

# LECTURE 05

## BREADTH FIRST SEARCH ALGORITHM



Phạm Nguyễn Sơn Tùng

Email: [sontungtn@gmail.com](mailto:sontungtn@gmail.com)

# Thuật toán BFS là gì?

Thuật toán **Breadth First Search** (BFS) là thuật toán tìm kiếm theo chiều rộng trên đồ thị **vô hướng** hoặc **có hướng**, **không** trọng số, giải quyết bài toán:

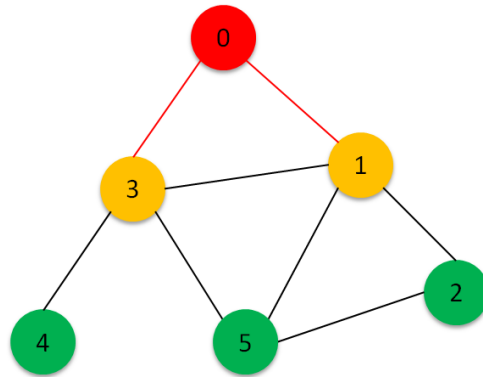
- Tìm kiếm đường đi ngắn nhất từ một đỉnh bất kỳ tới tất cả các đỉnh khác trong đồ thị (nếu 2 đỉnh thuộc cùng thành phần liên thông với nhau).
- **Luôn tìm được đường đi ngắn nhất.**

**Độ phức tạp:**  $O(V + E)$

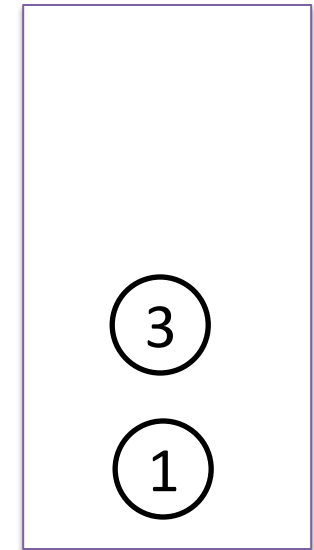
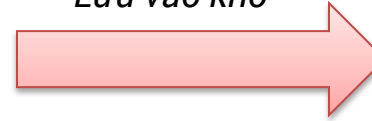
- Tập hợp  $V$  (Vertices) những phần tử gọi là đỉnh của đồ thị.
- Tập hợp  $E$  (Edges) những phần tử gọi là cạnh của đồ thị.

# Ý tưởng thuật toán

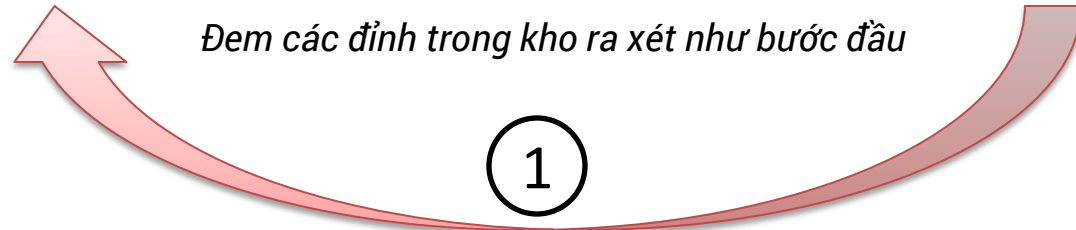
Xuất phát từ 1 đỉnh bất kỳ, đi tới tất các đỉnh kề của đỉnh này và lưu đỉnh kề này lại.



Lưu vào kho



Đem các đỉnh trong kho ra xét như bước đầu



1

Lưu vết đường đi lại

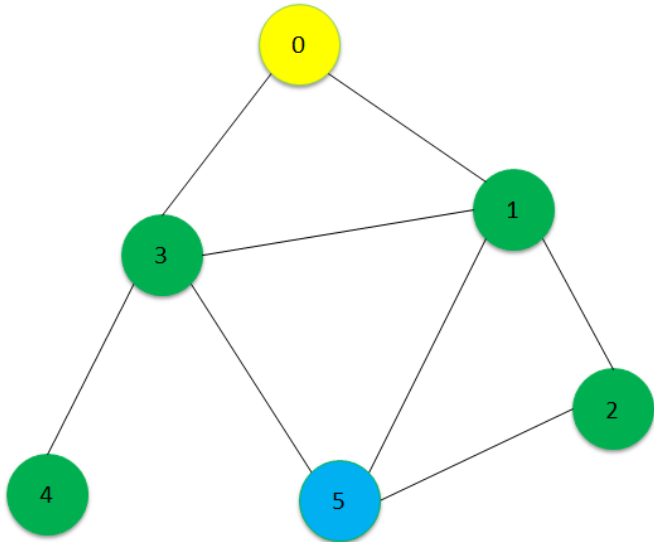
Đỉnh	0	1	2	3
Lưu vết	-1	0	-1	0



Dùng 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 ngắn nhất** từ đỉnh 0 đến đỉnh 5.



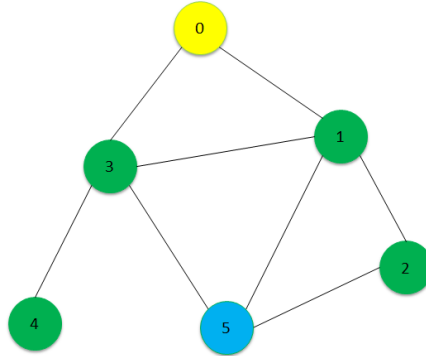
*Adjacency Matrix*

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

# 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**.

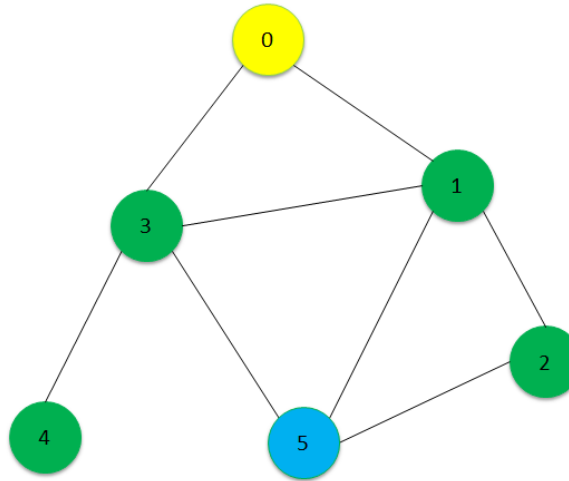
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	-1	-1	-1	-1	-1

Tạo hàng đợi lưu các đỉnh đang xét **queue**.

...

# Bước 0: chuẩn bị dữ liệu (tiếp theo)

**Đỉnh 0** là đỉnh bắt đầu đi. Bỏ đỉnh 0 vào hàng đợi 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

Hàng đợi lưu các đỉnh đang xét **queue**.

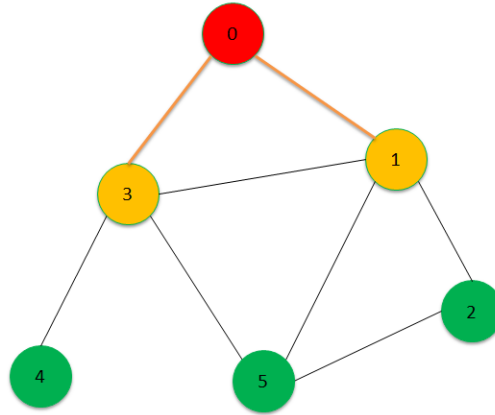


# Bước 1: Chạy thuật toán lần 1

queue

0
0

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



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

queue

0	1
1	3

path

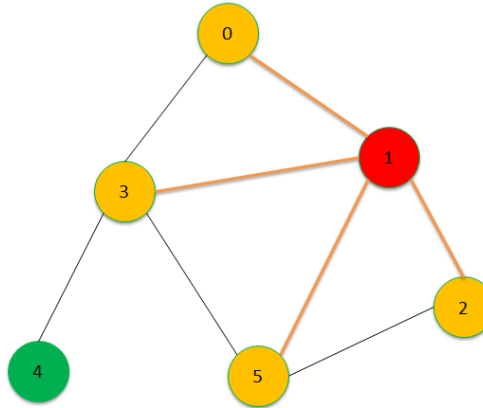
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	-1	0	-1	-1

# Bước 2: Chạy thuật toán lần 2

queue

0	1
1	3

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



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

queue

0	1	2
3	2	5

path

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

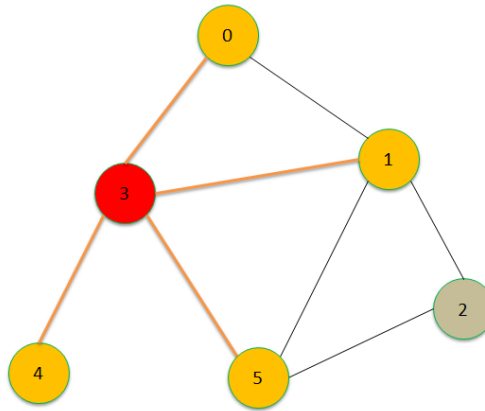


# Bước 3: Chạy thuật toán lần 3

queue

0	1	2
3	2	5

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



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

queue

0	1	2
2	5	4

path

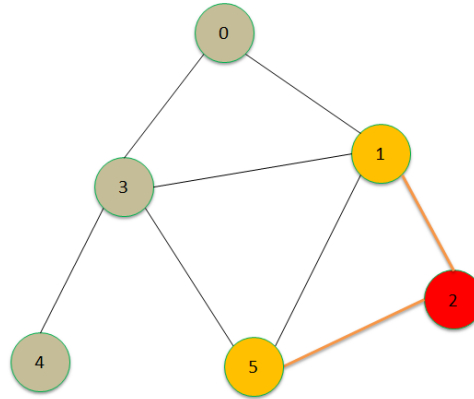
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	0	3	1

# Bước 4: Chạy thuật toán lần 4

queue

0	1	2
2	5	4

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



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

queue

0	1
5	4

path

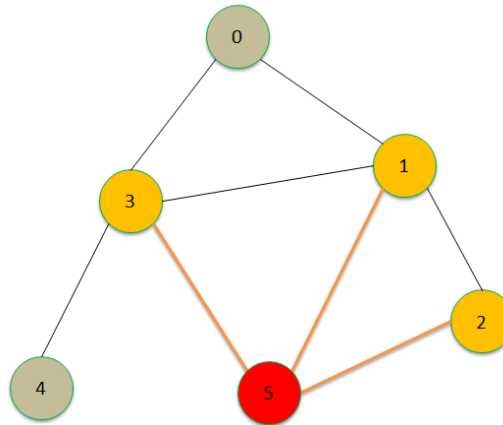
Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	0	3	1

# Bước 5: Chạy thuật toán lần 5

queue

0	1
5	4

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



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

queue

0
4

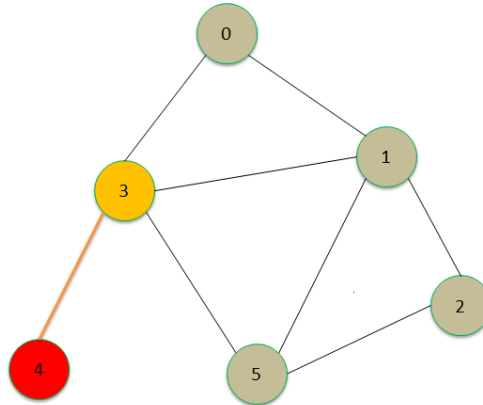
path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	0	3	1

# Bước 6: Chạy thuật toán lần 6

0  
queue 4

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



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

queue

...
...


path

Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	0	3	1

# Dừng thuật toán

Hàng đợi rỗng, tất cả các đỉnh đều được xét  $\rightarrow$  dừng thuật toán.

path



Đỉnh	0	1	2	3	4	5
Lưu vết	-1	0	1	0	3	1

**Kết quả**

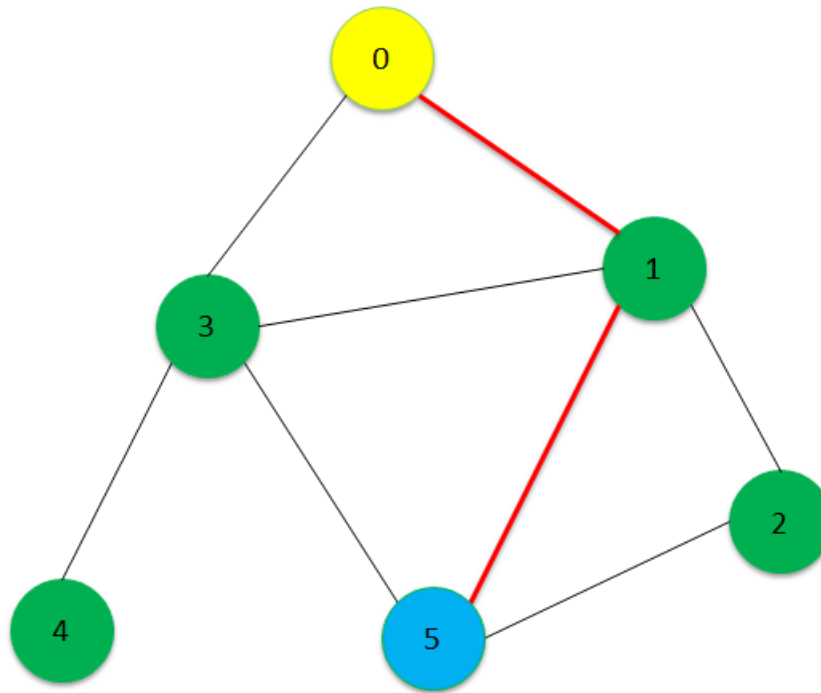
**$0 \rightarrow 1 \rightarrow 5$**

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

# Đáp án bài toán

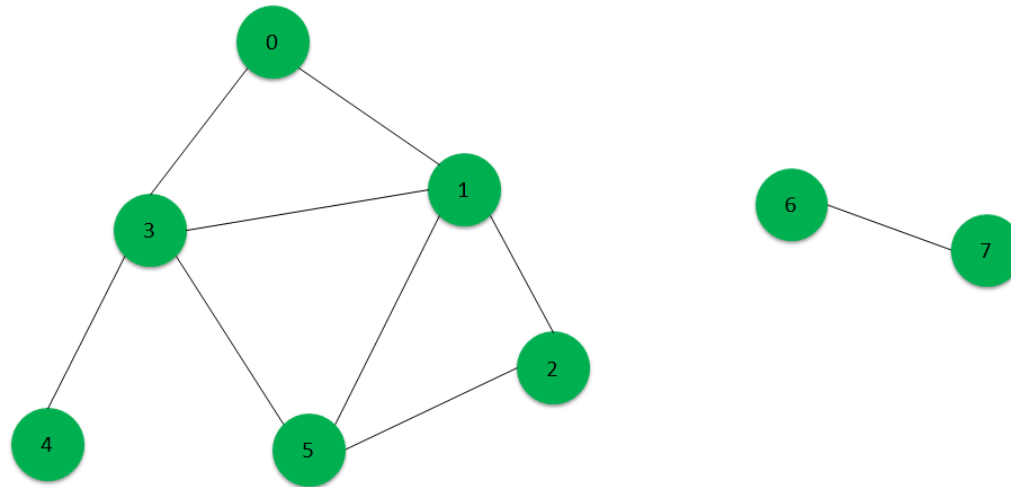
$0 \rightarrow 1 \rightarrow 5$

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



# Lưu ý khi sử dụng BFS

Khi 2 đỉnh cần tìm đường đi ngắn nhất 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	1	0	3	1	-1	-1

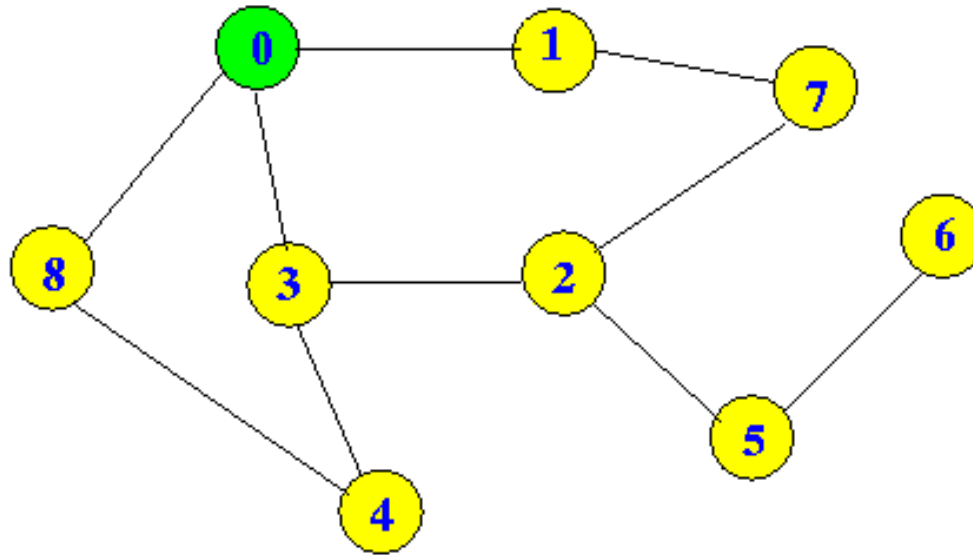
Trường hợp chạy bắt đầu từ đỉnh 6.

**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 thứ tự duyệt BFS và đường đi ngắn nhất từ 0 đến 5 của đồ thị sau:



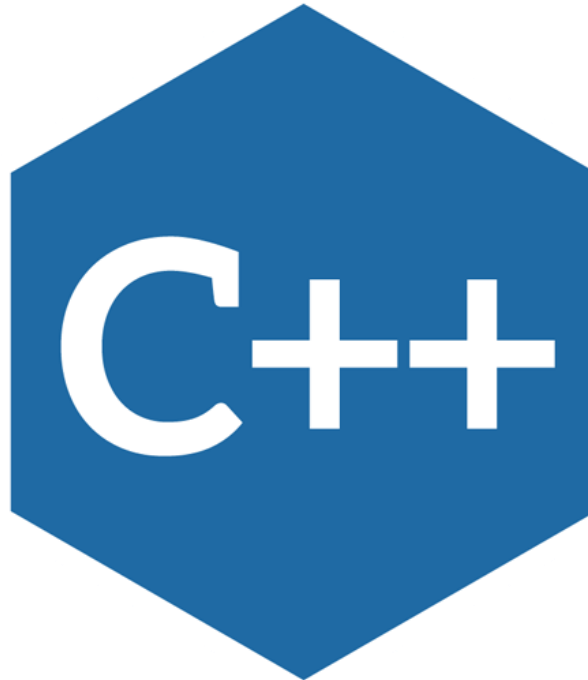
**Kết quả**

**0 → 3 → 2 → 5**

Thứ tự duyệt BFS là **0, 1, 3, 8, 7, 2, 4, 5, 6.**



# MÃ NGUỒN MINH HỌA BẰNG C++



# Source Code BFS C++

Khai báo thư viện và các biến toàn cục:

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

# Source Code BFS C++

## Thuật toán chính BFS (part 1)

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

# Source Code BFS C++

## Thuật toán chính BFS (part 2)

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

# Source Code BFS 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 BFS 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 BFS 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 BFS C++

Hàm main để gọi thực hiện:

```
int main() {  
    freopen("INPUT.INP", "rt", stdin);  
    int 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;  
    BFS(s);  
    printPath(s, f);  
    return 0;  
}
```



# MÃ NGUỒN MINH HỌA BẰNG PYTHON



# Source Code BFS Python

Khai báo thư viện và các biến toàn cục:

```
from queue import Queue

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

# Source Code BFS Python

## Thuật toán chính BFS

```
def BFS(s):  
    for i in range(V):  
        visited[i] = False  
        path[i] = -1  
  
    q = Queue()  
    visited[s] = True  
    q.put(s)  
    while q.empty() == False:  
        u = q.get()  
        for v in graph[u]:  
            if visited[v] == False:  
                visited[v] = True  
                q.put(v)  
                path[v] = u
```

# Source Code BFS 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 BFS 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 BFS 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  
    f = 5  
    BFS(s)  
    printPath(s, f)
```

# MÃ NGUỒN MINH HỌA BẰNG JAVA



# Source Code BFS Java

Khai báo thư viện:

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

Khai báo biến toàn cục (thuộc class Main)

```
private static ArrayList<ArrayList<Integer>> graph;  
private static int V;  
private static int E;  
private static ArrayList<Integer> path;  
private static ArrayList<Boolean> visited;
```



# Source Code BFS Java

## Thuật toán chính BFS (part 1)

```
public static void BFS(int s) {  
    Queue<Integer> q = new LinkedList <Integer>();  
    path = new ArrayList <Integer> ();  
    visited = new ArrayList <Boolean> ();  
    for (int i = 0; i < V; i++) {  
        visited.add(false);  
        path.add(-1);  
    }  
    visited.set(s, true);  
    q.add(s);  
    // to be continued
```

# Source Code BFS Java

## Thuật toán chính BFS (part 2)

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

# Source Code BFS Java

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

```
public static void printPath(int s, int f) {  
    if (s == f) {  
        System.out.print(s);  
        return;  
    }  
    if (path.get(f) == -1) {  
        System.out.print("No path");  
        return;  
    }  
    // to be continued
```

# Source Code BFS Java

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

```
ArrayList<Integer> b = new ArrayList<Integer> ();
int m = 0;
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));
    System.out.print(" ");
}
}
```

# Source Code BFS 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.println("No path");  
        else {  
            printPathRecursion(s, path.get(f));  
            System.out.print(f + " ");  
        }  
    }  
}
```

# Source Code BFS Java

Hàm main để gọi thực hiện:

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

# Hỏi đáp

