August 7, 2022 ▪ Graph

Breadth First Search (BFS): Level Order Traversal

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

Pre-req: Graph Representation, Queue STL

Examples:

```
Example 1:
Input:
```

Search

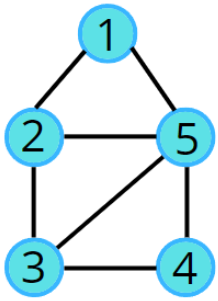
Search

Recent Posts

Top Array Interview Questions – Structured Path with Video Solutions

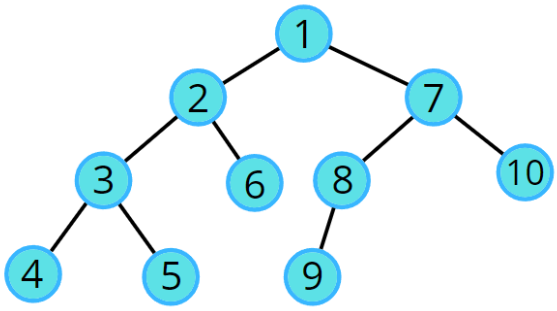
Longest Subarray with sum K | [Postives and Negatives]

Count Subarray sum Equals K



Output: 1 2 5 3 4

Example 2:
Input:



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

Solution

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

Approach:

Initial Configuration:

- Queue data structure: follows FIFO, and will always contain the starting.
- Visited array: an array initialized to 0

1. In BFS, we start with a "starting" node, mark it as visited, and push it into the queue data structure.

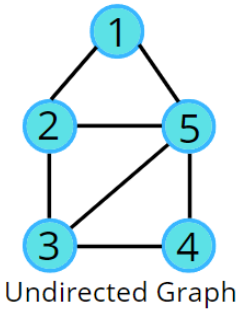
Binary Tree Representation in Java

Binary Tree Representation in C++

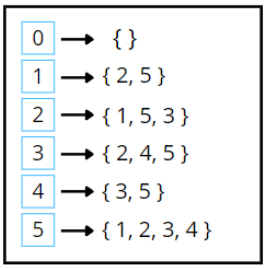
Accolite Digital Amazon Arcesium arrays Bank of America Barclays BFS Binary Search Binary Search Tree Commvault CPP DE Shaw DFS DSA Self Paced google HackerEarth Hashing infosys inorder Interview Experience Java Juspay Kreeti Technologies Morgan Stanley Newfold Digital Oracle post order recursion Samsung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

2. In every iteration, we pop out the node ‘v’ and put it in the solution vector, as we are traversing this node.
3. All the unvisited adjacent nodes from ‘v’ are visited next and are pushed into the queue. The list of adjacent neighbors of the node can be accessed from the adjacency list.
4. Repeat steps 2 and 3 until the queue becomes empty, and this way you can easily traverse all the nodes in the graph.

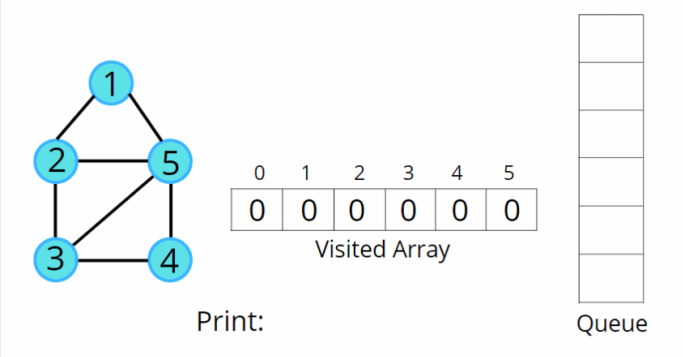
In this way, all the nodes are traversed in a breadthwise manner.



Undirected Graph



Adjacency List



Code:

C++ Code

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

```
class Solution {
  public:
    // Function to return Breadth Fi
    vector<int> bfsOfGraph(int V, ve
        int vis[V] = {0};
        vis[0] = 1;
        queue<int> q;
        // push the initial starting
        q.push(0);
        vector<int> bfs;
        // iterate till the queue is
        while(!q.empty()) {
           // get the topmost elemen
            int node = q.front();
            q.pop();
            bfs.push_back(node);
            // traverse for all its
            for(auto it : adj[node])
                // if the neighbour
                // store in Q and ma
                if(!vis[it]) {
                    vis[it] = 1;
                    q.push(it);
                }
            }
        }
        return bfs;
    }
};

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[6];

    addEdge(adj, 0, 1);
```

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

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

        return 0;
    }
```

Output: 0 1 4 2 3

Time Complexity: O(N) + O(2E), Where N = Nodes, 2E is for total degrees as we traverse all adjacent nodes.

Space Complexity: O(3N) ~ O(N), Space for queue data structure visited array and an adjacency list

Java Code

```
import java.util.*;
class Solution {
    // Function to return Breadth Fi
    public ArrayList<Integer> bfsOfG
    ArrayList<ArrayList<Integer>> ad

        ArrayList < Integer > bfs =
        boolean vis[] = new boolean[
        Queue < Integer > q = new Li

        q.add(0);
        vis[0] = true;

        while (!q.isEmpty()) {
            Integer node = q.poll();
            bfs.add(node);

            // Get all adjacent vert
            // If a adjacent has not
```

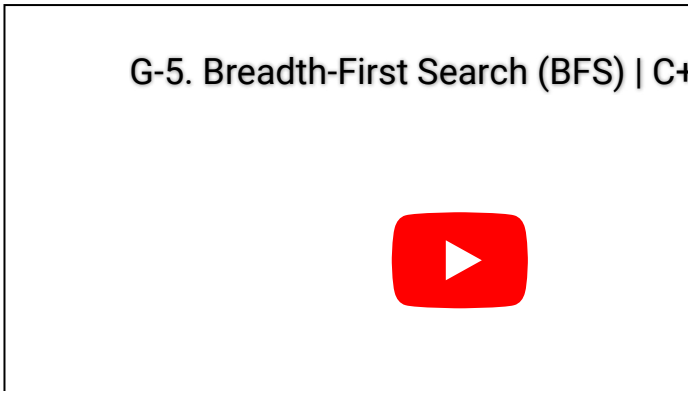
```
                // visited and enqueue i
                for (Integer it: adj.get
                    if (vis[it] == false
                        vis[it] = true;
                        q.add(it);
                    }
                }
            }

            return bfs;
        }

        public static void main(String a

            ArrayList < ArrayList < Inte
            for (int i = 0; i < 5; i++)
                adj.add(new ArrayList <
            }
            adj.get(0).add(1);
            adj.get(1).add(0);
            adj.get(0).add(4);
            adj.get(4).add(0);
            adj.get(1).add(2);
            adj.get(2).add(1);
            adj.get(1).add(3);
            adj.get(3).add(1);

            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 1 4 2 3

Time Complexity: O(N) + O(2E), Where N = Nodes, 2E is for total degrees as we traverse all adjacent nodes.

Space Complexity: O(3N) ~ O(N), Space for queue data structure 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
Connected Components in Graphs

Next Post »
Depth First Search (DFS)

Load Comments