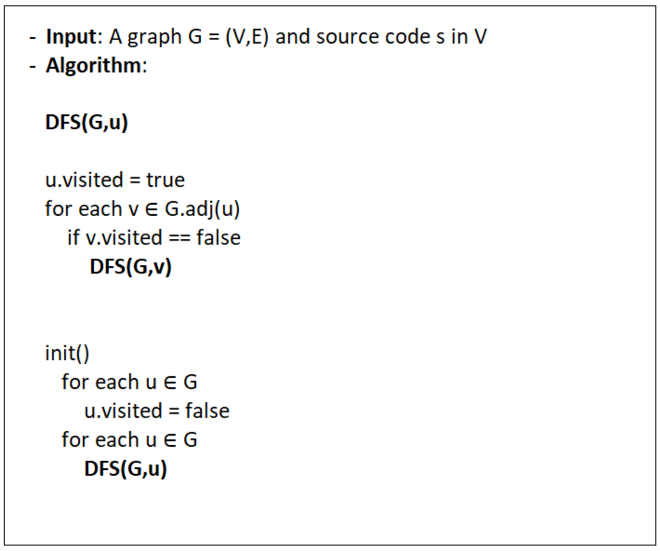
DFS Implementation in C | C Code For Depth First Search

codewithharry.com/videos/data-structures-and-algorithms-in-hindi-89

In the last lecture, we learned about the concepts of Depth First Search in graph traversal. We discussed the differences we observed in both the graph traversal methods. We discussed the algorithm to automate the DFS traversal of a graph. Today we will implement the same Depth First Search algorithm in C language.

Implementing Depth First Search, as I told you all in the last lecture, is one of the easiest jobs to do. DFS implementation asks you to use stacks, but if you know, a function call itself uses a memory stick, so calling a function recursively will do our work. And If you recall, we gave you all a pseudocode in the end and asked you all to read and try implementing the same in C. So, let me know if you could. The pseudocode was:



```
- 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)
```

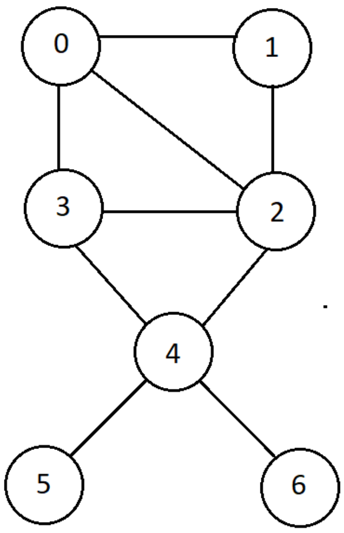
1. Now, the first thing we did was, we took the input. The input comprises the information concerning the graph, its nodes/vertices, and edges, and the source node we'll start the traversal with.
2. Then, we'll call the function DFS feeding a source node into it.
3. In the function, we'll mark the node visited, and then initiate a for loop which will access the connected nodes to this node, and see if they are visited or not.
4. And every time we find a node connected and not visited already, we call the DFS function recursively filling out our function stack. Eventually, the function stack becomes empty, and our traversal ends. And the visited array is the order of our Depth First Search traversal.

Well, this was the theory part. We're now ready to get started on the programming part. Let's move directly to our editors. I have attached the source code below. Follow it as we proceed.

Understanding the source code below:

1. Since we will be using functions to be able to use stacks directly, we don't really have to bring the implementation part of the stack from our previous lectures. Let’s jump to our DFS implementation directly.

2. Before we start programming the DFS implementation of the graph, we should first define a graph. And out of all the methods we have studied to represent a graph, we will be using the adjacency matrix one to define our graph here. And since we have already done this in our last programming lecture where we implemented the Breadth-First Search, we’ll bring the same graph here as well. This would save us a lot of time. So, basically, we got ourselves an adjacency matrix A[7][7] corresponding to the graph I’ve illustrated below.



3. Now, define an integer array *visited* to store the status of a node (of size 7 here), and the values corresponding to a node is 0 if it is unvisited and 1 if it is visited. We will fill this array with all zeros since no node has already been visited when we start.

Creating function DFS:

4. Now, we wish to define a function that would carry the Depth First Search traversal of the graph for us. So, create a void function *DFS* which will have the node *i* we want to visit as the only parameter. Inside that function, since that node which the function received as the parameter is the one to be visited now, we would mark it visited and print it at the same time.

5. Next step requires you to see all nodes connected directly to this node. Run a for loop of size 7 with an iterator *j*, since that is the number of nodes we have. If any of them is not visited and there is an edge in between *i* and *j*, we would recursively call DFS passing this node *j*. And that’s all. Trust your function and it will do things in its own way. Our job has finished.

```
void DFS(int i){
    printf("%d ", i);
    visited[i] = 1;
    for (int j = 0; j < 7; j++)
    {
        if(A[i][j]==1 && !visited[j]){
            DFS(j);
        }
    }
}
```

Code Snippet 1: Creating the DFS function

Here is the whole source code:

```
#include<stdio.h>
#include<stdlib.h>

int visited[7] = {0,0,0,0,0,0,0};
    int A [7][7] = {
        {0,1,1,1,0,0,0},
        {1,0,1,0,0,0,0},
        {1,1,0,1,1,0,0},
        {1,0,1,0,1,0,0},
        {0,0,1,1,0,1,1},
        {0,0,0,0,1,0,0},
        {0,0,0,0,1,0,0}
    };

void DFS(int i){
    printf("%d ", i);
    visited[i] = 1;
    for (int j = 0; j < 7; j++)
    {
        if(A[i][j]==1 && !visited[j]){
            DFS(j);
        }
    }
}

int main(){
    // DFS Implementation
    DFS(0);
    return 0;
}
```

Code Snippet 2: Implementing DFS using recursive function in C

We'll see if our algorithm actually works, and if it reverts the Depth First Search traversal order of the graph we fed into the program. If you could observe, we have started our traversal considering node 0 as the root node. And when we ran the program, the output we received was:

```
0 1 2 3 4 5 6
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 1: Output when the root node is 0

But if we change our root node from 0 to 4, the output changes to:

```
4 2 0 1 3 5 6
PS D:\MyData\Business\code playground\Ds & Algo with Notes\Code>
```

Figure 2: Output when the root node is 4

And this is how we implemented Depth First Search traversal in C without even explicitly using stacks just via a recursive function. You should all give it a shot too. No matter what node you start with, and whatever graph you feed the program with, it will work. Hopefully, that was clear. You can go through the lectures again for a better understanding of things. Stay tuned, as we are about to dive deep into graph theory.

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. Till then keep learning.