Dear reader, welcome to the article on the problem named **‘**[**Iterative Depth First Traversal**](https://www.pepcoding.com/resources/online-java-foundation/graphs/iterative-dft-official/ojquestion)**’.**

***Problem Statement:***

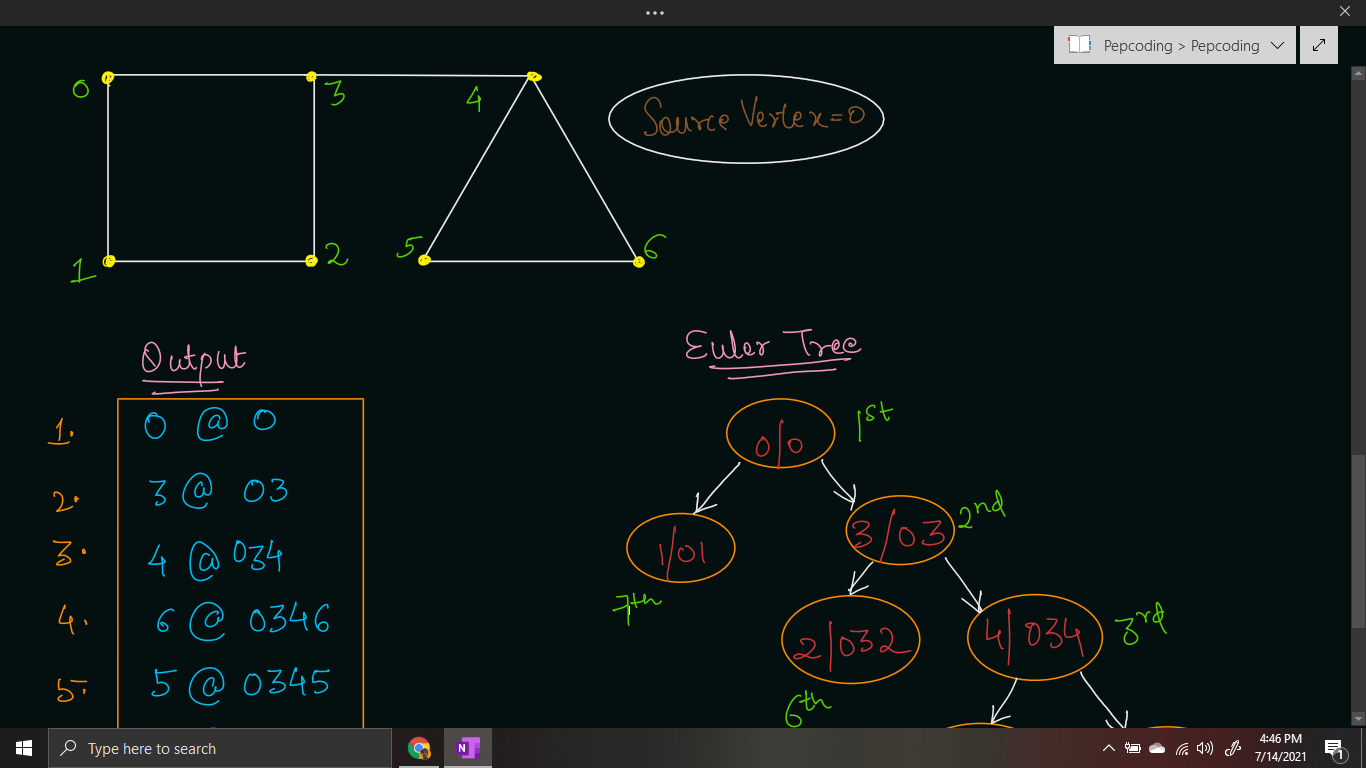
* You are given a graph, and a source vertex.
* You are required to do an iterative depth first traversal and print which vertex is reached via which path, starting from the source.

*Note*: Input is given in the form of adjacency list.

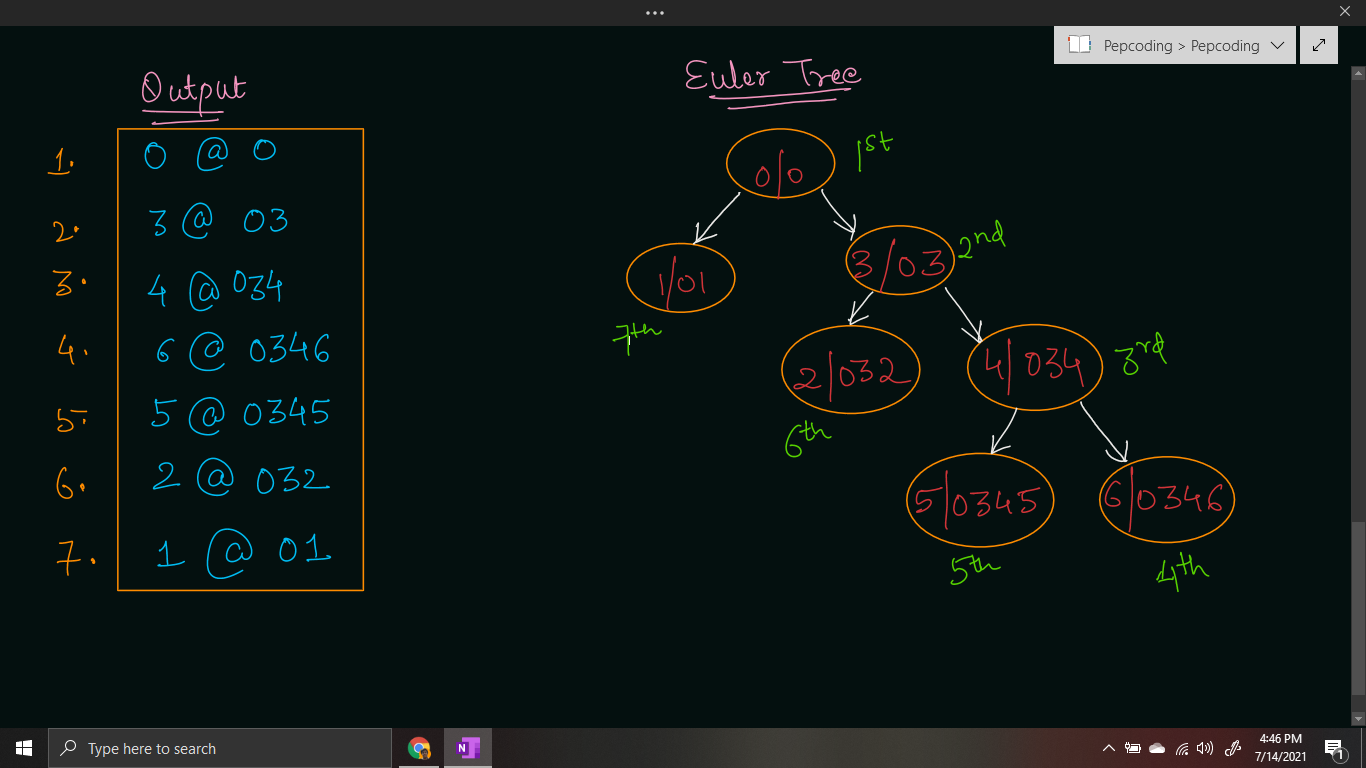
Iterative depth first traversal should mimic "Reverse pre order" i.e. nbr with highest value should be visited first and should be printed on the way down in recursion.

***Example:***

Input Graph:

****

Output:



***Solution***

Before moving onto the solution, these are the two prerequisites:

1. **Depth First Search (**[**DFS**](https://www.pepcoding.com/resources/online-java-foundation/graphs/has-path-official/ojquestion)**) Traversal using Recursion**

We know from the recursion section, recursive calls are pushed into a function call ***stack*** (in RAM), and when the function gets executed, the recursive call gets popped out from the call stack.

1. **Breadth First Search (**[**BFS**](https://www.pepcoding.com/resources/online-java-foundation/graphs/bfs-graph-official/ojquestion)**) Traversal using Queue**

We also know that for breadth first search traversal, we take use of the queue data structure.

Combining both of the knowledge, for an iterative version of Depth First Search Traversal, we just need to replace the queue data structure with ***stack*** data structure.

So, we will push the source node into a stack. Now, we will pop one element from the stack, visit it, and push all it’s unvisited neighbours into the stack. We will keep on doing it until the stack becomes empty.

***Pseudo Code***

1. Initialize an empty stack of Pair objects.
2. Initialize a visited array of size n (n = number of vertices), initially all false (no vertex has been visited yet).
3. Push a pair of source node’s value and a string path so far as source node initially, into the stack
4. Run a while loop until the stack becomes empty
   1. Pop the top element from the stack and store it into a Pair object rem.
   2. If the node is already visited, then continue.
   3. Else visit the node, i.e. mark visited[rem.v] = true
   4. Print the node according to the output format, i.e: ***System.out.println(rem.v + "@" + rem.psf);***
   5. Traverse through all the neighbouring vertices of the current node, and push the unvisited ones into the stack.

***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int vtces = Integer.parseInt(br.readLine());

ArrayList<Edge>[] graph = new ArrayList[vtces];

for (int i = 0; i < vtces; i++) {

graph[i] = new ArrayList<>();

}

int edges = Integer.parseInt(br.readLine());

for (int i = 0; i < edges; i++) {

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

int src = Integer.parseInt(br.readLine());

Stack<Pair> stack = new Stack<>();

stack.push(new Pair(src, src + ""));

boolean[] visited = new boolean[vtces];

while(stack.size() > 0){

Pair rem = stack.pop();

if(visited[rem.v] == true){

continue;

}

visited[rem.v] = true;

System.out.println(rem.v + "@" + rem.psf);

for (Edge e : graph[rem.v]) {

if (visited[e.nbr] == false) {

stack.push(new Pair(e.nbr, rem.psf + e.nbr));

}

}

}

}

static class Pair {

int v;

String psf;

Pair(int v, String psf){

this.v = v;

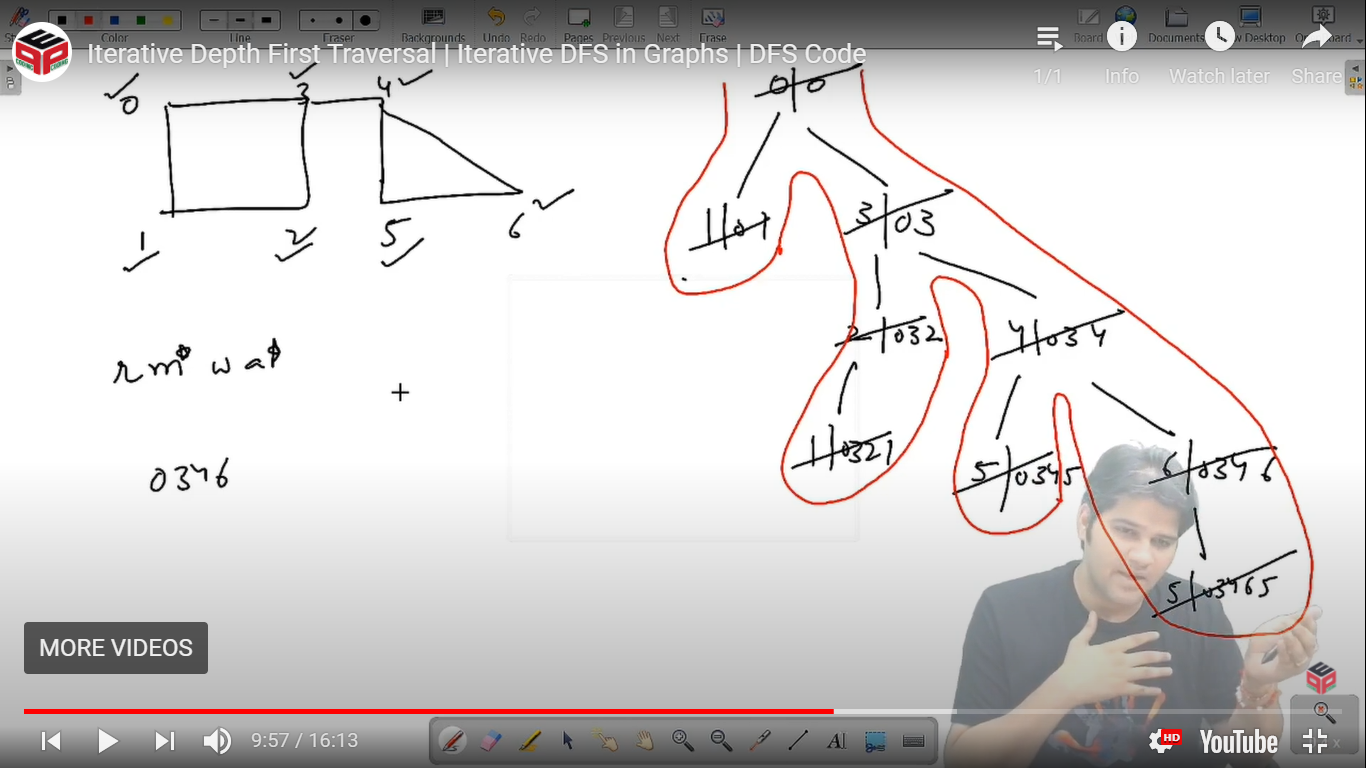
this.psf = psf;

}

}

}

This code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=iUtmQ66IC_0&list=TLGGQAmQLeJmDPoxNDA3MjAyMQ). Do check it out to understand the concept completely.



* What is the ***time complexity*** of the above code?

We have just replaced the queue data structure with stack, and the rest of the code remains the same. Hence, the time complexity will also remain O(N + E) where N = number of vertices and E = number of edges.

* What is the ***space complexity*** of the above code ?

We are using a stack data structure, which will store at max N nodes. Also, we are using a visited array of size N. Hence the total space complexity is o(N).

**Follow Up:**

**Q)** What should we prefer: DFS using recursion or DFS using stack data structure?

**R)** If the complexity of the implementation is what matters, then definitely DFS using recursion is a better choice.

But, since you must know the BFS traversal algorithm, remembering DFS traversal using a stack will not be a problem.

Infact, there can be cases where DFS using our own stack will be beneficial.

Do you know that Java allows at maximum 10000 depth of the function call stack?

Yes, large graphs, where depth of DFS euler tree will grow beyond 10000 levels will result in ***STACK OVERFLOW*** if we use recursion.

But, if we will use our own stack, then since it is built in the ***heap memory***, there will be no problem of stack overflow in such less depth.

Hence, it is advised that if you are using Java as your programming language, then prefer DFS using stack over DFS using recursion.

*Note*: Similarly, in the *dynamic programming* section, you will get to hear that java programmers should prefer tabulation over memoization. The reason is the same, i.e. the stack overflow problem.

Hope that you liked the article on ***’Iterative Depth First Search Traversal’***.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by: [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)