August 10, 2022 ▪ Data Structure / Graph

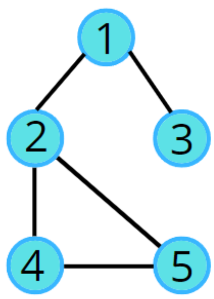
Depth First Search (DFS)

Problem Statement: Given an undirected graph, return a vector of all nodes by traversing the graph using depth-first search (DFS).

Pre-req: Recursion, Graph Representation

Examples:

Example 1:
Input:



Output: 1 2 4 5 3

Search

Search

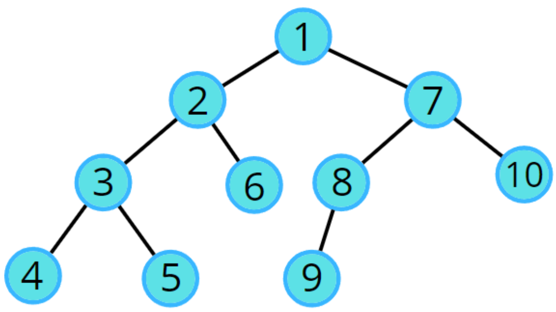
Recent Posts

Top Array Interview Questions – Structured Path with Video Solutions

Longest Subarray with sum K | [Postives and Negatives]

Count Subarray sum Equals K

```
Example 2:
Input:
```




```
Output: 1 2 3 4 5 6 7 8 9 10
```

Solution

***Disclaimer**: Don't jump directly to the solution, try it out yourself first.*

Approach:

DFS is a traversal technique which involves the idea of recursion and backtracking. DFS goes in-depth, i.e., traverses all nodes by going ahead, and when there are no further nodes to traverse in the current path, then it backtracks on the same path and traverses other unvisited nodes.

1. In DFS, we start with a node 'v', mark it as visited and store it in the solution vector. It is unexplored as its adjacent nodes are not visited.
2. We run through all the adjacent nodes, and call the recursive dfs function to explore the node 'v' which has not been visited previously. This leads to the

Binary Tree Representation in Java

Binary Tree Representation in C++

Accolite Digital Amazon Arcesium arrays Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing infosys inorder Interview Experience Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order recursion Samsung SDE Core Sheet SDE Sheet Searching set-bits sorting Strivers A2ZDSA Course sub-array subarray Swiggy takeuforward TCS TCS

exploration of another node 'u' which is its adjacent node and is not visited.

3. The adjacency list stores the list of neighbours for any node. Pick the neighbour list of node 'v' and run a for loop on the list of neighbours (say nodes 'u' and 'w' are in the list). We go in-depth with each node. When node 'u' is explored completely then it backtracks and explores node 'w'.
4. This traversal terminates when all the nodes are completely explored.

In this way, all the nodes are traversed in a depthwise manner.

CODEVITA TCS Ninja

TCS NQT

VMware XOR

Note: For a better understanding of the dry run please check the video listed below.

Code:

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
  private:
    void dfs(int node, vector<int> a
        vis[node] = 1;
        ls.push_back(node);
        // traverse all its neighbou
        for(auto it : adj[node]) {
            // if the neighbour is n
            if(!vis[it]) {
                dfs(it, adj, vis, ls
            }
        }
    }
  public:
    // Function to return a list con
    vector<int> dfsOfGraph(int V, ve
        int vis[V] = {0};
        int start = 0;
        // create a list to store df
```

```
        vector<int> ls;
        // call dfs for starting nod
        dfs(start, adj, vis, ls);
        return ls;
    }
};

void addEdge(vector <int> adj[], int
    adj[u].push_back(v);
    adj[v].push_back(u);
}

void printAns(vector <int> &ans) {
    for (int i = 0; i < ans.size();
        cout << ans[i] << " ";
    }
}

int main()
{
    vector <int> adj[5];

    addEdge(adj, 0, 2);
    addEdge(adj, 2, 4);
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 3);

    Solution obj;
    vector <int> ans = obj.dfsOfGrap
    printAns(ans);

    return 0;
}
```

Output: 0 2 4 1 3

Time Complexity: For an undirected graph, O(N) + O(2E), For a directed graph, O(N) + O(E), Because for every node we are calling the recursive function once, the time taken is O(N) and 2E is for total degrees as we traverse for all adjacent nodes.

Space Complexity: O(3N) ~ O(N), Space for dfs stack space, visited array and an adjacency list.

Java Code

```
import java.util.*;
class Solution {

    public static void dfs(int node,
    ArrayList<Integer> ls) {

        //marking current node as vi
        vis[node] = true;
        ls.add(node);

        //getting neighbour nodes
        for(Integer it: adj.get(node
            if(vis[it] == false) {
                dfs(it, vis, adj, ls
            }
        }
    }
    // Function to return a list con
    public ArrayList<Integer> dfsOfG
        //boolean array to keep trac
        boolean vis[] = new boolean[
        vis[0] = true;
        ArrayList<Integer> ls = new 
        dfs(0, vis, adj, ls);
        return ls;
    }

    public static void main(String a

        ArrayList < ArrayList < Inte
        for (int i = 0; i < 5; i++)
            adj.add(new ArrayList < 
        }
        adj.get(0).add(2);
        adj.get(2).add(0);
        adj.get(0).add(1);
        adj.get(1).add(0);
        adj.get(0).add(3);
        adj.get(3).add(0);
```

```
        adj.get(2).add(4);
        adj.get(4).add(2);

        Solution sl = new Solution()
        ArrayList < Integer > ans =
        int n = ans.size();
        for(int i = 0;i<n;i++) {
            System.out.print(ans.get
        }
    }
}
```

Output: 0 2 4 1 3

Time Complexity: For an undirected graph, O(N) + O(2E), For a directed graph, O(N) + O(E), Because for every node we are calling the recursive function once, the time taken is O(N) and 2E is for total degrees as we traverse for all adjacent nodes.

Space Complexity: O(3N) ~ O(N), Space for dfs stack space, visited array and an adjacency list.

> Special thanks to **Vanshika Singh Gour** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com

« Previous Post
Breadth First Search (BFS): Level Order Traversal

Next Post »
Python RegEx

Load Comments