Big-O Blue

LECTURE 06

DEPTH-FIRST SEARCH ALGORITHM

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Depth-first Search

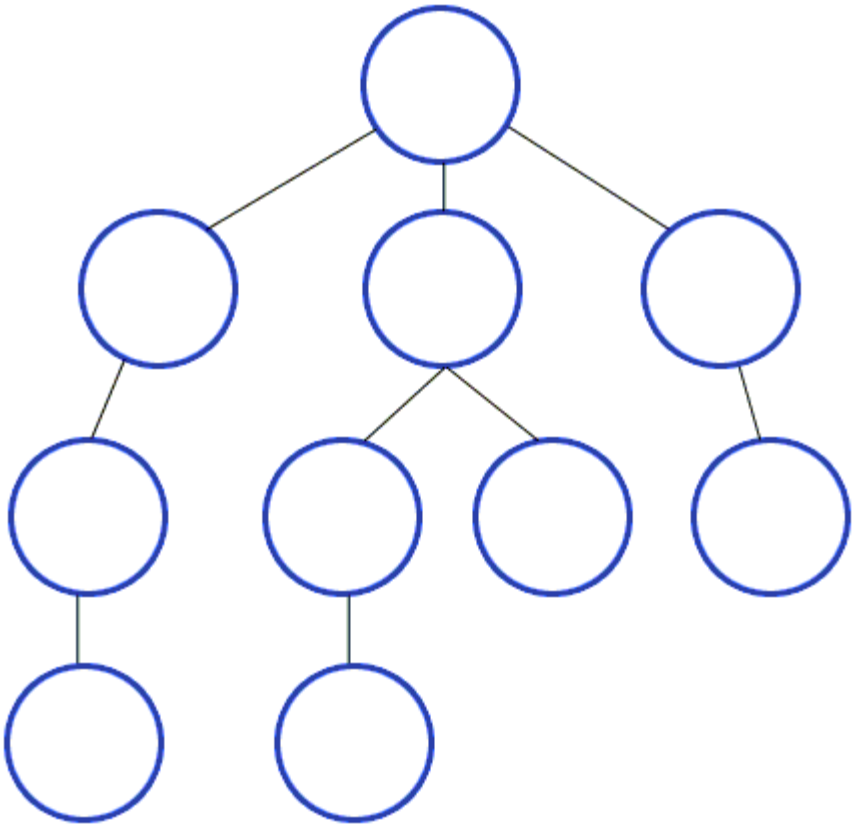
Depth-first Search (Thuật toán tìm kiếm theo chiều sâu): là thuật toán tìm kiếm đường đi trên đồ thị **vô hướng** hoặc **có hướng**, **không** trọng số.

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

Độ phức tạp: O(V + E)

- V (Vertices): số lượng đỉnh của đồ thị.
- E (Edges): số lượng cạnh của đồ thị.

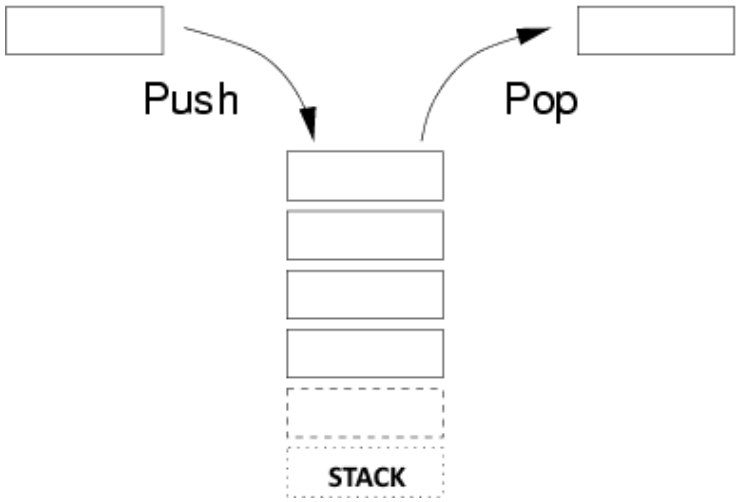
Mô phỏng cách chạy thuật toán



Hướng cài đặt

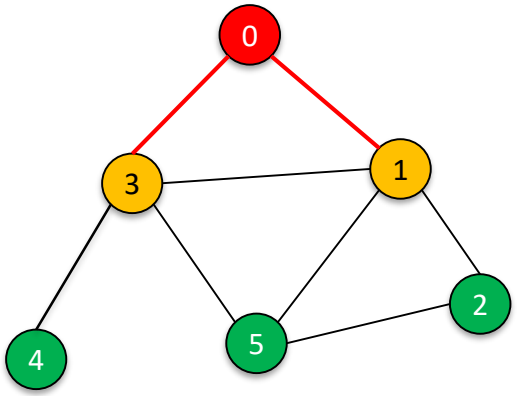
Thuật toán DFS có thể cài đặt theo 2 cách: dùng **đệ quy** hoặc **không đệ quy**.

Nếu cài đặt theo hướng không đệ quy thì cần sử dụng cấu trúc dữ liệu ngăn xếp (**stack**) để lưu các đỉnh đang xét.

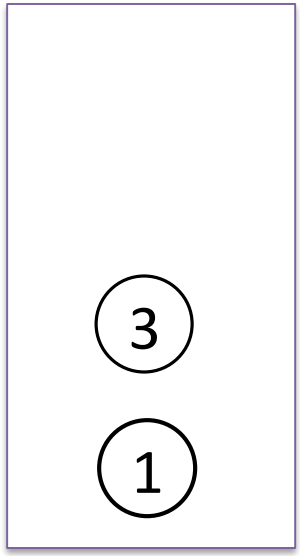


Ý tưởng thuật toán

Xuất phát từ 1 đỉnh bất kỳ, đi tới tất cả các đỉnh kề của đỉnh này và lưu đỉnh kề này lại.



Lưu vào kho



Đem các đỉnh trong kho ra xét như bước đầu

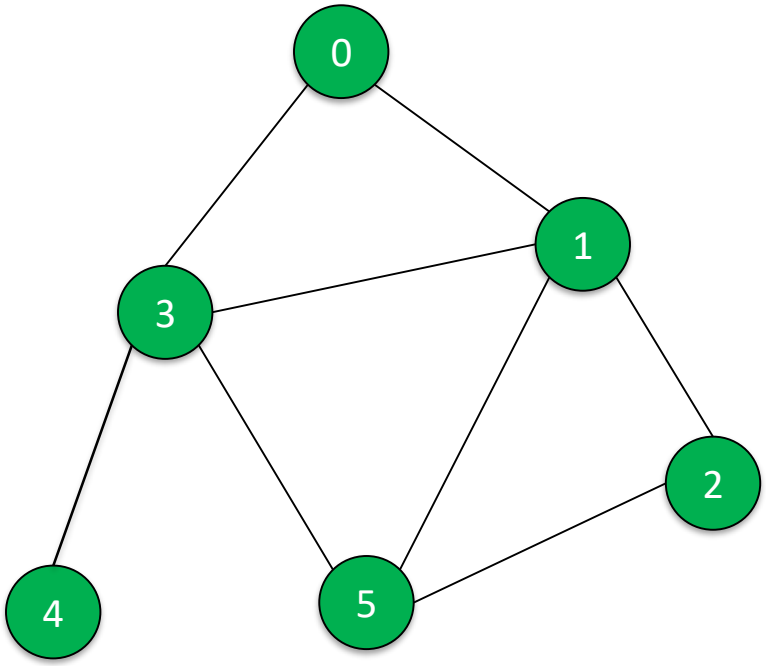
Lưu vết đường đi lại

Đỉnh	0	1	2	3
Lưu vết	-1	0	-1	0

Dừng thuật toán khi kho rỗng và in kết quả bài toán.

Bài toán minh họa

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



Adjacency Matrix

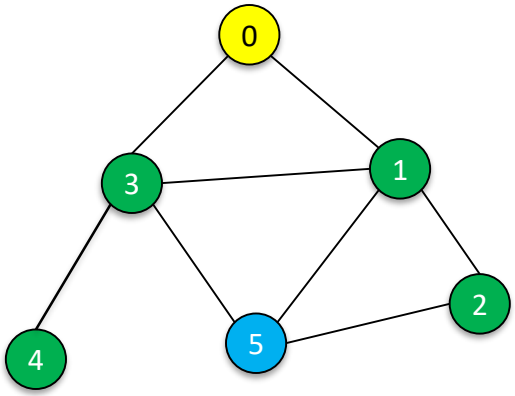
```
6

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
```

Edge List

```
6 8

0 1

0 3

1 2

1 3

1 5

2 5

3 4

3 5
```

CÀI ĐẶT THUẬT TOÁN DFS DÙNG STACK (KHÔNG ĐỆ QUY)

Bước 0: Chuẩn bị dữ liệu



Chuyển danh sách cạnh kề vào **graph**.

Đỉnh	0	1	2	3	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

Mảng đánh dấu các đỉnh đã xét **visited**.

Đỉnh	0	1	2	3	4	5
Trạng thái	false	false	false	false	false	false

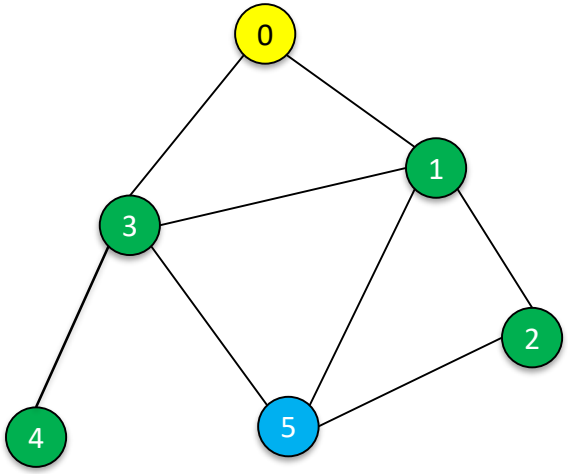
Mảng lưu vết đường đi **path**.

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	-1	-1	-1	-1	-1

Tạo ngăn xếp lưu các đỉnh đang xét **stack**. ...

Bước 0: Chuẩn bị dữ liệu (tiếp theo)

Đỉnh 0 là đỉnh bắt đầu đi. Bỏ đỉnh 0 vào ngăn xếp và đánh dấu đã xét đỉnh 0.



Mảng đánh dấu các đỉnh đã xét **visited**.

Đỉnh	**0**	1	2	3	4	5
Trạng thái	**true**	false	false	false	false	false

Ngăn xếp lưu các đỉnh đang xét **stack**.

0

0

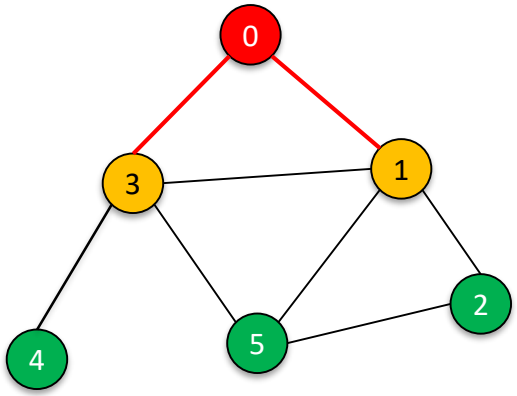
Bước 1: Chạy thuật toán lần 1

Big-O Blue

stack

0
0

Lấy **đỉnh 0** ra xét và tìm những đỉnh có kết nối với đỉnh 0 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	**0**	1	2	3	4	5
Đỉnh kề	**1, 3**	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	**1**	2	**3**	4	5
Trạng thái	true	**true**	false	**true**	false	false

stack

0	**1**
1	**3**

path

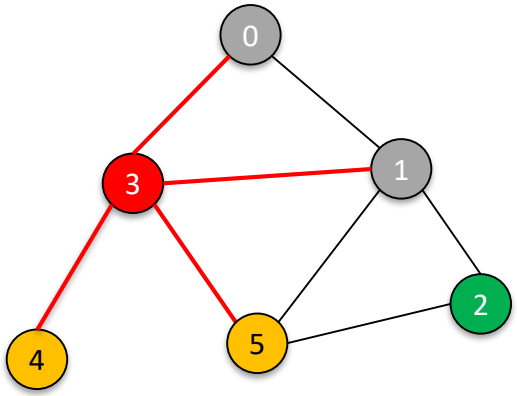
Đỉnh	0	**1**	2	**3**	4	5
Lưu vết	-1	**0**	-1	**0**	-1	-1

Bước 2: Chạy thuật toán lần 2

stack

0	1
1	3

Lấy **đỉnh 3** ra xét và tìm những đỉnh có kết nối với đỉnh 3 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	0	1	2	**3**	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	**0, 1, 4, 5**	3	1, 2, 3

visited

Đỉnh	0	1	2	3	**4**	**5**
Trạng thái	true	true	false	true	**true**	**true**

stack

0	**1**	**2**
1	**4**	**5**

path

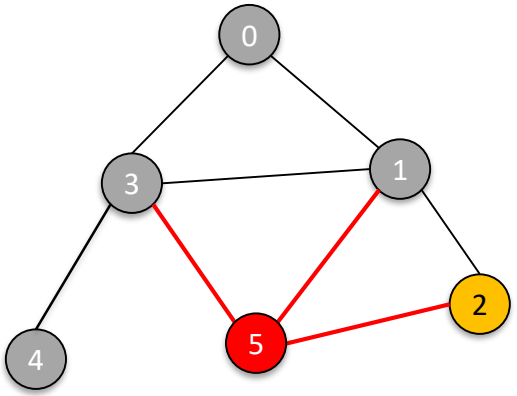
Đỉnh	0	1	2	3	**4**	**5**
Lưu vết	-1	0	-1	0	**3**	**3**

Bước 3: Chạy thuật toán lần 3

stack

0	1	2
1	4	5

Lấy **đỉnh 5** ra xét và tìm những đỉnh có kết nối với đỉnh 5 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	0	1	2	3	4	**5**
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	**1, 2, 3**

visited

Đỉnh	0	1	**2**	3	4	5
Trạng thái	true	true	**true**	true	true	true

stack

0	1	**2**
1	4	**2**

path

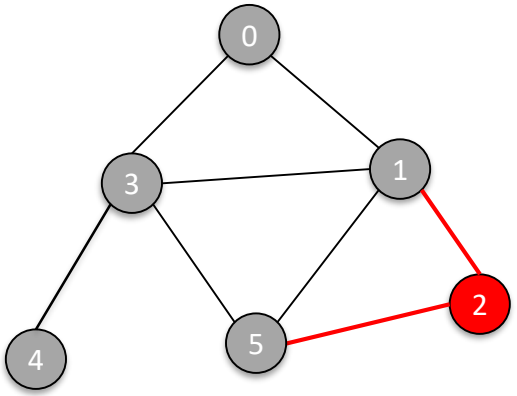
Đỉnh	0	1	**2**	3	4	5
Lưu vết	-1	0	**5**	0	3	3

Bước 4: Chạy thuật toán lần 4

stack

0	1	2
1	4	2

Lấy **đỉnh 2** ra xét và tìm những đỉnh có kết nối với đỉnh 2 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	0	1	**2**	3	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	**1, 5**	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	true	true	true	true	true

stack

0	1
1	4

path

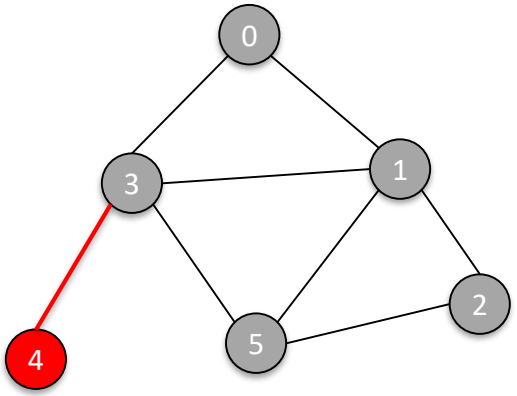
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	5	0	3	3

Bước 5: Chạy thuật toán lần 5

stack

0	1
1	4

Lấy **đỉnh 4** ra xét và tìm những đỉnh có kết nối với đỉnh 4 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	0	1	2	3	**4**	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	**3**	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	true	true	true	true	true

stack

0
1

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	5	0	3	3

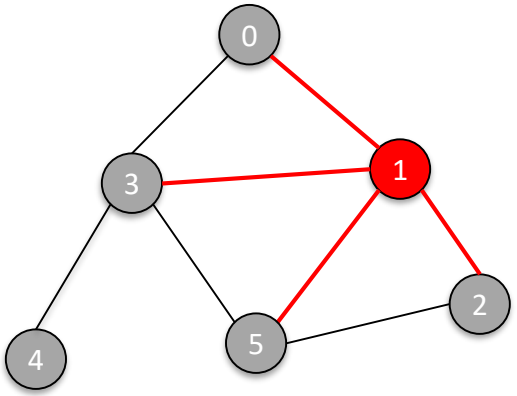
Bước 6: Chạy thuật toán lần 6

Big-O Blue

0

stack | 1 |

Lấy **đỉnh 1** ra xét và tìm những đỉnh có kết nối với đỉnh 1 mà chưa được xét, bỏ vào stack.



graph

Đỉnh	0	**1**	2	3	4	5
Đỉnh kề	1, 3	**0, 2, 3, 5**	1, 5	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
T. Thái	true	true	true	true	true	true

...

stack | ... |

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	5	0	3	3

Dừng thuật toán

Ngăn xếp rỗng, tất cả các đỉnh đều được xét → dừng thuật toán.

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	5	0	3	3

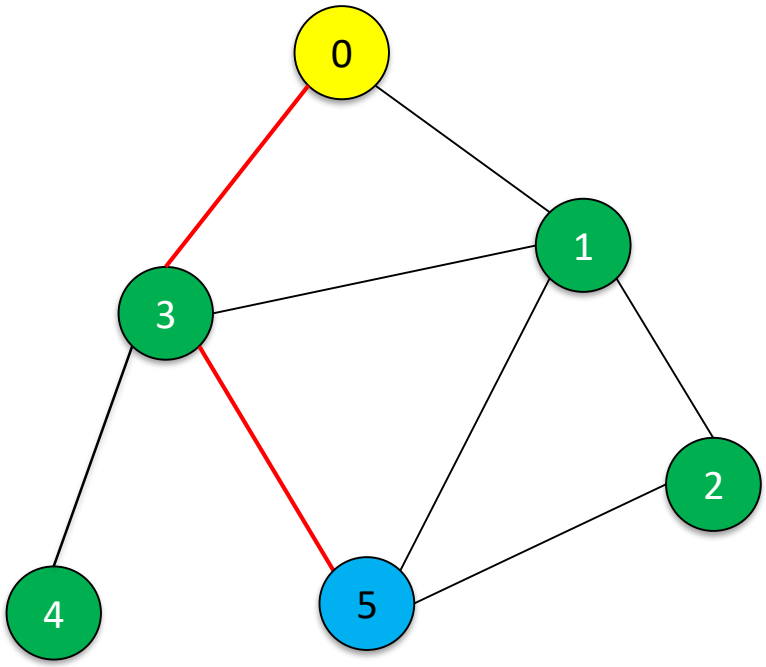
0 → 3 → 5

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

Đáp án bài toán (dùng stack)

0 → 3 → 5

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



Source Code DFS (dùng stack)

Khai báo thư viện và các biến toàn cục:

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

Source Code DFS (dùng stack)

Thuật toán chính DFS (part 1)

```
11. void DFS(int src)
12. {
13.     for (int i = 0; i < V; i++)
14.     {
15.         visited[i] = false;
16.         path[i] = -1;
17.     }
18.     stack<int> s;
19.     visited[src] = true;
20.     s.push(src);
21.     // to be continued
```

Source Code DFS (dùng stack)

Thuật toán chính DFS (part 2)

```
22.     while (!s.empty())
23.     {
24.         int u = s.top();
25.         s.pop();
26.         for (int i = 0; i < graph[u].size(); i++)
27.         {
28.             int v = graph[u][i];
29.             if (!visited[v])
30.             {
31.                 visited[v] = true;
32.                 s.push(v);
33.                 path[v] = u;
34.             }
35.         }
36.     }
37. }
```

Source Code DFS (dùng stack)

Hàm main để gọi thực hiện chương trình:

```
38. int main()
39. {
40.     int u, v;
41.     cin >> V >> E;
42.     for (int i = 0; i < E; i++)
43.     {
44.         cin >> u >> v;
45.         graph[u].push_back(v);
46.         graph[v].push_back(u);
47.     }
48.     int s = 0;
49.     int f = 5;
50.     DFS(s);
51.     printPath(s, f);
52.     return 0;
53. }
```

Source Code DFS (dùng stack)

Khai báo thư viện và các biến toàn cục:

```
1. MAX = 100
2. V = None
3. E = None
4. visited = [False for i in range(MAX)]
5. path = [0 for i in range(MAX)]
6. graph = [[] for i in range(MAX)]
```

Source Code DFS (dùng stack)

```
7.  def DFS(src):
8.      for i in range(V):
9.          visited[i] = False
10.         path[i] = -1
11.     s = []
12.     visited[src] = True
13.     s.append(src)
14.     while len(s) > 0:
15.         u = s.pop()
16.         for v in graph[u]:
17.             if not visited[v]:
18.                 visited[v] = True
19.                 s.append(v)
20.                 path[v] = u
```

Source Code DFS (dùng stack)

Hàm main để gọi thực hiện chương trình:

```
21. if __name__ == '__main__':
22.     V, E = map(int, input().split())
23.     for i in range(E):
24.         u, v = map(int, input().split())
25.         graph[u].append(v)
26.         graph[v].append(u)
27.     s = 0
28.     f = 5
29.     DFS(s)
30.     printPath(s, f)
```

Source Code DFS (dùng stack)

Khai báo thư viện và các biến toàn cục:

```
1. import java.util.*;
2. public class Main {
3.     private static ArrayList<ArrayList<Integer>> graph;
4.     private static int V;
5.     private static int E;
6.     private static ArrayList<Integer> path;
7.     private static ArrayList<Boolean> visited;
```

Source Code DFS (dùng stack)

Thuật toán chính DFS (part 1)

```
8.      private static void DFS(int src) {
9.          Stack<Integer> s = new Stack<>();
10.         path = new ArrayList<>();
11.         visited = new ArrayList<>();
12.         for (int i = 0; i < V; i++) {
13.             visited.add(false);
14.             path.add(-1);
15.         }
16.         visited.set(src, true);
17.         s.add(src);
18.         // to be continued
```

Source Code DFS (dùng stack)

Thuật toán chính DFS (part 2)

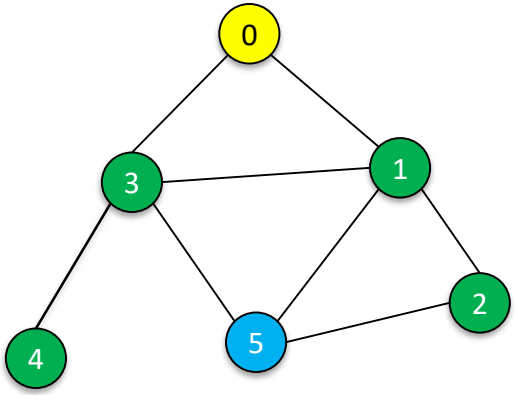
```
19.             while (!s.isEmpty()) {
20.                 int u = s.pop();
21.                 for (int i = 0; i < graph.get(u).size(); i++) {
22.                     int v = graph.get(u).get(i);
23.                     if (!visited.get(v)) {
24.                         visited.set(v, true);
25.                         path.set(v, u);
26.                         s.add(v);
27.                     }
28.                 }
29.             }
30.         }
```

Source Code DFS (dùng stack)

```
31.     public static void main(String[] args) {
32.         Scanner sc = new Scanner(System.in);
33.         V = sc.nextInt();
34.         E = sc.nextInt();
35.         graph = new ArrayList<>(V);
36.         for (int i = 0; i < V; i++)
37.             graph.add(new ArrayList<>());
38.         for (int i = 0; i < E; i++) {
39.             int u = sc.nextInt();
40.             int v = sc.nextInt();
41.             graph.get(u).add(v);
42.             graph.get(v).add(u);
43.         }
44.         int s = 0, f = 5;
45.         DFS(s);
46.         printPath(s, f);
47.     }
48. }
```

CÀI ĐẶT THUẬT TOÁN DFS BẰNG ĐỆ QUY

Bước 0: Chuẩn bị dữ liệu



Chuyển danh sách cạnh kề vào CTDL **graph**.

Đỉnh	0	1	2	3	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

Mảng đánh dấu các đỉnh đã xét **visited**.

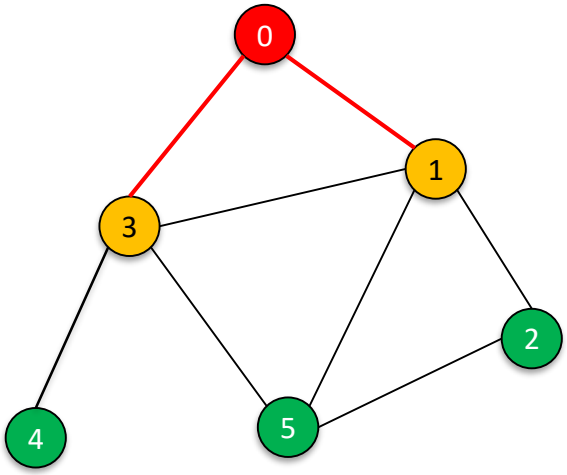
Đỉnh	0	1	2	3	4	5
Trạng thái	false	false	false	false	false	false

Mảng lưu vết đường đi **path**.

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	-1	-1	-1	-1	-1

Bước 1: Chạy thuật toán lần 1

Lấy **đỉnh 0 (đỉnh bắt đầu đi)** ra xét và tìm những đỉnh có kết nối với đỉnh 0 (những đỉnh chưa xét).



graph

Đỉnh	**0**	1	2	3	4	5
Đỉnh kề	**1, 3**	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	false	false	false	false	false

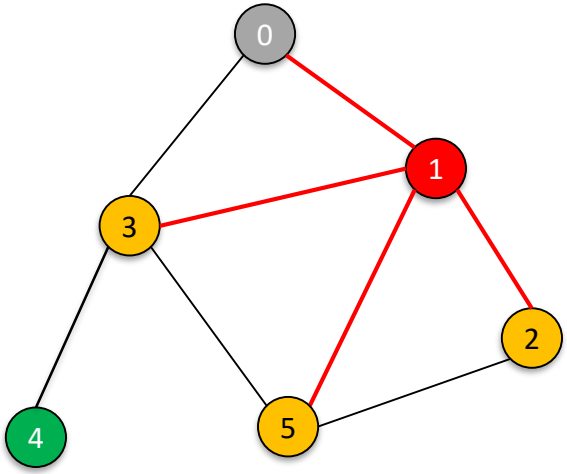
path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	-1	-1	-1	-1

Gọi đệ quy đỉnh 1.

Bước 2: Chạy thuật toán lần 2

Gọi đệ quy **đỉnh 1** tìm những đỉnh có kết nối với đỉnh 1 (những đỉnh chưa xét).



graph

Đỉnh	0	**1**	2	3	4	5
Đỉnh kề	1, 3	**0, 2, 3, 5**	1, 5	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	**true**	false	false	false	false

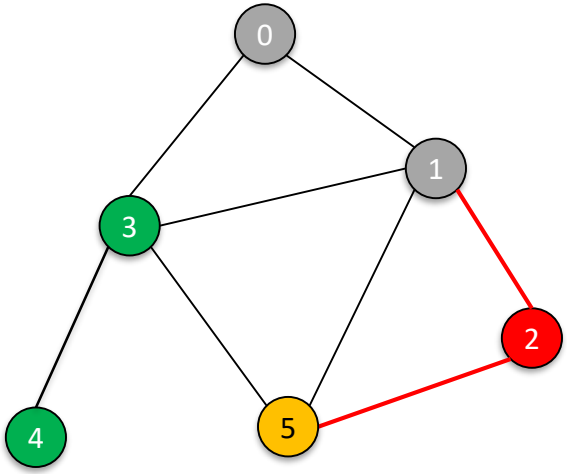
path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	**1**	-1	-1	-1

Gọi đệ quy đỉnh 2.

Bước 3: Chạy thuật toán lần 3

Gọi đệ quy **đỉnh 2** tìm những đỉnh có kết nối với đỉnh 2 (những đỉnh chưa xét).



graph

Đỉnh	0	1	**2**	3	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	**1, 5**	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	true	true	false	false	false

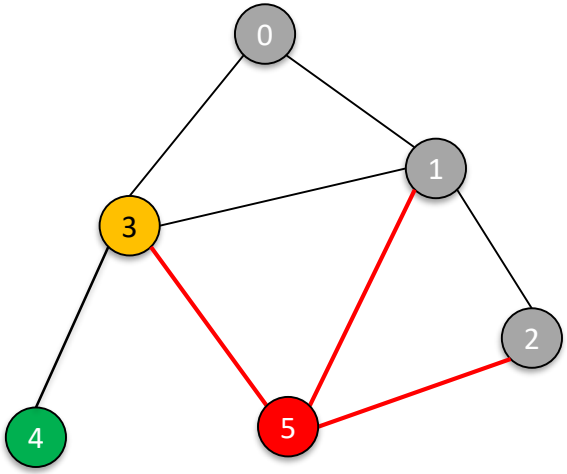
path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	-1	-1	2

Gọi đệ quy đỉnh 5.

Bước 4: Chạy thuật toán lần 4

Gọi đệ quy **đỉnh 5** tìm những đỉnh có kết nối với đỉnh 5 (những đỉnh chưa xét).



graph

Đỉnh	0	1	2	3	4	**5**
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	**1, 2, 3**

visited

Đỉnh	0	1	2	3	4	**5**
Trạng thái	true	true	true	false	false	**true**

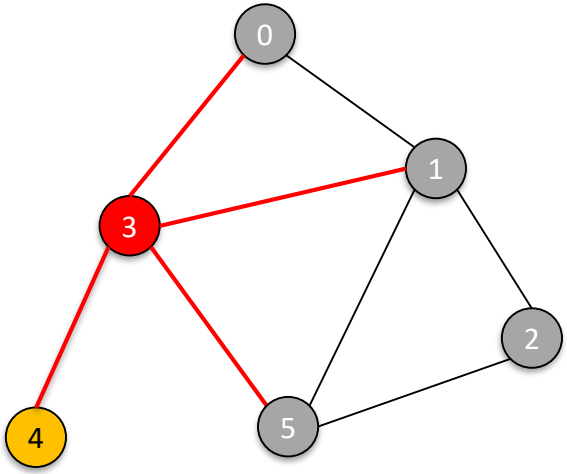
path

Đỉnh	0	1	2	**3**	4	5
Lưu vết	-1	0	1	**5**	-1	2

Gọi đệ quy đỉnh 3.

Bước 5: Chạy thuật toán lần 5

Gọi đệ quy **đỉnh 3** tìm những đỉnh có kết nối với đỉnh 3 (những đỉnh chưa xét).



graph

Đỉnh	0	1	2	**3**	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	**0, 1, 4, 5**	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	true	true	**true**	false	true

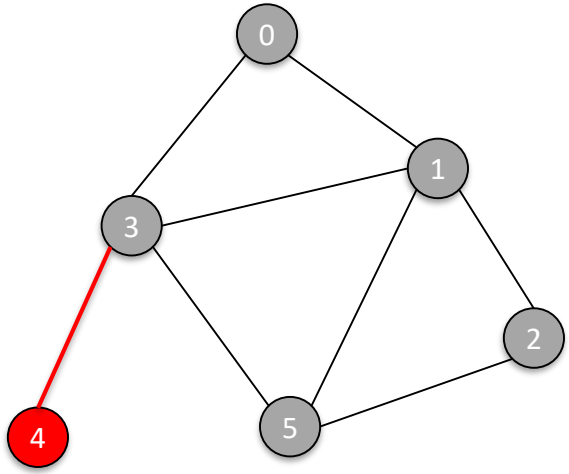
path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	5	**3**	2

Gọi đệ quy đỉnh 4.

Bước 6: Chạy thuật toán lần 6

Gọi đệ quy **đỉnh 4** tìm những đỉnh có kết nối với đỉnh 4 (những đỉnh chưa xét).



graph

Đỉnh	0	1	2	3	4	5
Đỉnh kề	1, 3	0, 2, 3, 5	1, 5	0, 1, 4, 5	3	1, 2, 3

visited

Đỉnh	0	1	2	3	4	5
Trạng thái	true	true	true	true	true	true

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	5	3	2

Đệ quy ngược lại các đỉnh 3, 5, 2, 1, 0 để kiểm tra.

Kết quả chạy thuật toán bằng đệ quy

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

path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	5	3	2

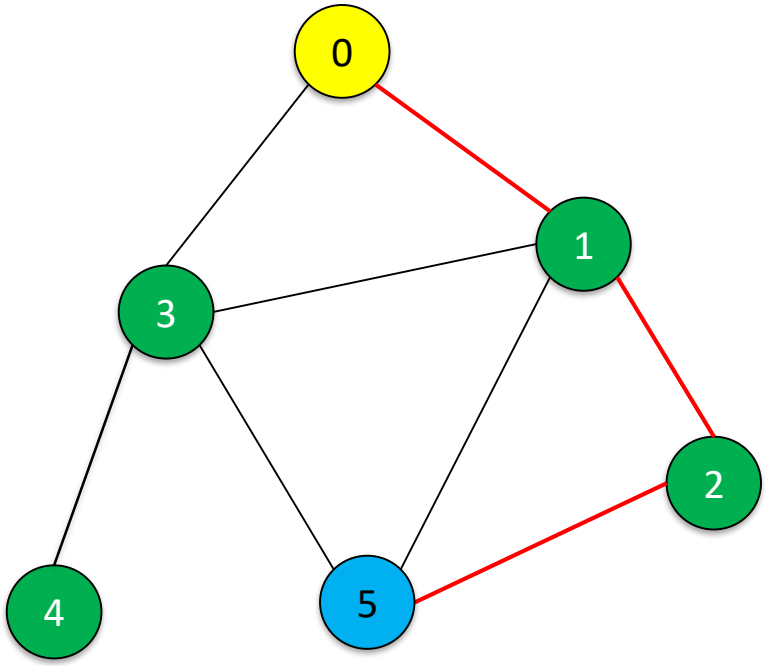
0 → 1 → 2 → 5

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

Đáp án bài toán (đệ quy)

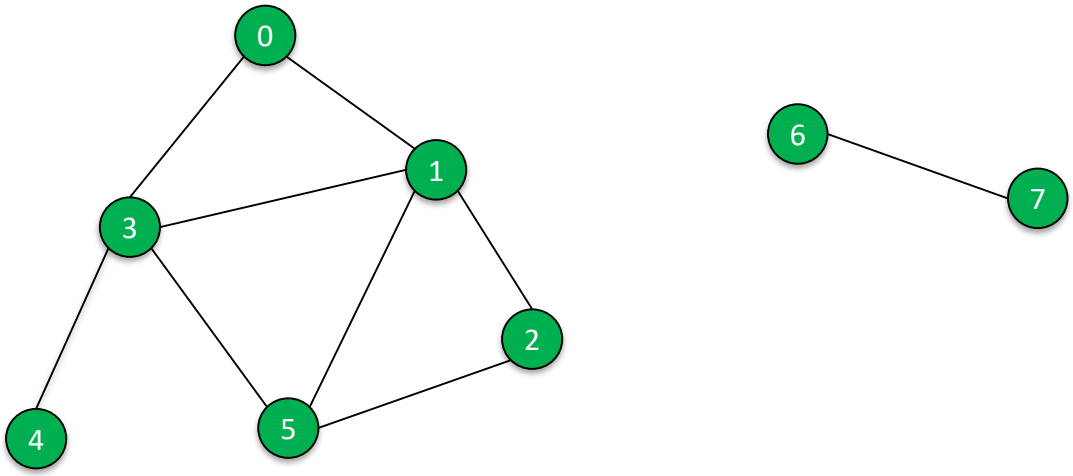
0 → 1 → 2 → 5

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?



Trường hợp chạy bắt đầu từ đỉnh 0.

path

Đỉnh	0	1	2	3	4	5	6	7
Lưu vết	-1	0	5	0	3	3	-1	-1

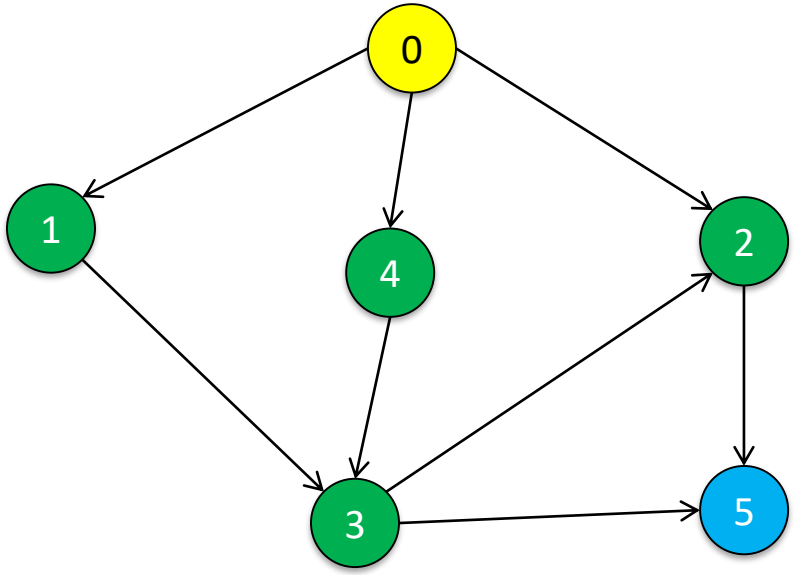
Trường hợp chạy bắt đầu từ đỉnh 6.

path

Đỉnh	0	1	2	3	4	5	6	7
Lưu vết	-1	-1	-1	-1	-1	-1	-1	6

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

Source Code DFS (Recursion)

Thuật toán chính DFS đệ quy:

```
1.  void DFSRecursion(int s)
2.  {
3.      visited[s] = true;
4.      for (int i = 0; i < graph[s].size(); i++)
5.      {
6.          int v = graph[s][i];
7.          if (!visited[v])
8.          {
9.              path[v] = s;
10.             DFSRecursion(v);
11.         }
12.     }
13. }
```

Source Code DFS (Recursion)

```
14. int main()
15. {
16.     int u, v;
17.     cin >> V >> E;
18.     for (int i = 0; i < E; i++)
19.     {
20.         cin >> u >> v;
21.         graph[u].push_back(v);
22.         graph[v].push_back(u);
23.     }
24.     int s = 0;
25.     int f = 5;
26.     for (int i = 0; i < V; i++)
27.     {
28.         visited[i] = false;
29.         path[i] = -1;
30.     }
31.     DFSRecursion(s);
32.     printPath(s, f);
33.     return 0;
34. }
```

Source Code DFS (Recursion)

Thuật toán chính DFS đệ quy:

```
1. def DFSRecursion(s):
2.     visited[s] = True
3.     for v in graph[s]:
4.         if not visited[v]:
5.             path[v] = s
6.             DFSRecursion(v)
```

Source Code DFS (Recursion)

Hàm main để gọi thực hiện chương trình:

```
7.  if __name__ == '__main__':
8.      V, E = map(int, input().split())
9.      for i in range(E):
10.         u, v = map(int, input().split())
11.         graph[u].append(v)
12.         graph[v].append(u)
13.     s = 0
14.     f = 5
15.     for i in range(V):
16.         visited[i] = False
17.         path[i] = -1
18.     DFSRecursion(s)
19.     printPath(s, f)
```

Source Code DFS (Recursion)

Thuật toán chính DFS đệ quy:

```
1.      private static void DFSRecursion(int s) {
2.          visited.set(s, true);
3.          for (int i = 0; i < graph.get(s).size(); i++) {
4.              int v = graph.get(s).get(i);
5.              if (!visited.get(v)) {
6.                  path.set(v, s);
7.                  DFSRecursion(v);
8.              }
9.          }
10.     }
```

Source Code DFS (Recursion)

Hàm main DFS đệ quy - part 1

```
11.     public static void main(String[] args) {
12.         Scanner sc = new Scanner(System.in);
13.         V = sc.nextInt();
14.         E = sc.nextInt();
15.         graph = new ArrayList<>(V);
16.         for (int i = 0; i < V; i++)
17.             graph.add(new ArrayList<>());
18.         for (int i = 0; i < E; i++) {
19.             int u = sc.nextInt();
20.             int v = sc.nextInt();
21.             graph.get(u).add(v);
22.             graph.get(v).add(u);
23.         }
```

Source Code DFS (Recursion)

Hàm main DFS đệ quy - part 2

```
24.             int s = 0, f = 5;
25.             path = new ArrayList<>();
26.             visited = new ArrayList<>();
27.             for (int i = 0; i < V; i++) {
28.                 visited.add(false);
29.                 path.add(-1);
30.             }
31.             DFSRecursion(s);
32.             printPath(s, f);
33.         }
```

Hỏi đáp