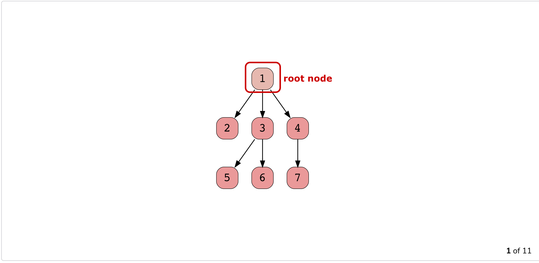
What is Depth First Search?

Depth First Search is a method used to traverse and search all nodes in a graph. The algorithm allows us to determine if two nodes, node a and node b, have a path between them. This process starts from the **root node** and then traverses all through that **branch** until it reaches the **leaf**, the last node with no other children, and then **backtracks**. This continues until all nodes have been traversed. The illustration below explains the process of `DFS` in a directed graph.



Implementing the Code

The code below shows how to implement this process using recursion. First, let's examine the code, and then we will move on to its explanation.

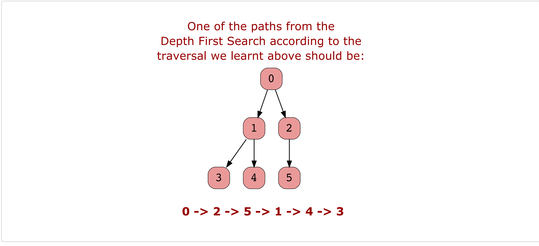
You must modify the edges by using `addEdge`, and the number of vertices `nVertices` to create your own graph, `g`.Then, run `DFSRecursion` on these variables.

```
class ExampleClass {

    static class Graph {
        int numVertices;
        LinkedList<Integer>[] tempList;

        Graph(int numVertices) {
            this.numVertices = numVertices;
            tempList = new LinkedList[numVertices];
            for (int i = 0; i < numVertices ; i++) {
                tempList[i] = new LinkedList<>();
            }
        }

        // Method to add an edge between 2 nodes in the Graph
        // fromNode 2 toNode 5 ==> 2 -> 5
        public void addEgde(int fromNode, int toNode) {
            tempList[fromNode].addFirst(toNode);
        }

        public void DFSRecursion(int startVertex) {
            boolean[] visitedArr = new boolean[numVertices];
            dfs(startVertex, visitedArr);
        }

        // DFS Recursion takes place here
        public void dfs(int start, boolean [] visitedArr) {
            visitedArr[start] = true;
```

Run | Save | Reset

Understanding the Code

The code snippet above runs a **Depth First Search** on the graph created by the illustration below.



The recursive code can be broken down into two parts. The first is the recursive method and the second is the main where the method is called. Before we examine the actual `dfs` function, let's just quickly go over some of the details from the class `Graph`.

- The class `Graph` from **lines 3 to 13** has two variables; `numVertices` which depicts the total number of vertices in the graph and a list that contains elements of type **Integer**, called the `tempList`. The *addEdge* method from **line 17 to 19** takes in two nodes as its arguments and creates an edge between them. This class also has the `DFSRecursion` and the `dfs` methods, which will run the Depth-first Search on the graph.

Now let's look at the two divisions of code.

Driver Method

First, let's look at the driver method which calls the recursive code.

- In the `main` method, on **line 43**, the `nVertices` variable is initialized. This depicts the number of vertices in the graph.
- On **line 45**, a new Graph is created with the number of vertices, named `nVertices`, as its argument.
- From **lines 47 to 51**, multiple edges are created where each edge takes 2 integers, e.g, `fromNode` 2 to a `toNode` 5, to create an edge that points from **Node 2** to **Node 5**.
- On **line 54**, the recursive method `DFSRecursion` is called with a **root node** as its argument.

Recursive Method

Now that we have outlined the recursive code is called, let's look at `DFSRecursion` and `dfs` in detail. This is the code segment from **lines 21 to line 37** in the code snippet above.

Let's first go over the method `DFSRecursion` which takes the root vertex as an argument and creates a type `boolean` array, called `visitedArr`.This array marks the vertices that have been visited and then calls the main recursive `dfs` method.

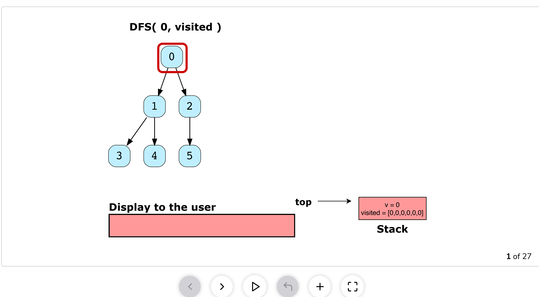
The `dfs` method takes in two arguments, the vertex `startVertex` which needs to be traversed as the first and the `visitedArr` as the second argument. This stores whether or not a node has been traversed.

On **line 28**, the dfs method first marks the vertex that has been passed as true in the `visitedArr` array. This serves to maintain the visited status of all the vertices.**Line 30** then prints that vertex number to the console.

Recursive Case

- **Lines 32 to 36** serves as the `for` loop, which traverses the `tempList` containing all the nodes of the graph.
  - The *if condition* on **line 34** states that if the `toNode` destination node has not been visited, it must call the `dfs` method with new parameters; The `start` vertex thus becomes `toNode`, the destination for the edge.
  - If that condition is not met, the loop continues until the end of `edges` is reached.

Understanding through Stack



Now that you have learned how to carry out a depth-first search on a directed graph using recursion in Java, the next lesson will teach you how to carry out a topological sort on a graph.