-----

Bước 0: Chuẩn bị dữ liệu (tiếp theo)

Lấy đỉnh bắt đầu đi (**đỉnh 0**). Đánh dấu đã xét đỉnh 0 và bỏ vào ngăn xếp.



Mảng đánh dấu các đỉnh đã xét.

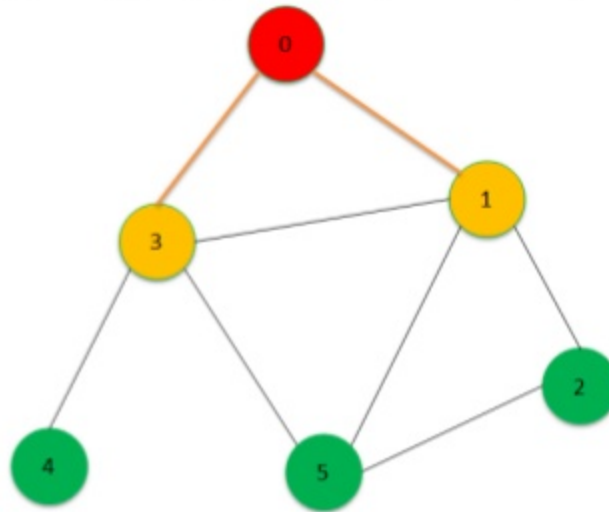
	0	1	2	3	4	5
visited	true	false	false	false	false	false

Hàng đợi lưu các đỉnh đang xét.

stack	0
-------	---

Bước 1: Chạy thuật toán lần 1

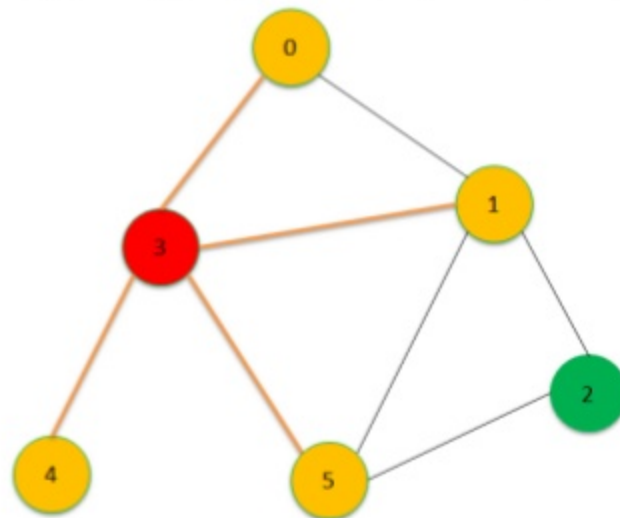
Lấy **đỉnh 0** ra xét và tìm những đỉnh có kết nối với đỉnh 0 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack	1	3				
path	-1	0	-1	0	-1	-1
visited	true	true	false	true	false	false

Bước 2: Chạy thuật toán lần 2

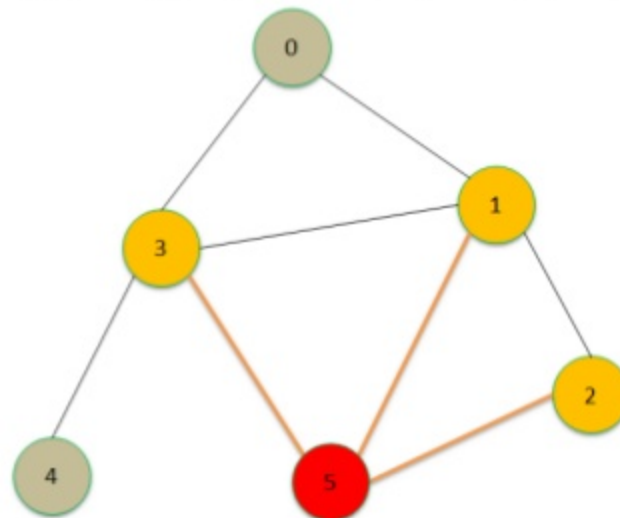
Lấy **đỉnh 3** ra xét và tìm những đỉnh có kết nối với đỉnh 3 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack	1	4	5			
path	-1	0	-1	0	3	3
visited	true	true	false	true	true	true

Bước 3: Chạy thuật toán lần 3

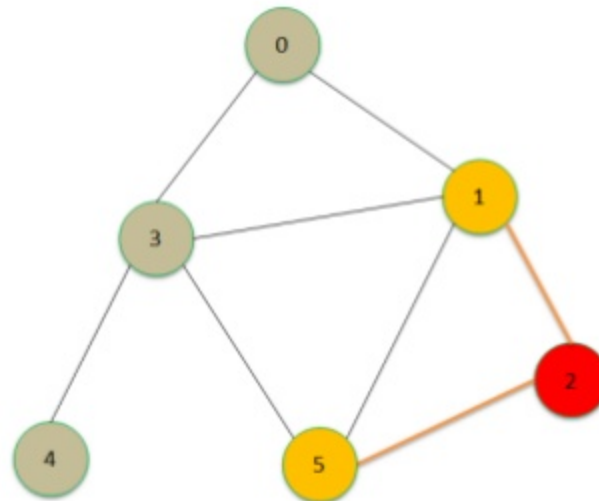
Lấy **đỉnh 5** ra xét và tìm những đỉnh có kết nối với đỉnh 5 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack	1	4	2			
path	-1	0	5	0	3	3
visited	true	true	true	true	true	true

Bước 4: Chạy thuật toán lần 4

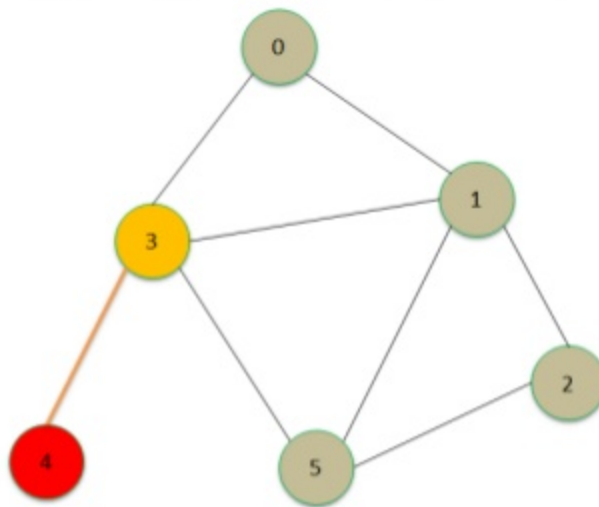
Lấy **đỉnh 2** ra xét và tìm những đỉnh có kết nối với đỉnh 2 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack	1	4				
path	-1	0	5	0	3	3
visited	true	true	true	true	true	true

Bước 5: Chạy thuật toán lần 5

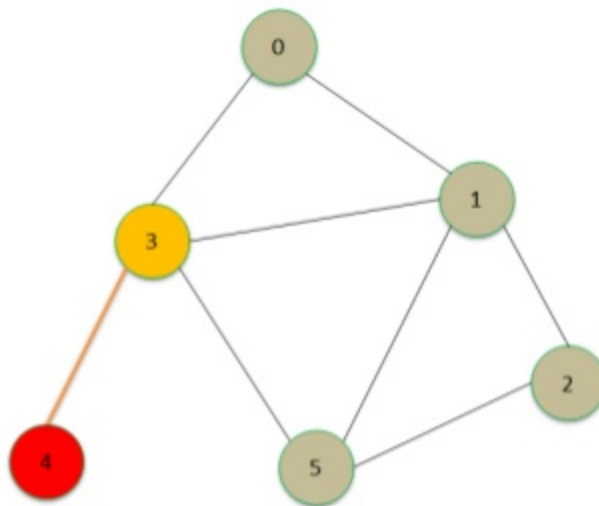
Lấy **đỉnh 4** ra xét và tìm những đỉnh có kết nối với đỉnh 4 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack	0	1				
path	-1	0	5	0	3	3
visited	true	true	true	true	true	true

Bước 6: Chạy thuật toán lần 6

Lấy **đỉnh 1** ra xét và tìm những đỉnh có kết nối với đỉnh 1 (những đỉnh chưa xét) bỏ vào stack.



	0	1	2	3	4	5
graph	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3
stack				
path	-1	0	5	0	3	3
visited	true	true	true	true	true	true

Dừng thuật toán

Tất cả các đỉnh đều được xét → dừng thuật toán.

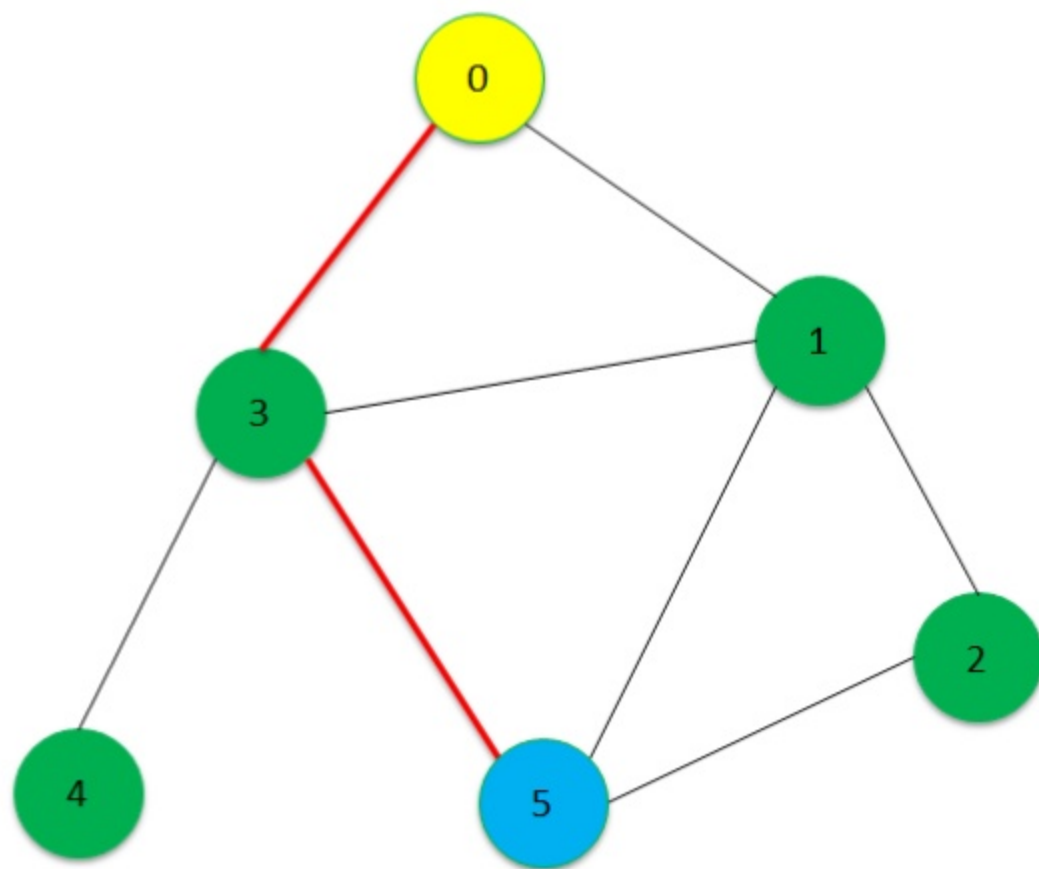


0 → 3 → 5

Thứ tự duyệt DFS là **0, 3, 5, 2, 4, 1**.

Đáp án bài toán (dùng stack)

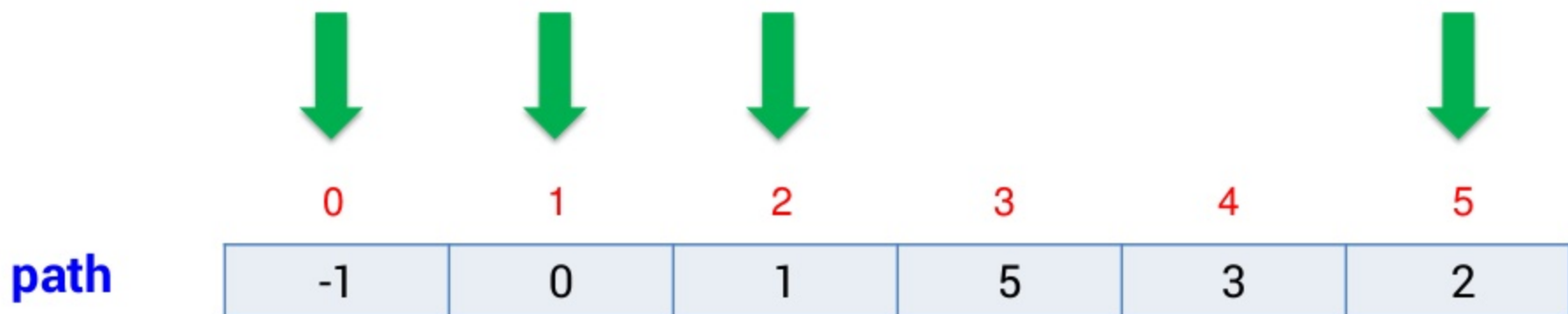
Đường đi từ đỉnh 0 đến đỉnh 5 như hình vẽ.



KẾT QUẢ KHI CHẠY ĐỆ QUY

Dừng thuật toán (đệ quy)

Tất cả các đỉnh đều được xét → dừng thuật toán.

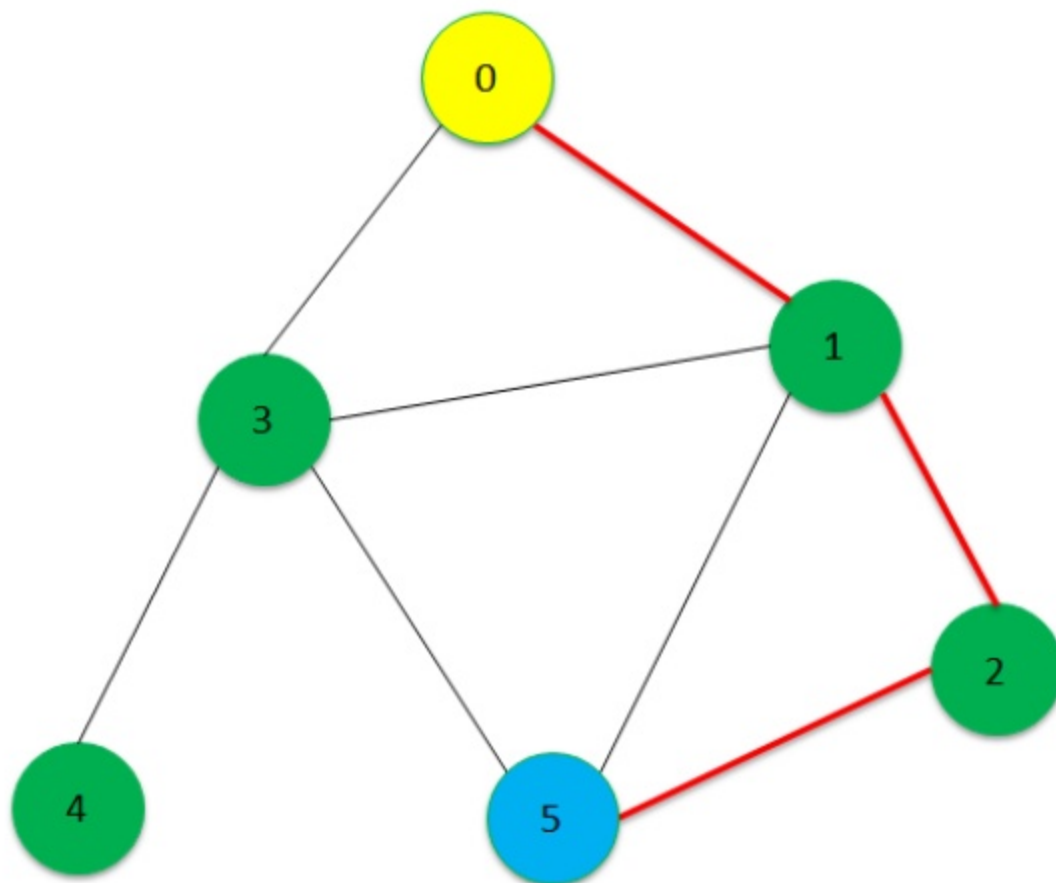


0 → 1 → 2 → 5

Thứ tự duyệt DFS đệ quy là **0, 1, 2, 5, 3, 4.**

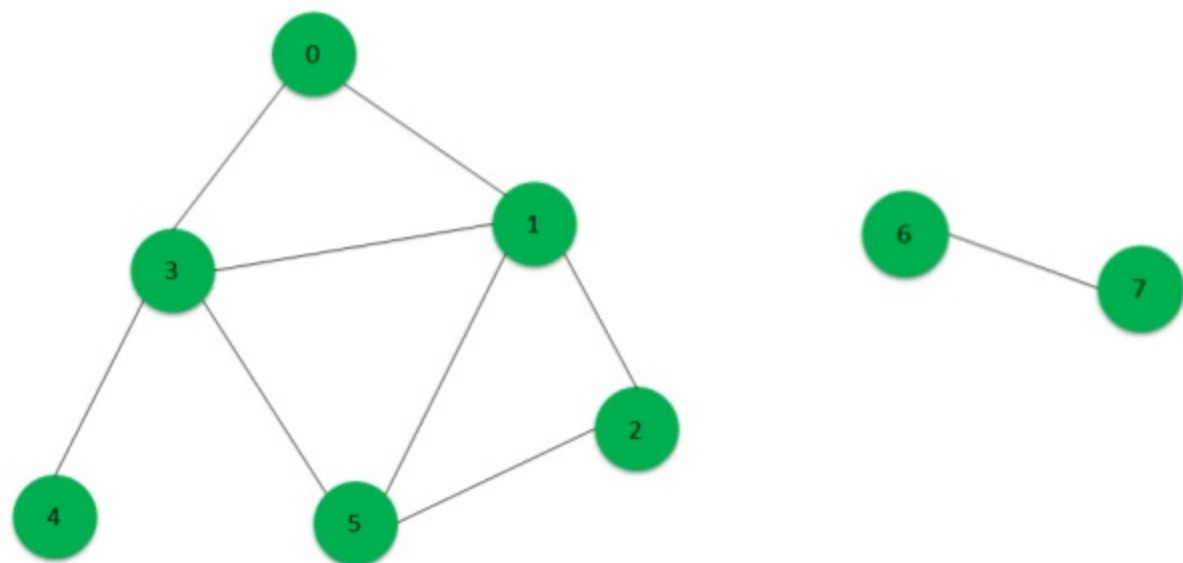
Đáp án bài toán (đệ quy)

Đường đi từ đỉnh 0 đến đỉnh 5 như hình vẽ.



Lưu ý khi sử dụng DFS

Khi 2 đỉnh cần tìm đường đi nhưng lại không có đường đi tới nhau được thì kết quả trả về sẽ như thế nào?

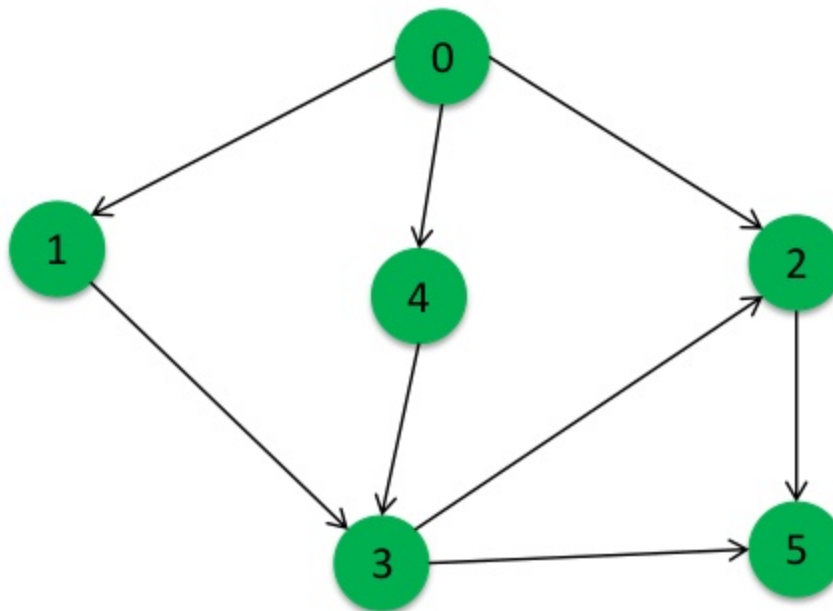


path

0	1	2	3	4	5	6	7
-1	0	5	0	3	3	-1	-1

Bài tập luyện tập

Tìm đường đi từ đỉnh 0 đến đỉnh 5:



stack

0 → 4 → 3 → 5

Đệ quy

0 → 1 → 3 → 2 → 5

MÃ NGUỒN MINH HỌA BẰNG C++



Source Code DFS C++

Khai báo thư viện và các biến toàn cục:

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
#define MAX 100
int V;
bool visited[MAX];
int path[MAX];
vector<int> graph[MAX];
```


Source Code DFS C++

Thuật toán chính DFS (part 1)

```
void DFS(int src) {  
    for (int i = 0; i < V; i++) {  
        visited[i] = false;  
        path[i] = -1;  
    }  
    stack<int> s;  
    visited[src] = true;  
    s.push(src);  
  
    // to be continued
```

Source Code DFS C++

Thuật toán chính DFS (part 2)

```
while (!s.empty()) {  
    int u = s.top();  
    s.pop();  
    for (int i = 0; i < graph[u].size(); i++) {  
        int v = graph[u][i];  
        if (!visited[v]) {  
            visited[v] = true;  
            s.push(v);  
            path[v] = u;  
        }  
    }  
}
```

Source Code DFS C++

In đường đi từ mảng lưu vết (dùng đệ quy):

```
void printPathRecursion(int s, int f) {  
    if (s == f)  
        cout << f << " ";  
    else {  
        if (path[f] == -1)  
            cout << "No path" << endl;  
        else {  
            printPathRecursion(s, path[f]);  
            cout << f << " ";  
        }  
    }  
}
```

Source Code DFS C++

In đường đi từ mảng lưu vết (KHÔNG dùng đệ quy):

```
void printPath(int s, int f) {  
    int b[MAX];  
    int m = 0;  
    if (f == s) {  
        cout << s;  
        return;  
    }  
    if (path[f] == -1) {  
        cout << "No path" << endl;  
        return;  
    }  
    // to be continued
```

Source Code DFS C++

In đường đi từ mảng lưu vết (KHÔNG dùng đệ quy):

```
while (1) {  
    b[m++] = f;  
    f = path[f];  
    if (f == s) {  
        b[m++] = s;  
        break;  
    }  
}  
  
for (int i = m - 1; i >= 0; i--) {  
    cout << b[i] << " ";  
}  
  
}
```

Source Code DFS C++

Hàm main để gọi thực hiện:

```
int main() {  
    freopen("INPUT.INP", "rt", stdin);  
    int E, u, v;  
    cin >> V >> E;  
    for (int i = 0; i < E; i++) {  
        cin >> u >> v;  
        graph[u].push_back(v);  
        graph[v].push_back(u);  
    }  
    int s = 0;  
    int f = 5;  
    DFS(s);  
    printPath(s, f);  
    return 0;  
}
```

Source Code DFS C++ (Đệ quy)

Thuật toán chính DFS đệ quy:

```
void DFSRecursion(int s) {  
    visited[s] = true;  
    for (int i = 0; i < graph[s].size(); i++) {  
        int v = graph[s][i];  
        if (!visited[v]) {  
            path[v] = s;  
            DFSRecursion(v);  
        }  
    }  
}
```

Source Code DFS C++ (Đệ quy)

```
int main() {
    freopen("INPUT.INP", "rt", stdin);
    int E, u, v;
    cin >> V >> E;
    for (int i = 0; i < E; i++) {
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    int s = 0, f = 5;
    for (int i = 0; i < V; i++) {
        visited[i] = false;
        path[i] = -1;
    }
    DFSRecursion(s);
    printPath(s, f);
    return 0;
}
```


MÃ NGUỒN MINH HỌA BẰNG PYTHON



Source Code DFS python

Khai báo thư viện và các biến toàn cục:

```
MAX = 100
V = None
E = None
visited = [False]*MAX
path = [0]*MAX
graph = [[] for i in range(MAX)]
```

Source Code DFS python

Thuật toán chính DFS (part 1)

```
def DFS(src) :  
    for i in range(V) :  
        visited[i] = False  
        path[i] = -1  
  
    s = []  
    visited[src] = True  
    s.append(src)  
  
// to be continued
```

Source Code DFS python

Thuật toán chính DFS (part 2)

```
while len(s) > 0:
    u = s[-1]
    s.pop()
    for v in graph[u]:
        if visited[v] == False:
            visited[v] = True
            s.append(v)
            path[v] = u
```

Source Code DFS python

In đường đi từ mảng lưu vết (dùng đệ quy):

```
def printPathRecursion(s, f):  
    if s == f:  
        print(f, end=' ')  
    else:  
        if path[f] == -1:  
            print("No path")  
        else:  
            printPathRecursion(s, path[f])  
            print(f, end = ' ')
```

Source Code DFS python

In đường đi từ mảng lưu vết (KHÔNG dùng đệ quy):

```
def printPath(s, f):  
    b = []  
    if f == s:  
        print(f)  
        return  
    if path[f] == -1:  
        print("No path")  
        return  
    while True:  
        b.append(f)  
        f = path[f]  
        if f == s:  
            b.append(s)  
            break  
    for i in range(len(b)-1, -1, -1):  
        print(b[i], end = ' ')
```

Source Code DFS python

Hàm main để gọi thực hiện:

```
if __name__ == '__main__':  
    V, E = map(int, input().split())  
    for i in range(E):  
        u, v = map(int, input().split())  
        graph[u].append(v)  
        graph[v].append(u)  
    s = 0  
    d = 5  
    DFS(s)  
    printPath(s, d)
```

Source Code DFS python (Đệ quy)

Thuật toán chính DFS đệ quy:

```
def DFSRecursion(s):  
    visited[s] = True  
    for v in graph[s]:  
        if visited[v] == False:  
            path[v] = s  
            DFSRecursion(v)
```


Source Code DFS python (Đệ quy)

```
if __name__ == '__main__':  
    V, E = map(int, input().split())  
    for i in range(E):  
        u, v = map(int, input().split())  
        graph[u].append(v)  
        graph[v].append(u)  
    s = 0  
    f = 5  
    DFSRecursion(s)  
    printPath(s, f, path)
```

MÃ NGUỒN MINH HỌA BẰNG JAVA



Source Code DFS Java

Khai báo thư viện và các biến toàn cục:

```
import java.util.Scanner;  
import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.Stack;
```

Source Code DFS Java

Thuật toán chính DFS (part 1)

```
public static void DFS(int src) {  
    int V = graph.size();  
    path = new ArrayList<Integer>();  
    visited = new ArrayList<Boolean>();  
    for (int i = 0; i < V; i++) {  
        visited.add(false);  
        path.add(-1);  
    }  
    Stack<Integer> stack = new Stack<Integer>();  
    visited.set(src, true);  
    stack.add(src);  
    // to be continued
```

Source Code DFS Java

Thuật toán chính DFS (part 2)

```
while (!stack.isEmpty()) {
    int u = stack.peek();
    stack.pop();
    for (int i = 0; i < graph.get(u).size(); i++) {
        int v = graph.get(u).get(i);
        if (!visited.get(v)) {
            visited.set(v, true);
            stack.add(v);
            path.set(v, u);
        }
    }
}
```

Source Code DFS Java

In đường đi từ mảng lưu vết (dùng đệ quy):

```
public static void printPathRecursion(int s, int f) {  
    if (s == f) {  
        System.out.print(f + " ");  
    }  
    else {  
        if (path.get(f) == -1) {  
            System.out.print("NO PATH");  
        } else {  
            printPathRecursion(s, path.get(f));  
            System.out.print(f + " ");  
        }  
    }  
}
```

Source Code DFS Java

In đường đi từ mảng lưu vết (KHÔNG dùng đệ quy):

```
public static void printPath(int s, int f) {  
    if (f == s) {  
        System.out.print(s);  
        return;  
    }  
    if (path.get(f) == -1) {  
        System.out.print("NO PATH");  
        return;  
    }  
    ArrayList<Integer> b = new ArrayList<Integer>();  
    // to be continued
```

Source Code DFS Java

In đường đi từ mảng lưu vết (KHÔNG dùng đệ quy):

```
while (true) {  
    b.add(f);  
    f = path.get(f);  
    if (s == f) {  
        b.add(f);  
        break;  
    }  
}  
  
for (int i = b.size() - 1; i >= 0; i--) {  
    System.out.print(b.get(i) + " ");  
}  
  
}
```


Source Code DFS Java

Hàm main để gọi thực hiện:

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int V = sc.nextInt(), E = sc.nextInt();  
    graph = new ArrayList<ArrayList<Integer>>();  
    for (int i = 0; i < V; i++)  
        graph.add(new ArrayList<Integer>());  
    for (int i = 0; i < E; i++) {  
        int u = sc.nextInt(), v = sc.nextInt();  
        graph.get(u).add(v);  
        graph.get(v).add(u);  
    }  
    int s = 0, f = 5;  
    DFS(s);  
    printPath(s, f);  
}
```

Source Code DFS Java (Đệ quy)

Thuật toán chính DFS đệ quy:

```
public static void DFSRecursion(int s) {  
    visited.set(s, true);  
    for (int i = 0; i < graph.get(s).size(); i++) {  
        int v = graph.get(s).get(i);  
        if (!visited.get(v)) {  
            path.set(v, s);  
            DFSRecursion(v);  
        }  
    }  
}
```

Source Code DFS Java (Đệ quy)

Source code hàm main DFS đệ quy - part 1

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int V = sc.nextInt(), E = sc.nextInt();
    graph = new ArrayList<ArrayList<Integer>>();
    for (int i = 0; i < V; i++)
        graph.add(new ArrayList<Integer>());
    for (int i = 0; i < E; i++) {
        int u = sc.nextInt(), v = sc.nextInt();
        graph.get(u).add(v);
        graph.get(v).add(u);
    }
    // to be continued
}
```

Source Code DFS Java (Đệ quy)

Source code hàm main DFS đệ quy - part 2

```
int s = 0, f = 5;
path = new ArrayList<Integer>();
visited = new ArrayList<Boolean>();
for (int i = 0; i < V; i++) {
    visited.add(false);
    path.add(-1);
}
DFSRecursion(s);
printPath(s, f);
}
```

Hỏi đáp

