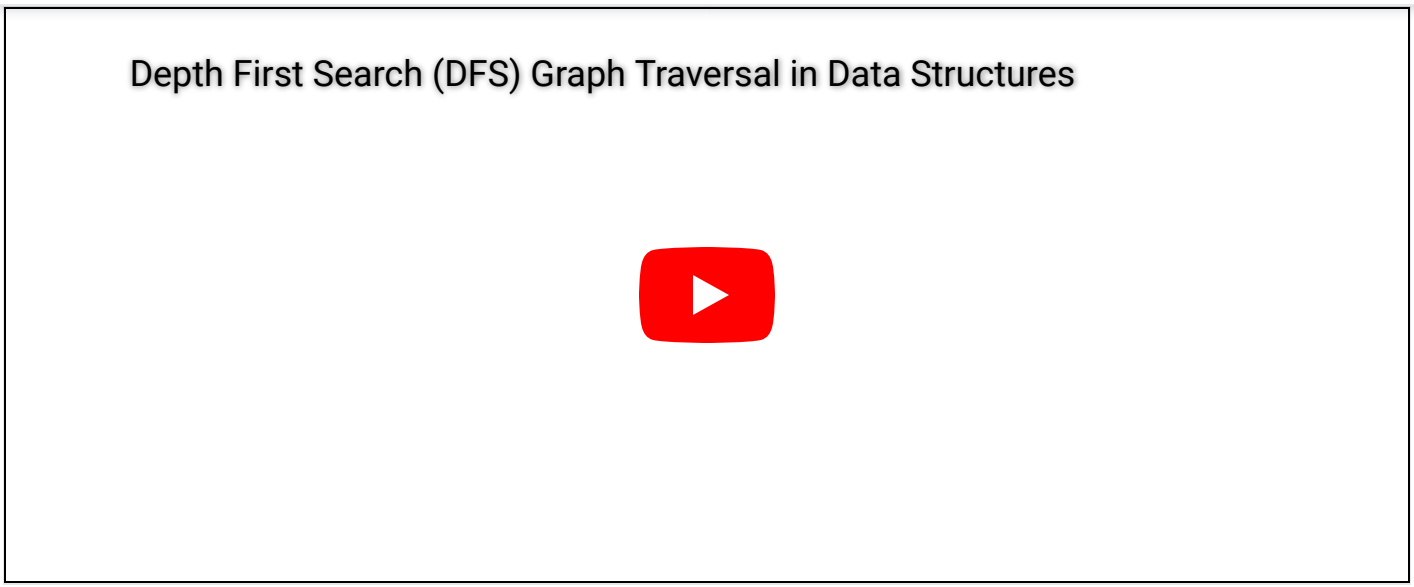


Show Course Contents

Overview Q&A Downloads Announcements

Depth First Search (DFS) Graph Traversal in Data Structures

In the last lecture, we completed learning about the first method of graph traversal which was the Breadth-First Search as well as its implementation in C. Today, we'll be dealing solely with the Depth First Search abbreviated as DFS in greater detail. Before we actually move on to writing the algorithm of the Depth First Search or programming its implementation, we'll first visualize how Depth First Search works.

As we know, graph traversal becomes impossible when done manually for very large graphs. Therefore, some algorithms are used to automate this task. One such algorithm we learned was BFS, and another one, which we would learn today is the Depth First Search.

What is Depth First Search?

In Depth First Search, we start with a node and start exploring its connected nodes, keeping on suspending the exploration of previous vertices. And those suspended nodes are explored once we finish exploring the node below. And this way we

explore all the nodes of the graph.

If we now compare what we are doing differently from BFS, we would find that there is one major thing that shouldn't go unnoticed.

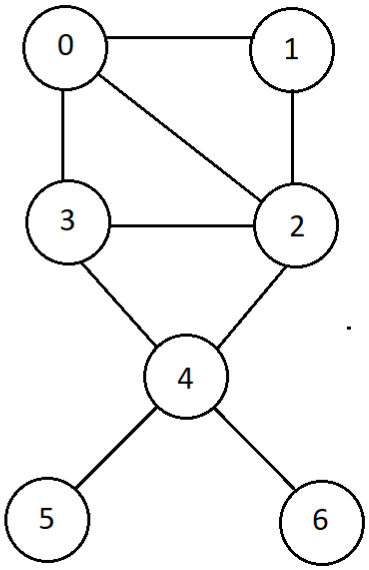
In BFS, we explore all the nodes connected to a node, then explore the nodes we visited in a horizontal manner, while in DFS, we start with the first connected node, and similarly go deep, so this looks like visiting the nodes vertically. This might sound confusing for now, so let's make it more intuitive for you to understand.

DFS Procedure:

The procedure we follow to accomplish the depth-first search traversal of a graph is:

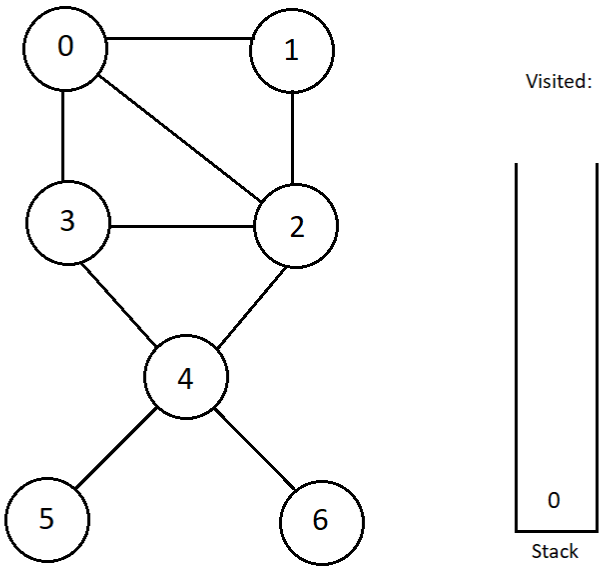
1. Let's define a stack named which would hold the nodes we'll be suspending to explore one by one later. And stack, if you remember, is a data structure in which the element you push the last comes out the first. Choose any node as the source node and push it into the top of the stack we created. We would maintain another array holding the status of whether a node is already **visited** or not.
2. Take the top item of the stack and mark it visited or add it to the visited list.
3. Create a list of the nodes directly connected to the vertex we visited. Push the ones which are still not visited into the stack.
4. Repeat steps 2 and 3 until the stack is not empty.

To understand the procedure of the Depth First Search, consider the graph I've illustrated below. For better contrast, I took the same graph as in the last lecture.

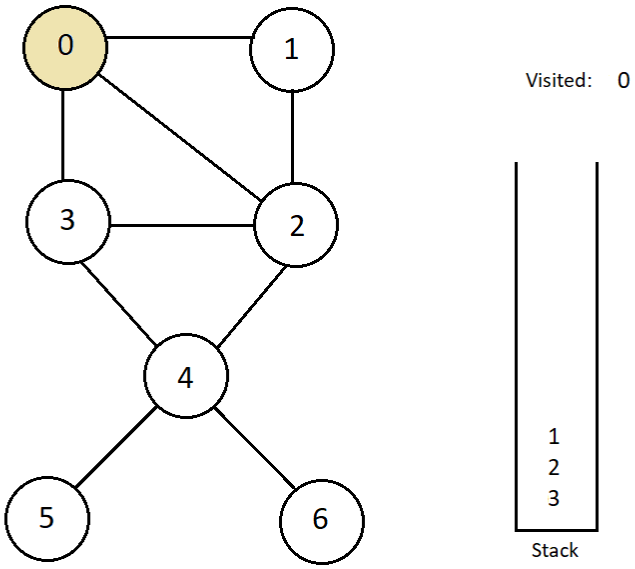


Considering the fact that we could begin traversing with any source node; we'll

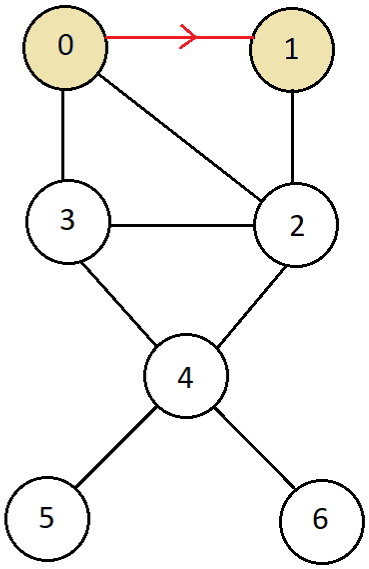
start with 0 only. So, following step1, we would push this node into the stack and begin our Depth First Search traversal.



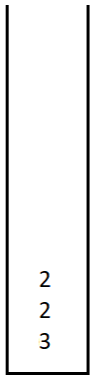
The next step says, pop the top element from the stack which is node 0 here, and mark it visited. Then, we'll start visiting all the nodes connected to node 0 which are not visited already, but before that, we are asked to push them all into the stack, and the order in which you push doesn't matter at all. Therefore, we will push nodes 3, 2, and 1 into the stack.



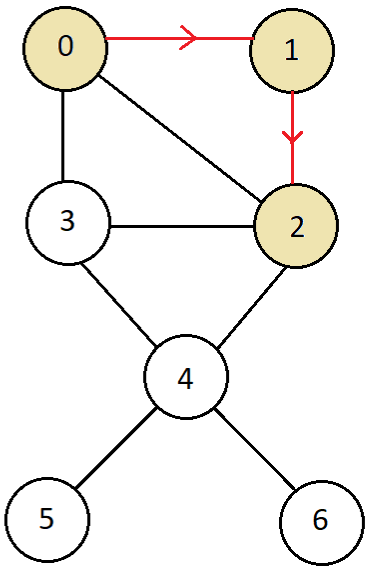
Repeating the steps above, we'll now pop the top element from the stack which is node 1, and mark it visited. Only nodes connected to node 1 were nodes 0 and 2, and since the only unvisited one is node 2. It's important to observe here, that although node 2 is in the stack, it is not visited. So, we'll push it into the stack again.



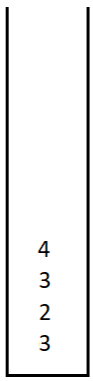
Visited: 0 1



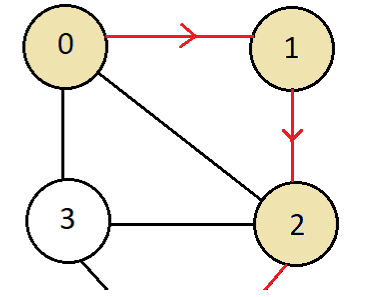
Next, we have node 2 at the top of the stack. We'll mark node 2 visited and unvisited nodes connected to node 2 are nodes 3 and 4, regardless of the fact that 3 is already there in the stack. So, we'll just push nodes 3 and 4 into the stack.



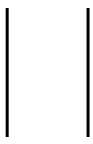
Visited: 0 1 2

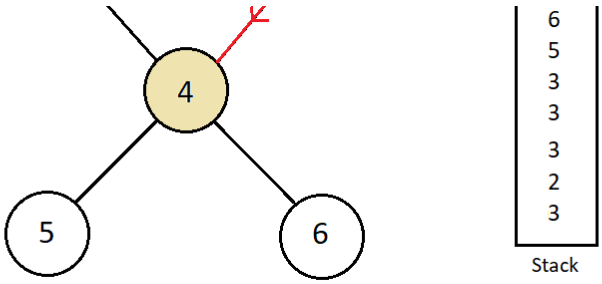


Node 4 is the next we have on the top. So, just mark it as visited. Since, all nodes 3, 5, and 6, except node 2, which are directly connected to it are not visited, we'll push them into the stack.

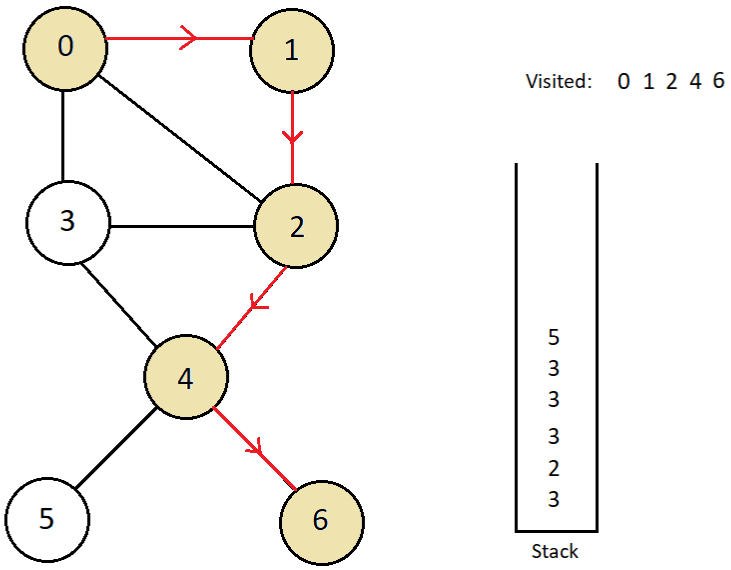


Visited: 0 1 2 4

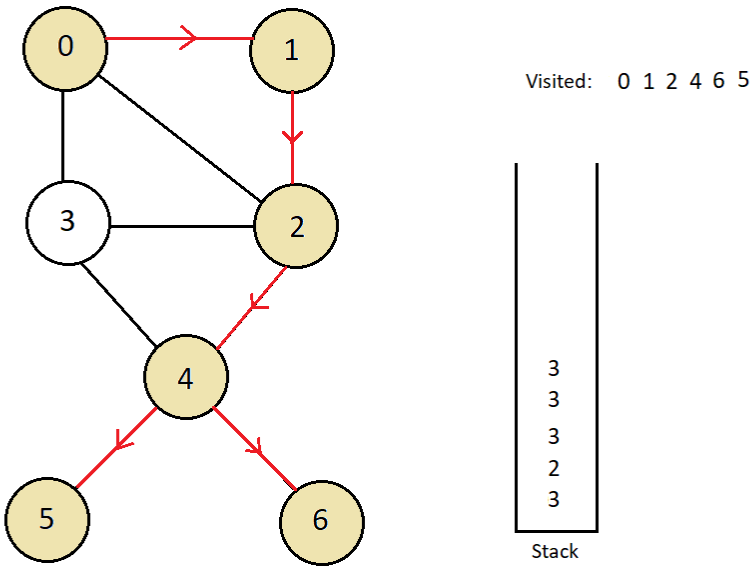




Next, we have node 6 on the top of the stack. Pop it and mark it visited. Since there are no nodes that are directed connected to node 6 and unvisited, we'll continue further without doing anything.

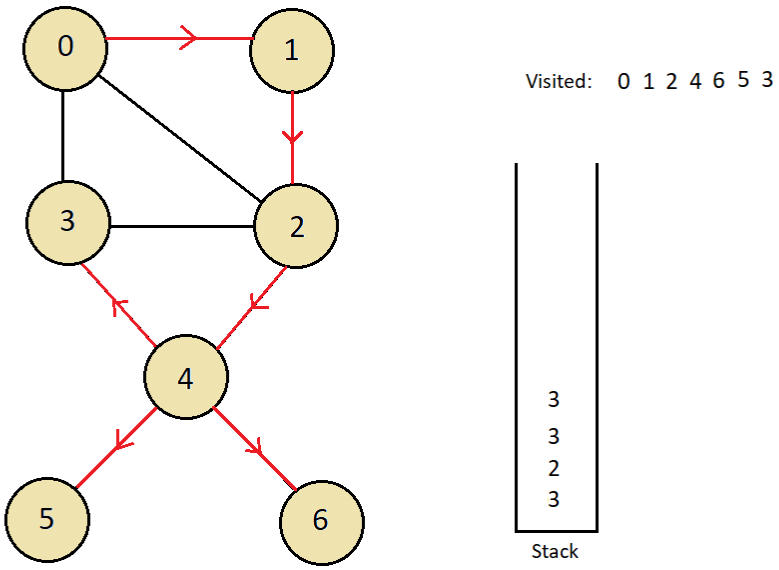


Next, we pop node 5 out of the stack and mark it visited. And since there is no unvisited node connected to it, we continue.

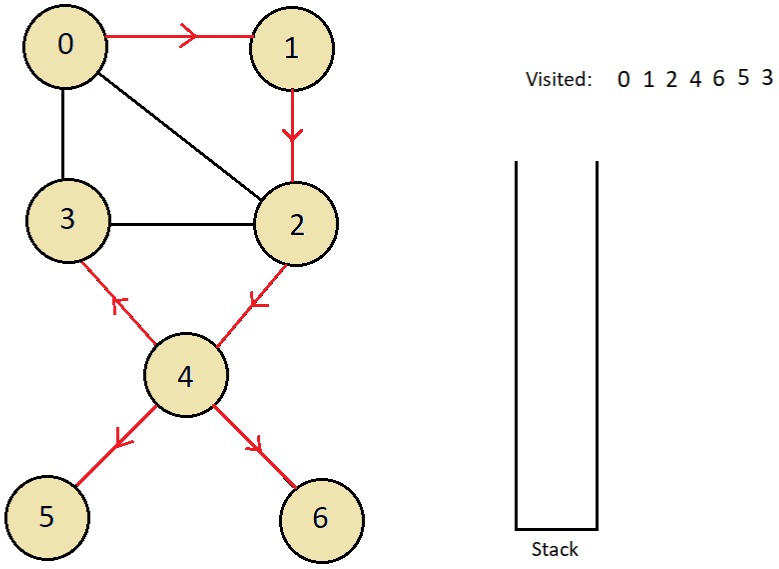


Node 3 comes next to be visited, being on the top in the stack. Mark node 3 visited

and again there are no nodes left unvisited and connected to node 3. So, we just continue popping out elements from the stack.



Now, if you could observe, there are no nodes left to be visited. Although there are elements in the stack to be explored. So, we just pop them one by one and ignore finding them already visited. And this gets our stack emptied and every node traversed in Depth First Search manner, ultimately.



And the order in which we marked our nodes visited is the Depth First Search traversal order. Here, it is **0, 1, 2, 4, 6, 5, 3.** So basically, the visited array maintains whether the node itself is visited or not, and the stack maintains nodes whose exploration got suspended earlier. This was the difference.

One important reason why stack is used in the implementation of DFS and what

makes it relatively easy is that it can directly be used in the form of functions since function calls use memory stacks. I will tell you how in the next lecture.
Let's now see how the process we did manually above can be automated in C although we will be using pseudocode for now. In the next lecture, we'll discuss it in more detail.

```
- Input: A graph G = (V,E) and source code s in V
- Algorithm:

  DFS(G,u)

  u.visited = true
  for each v ∈ G.adj(u)
     if v.visited == false
        DFS(G,v)


  init()
    for each u ∈ G
       u.visited = false
    for each u ∈ G
       DFS(G,u)
```

We are now just left with the programming part, which will be covered in the next segment. Keep practicing the concepts until then. Try writing the algorithm in C yourselves using the pseudocode I have provided above. We would anyway do that in the next lecture.

Thank you for being with me throughout the session. I hope you enjoyed it. If you appreciate my work, please let your friends know about this channel too. Lectures on graphs have just started, and if you haven't saved this already, you just have to move on to codewithharry.com or my YouTube channel to access them. See you all there in the next lecture where we'll see the implementation of the Depth First Search algorithm in C. Till then keep learning.

Previous Next

CodeWithHarry