Day 2

Done	☑
Topic	Bellman-Ford Algorithm And Implentation, Minimum Depth of a Binary Tree
Languages	Java
Difficulty	⭐⭐
Date Started	@March 2, 2023 11:00 PM
Date Completed	@March 3, 2023 3:42 AM
Related to Progress (Days)	Your Progress

What I Learned Today

What is Bellman-Ford Algorithm ? And It's Drawback.

Find Minimum Depth of a Binary Tree. By DFS(Bruteforce) and BFS(Optimised) Approach.

Key Concepts

Bellman-Ford Algorithm

- The Bellman-Ford algorithm is a dynamic programming algorithm used to find the shortest path between a single source vertex and all other vertices in a weighted directed graph. It can handle graphs with negative edge weights, which cannot be handled by Dijkstra's algorithm.
- The algorithm works by maintaining an array of distances from the source vertex to all other vertices and iteratively relaxing all the edges in the graph.
- It performs V-1 iterations, where V is the number of vertices in the graph, and checks for negative-weight cycles by performing one additional iteration.
- The algorithm has a time complexity of O(V * E), making it less efficient than Dijkstra's algorithm, but necessary for handling graphs with negative edge weights.

> Why bellman ford algorithm arrived is there is already Dijkstra's Algorithm?

Bellman Ford's Algorithm works when there is negative weight edge, it also detects the negative weight cycle. Dijkstra's Algorithm may or may not work when there is negative weight edge. But will definitely not work when there is a negative weight cycle.

DRAWBACKS

The drawbacks of the Bellman-Ford algorithm include its slow performance for large graphs O(V*E){For Complete Graph it extends to O(n^3)} , its inability to handle negative weight cycles, the possibility of non-unique shortest paths, and incomplete results when not all nodes are reachable from the source node.

Find Minimum Depth of a Binary Tree?

Quick Links

Bellman-Ford Tutorial

Bellman-Ford Algorithm(Documentation

Find Minimum Depth of a Binary Tree Tutorial

Find Minimum Depth of a Binary Tree(Documentation)

We have a binary tree, find its minimum depth. The Minimum Depth is the number of nodes along the shortest path from the root node down to the nearest leaf node. For example, minimum height of below Binary Tree is 2.

```
    1
   / \
  8   2
 / \  /
6  5 9
```

▼ **Method 1 : DFS**

The idea is to traverse the given Binary Tree. For every node, check if it is a leaf node. If yes, then return 1. If not leaf node then if the left subtree is NULL, then recur for the right subtree. And if the right subtree is NULL, then recur for the left subtree. If both left and right subtrees are not NULL, then take the minimum of two heights.

▼ **Method 2: BFS**

The above method may end up with complete traversal of Binary Tree even when the topmost leaf is close to root. A Better Solution is to do Level Order Traversal. While doing traversal, returns depth of the first encountered leaf node.

In this **Method-2**, we first check if the root is null, and return 0 if it is. We then initialize a queue with the root and a depth variable to 1.

We then enter a while loop that continues until the queue is empty. In each iteration of the loop, we retrieve the number of nodes in the current level of the tree, and iterate through each of those nodes. For each node, we check if it is a leaf node by checking if both its left and right child are null. If it is a leaf node, we return the current depth as the minimum depth of the tree.

If the node is not a leaf node, we add its left and right child to the back of the queue if they exist. We then increment the depth variable for the next level of the tree.

If the loop completes without finding a leaf node, we return the final depth as the minimum depth of the tree.

Code Snippets

```
import java.util.ArrayList;

public class BellmanFordAlgorithm {
    static class Edge {
        int src;
        int dest;
        int wt;

        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }
//    static void createGraph(ArrayList<Edge> graph[]){
//        for (int i=0; i< graph.length; i++){
//            graph[i]=new ArrayList<>();
//        }
//        graph[0].add(new Edge(0,1,2));
//        graph[0].add(new Edge(0,2,4));
//
//        graph[1].add(new Edge(1,2,-4));
//
//        graph[2].add(new Edge(2,3,2));
//
//        graph[3].add(new Edge(3,4,4));
//
//    }

    static void createGraph(ArrayList<Edge> graph) {
        graph.add(new Edge(0, 1, 2));
```

```
        graph.add(new Edge(0, 2, 4));
        graph.add(new Edge(1, 2, -4));
        graph.add(new Edge(2, 3, 2));
        graph.add(new Edge(3, 4, 4));

    }

    //    public static void bellmanFord(ArrayList<Edge> []graph, int src){
//        int[] dist = new int[graph.length];
//        for (int i=0; i< dist.length; i++){
//            if (i != src){
//                dist[i] = Integer.MAX_VALUE;
//            }
//        }
//        int V = graph.length;
//        //Algorithm
//        // O(V*E)
//        for (int i=0; i< V-1; i++){
//            //edge - O(E)
//            for (int j=0; j< graph.length; j++){
//                for (int k=0; k<graph[j].size(); k++){
//                    Edge e = graph[j].get(k);
//                    //u,v,wt
//                    int u = e.src;
//                    int v = e.dest;
//                    int wt = e.wt;
//                    if (dist[u]!=Integer.MAX_VALUE && dist[u]+wt<dist[v]) { // in java if something added in infinity value then it
//                        dist[v] = dist[u]+wt;
//                    }
//                }
//            }
//        }
//        //print
//        for (int i = 0; i<dist.length; i++){
//            System.out.print(dist[i]+" ");
//        }
//        System.out.println();
//    }
    public static void bellmanFord(ArrayList<Edge> graph, int src, int V) {
        int[] dist = new int[V];
        for (int i = 0; i < dist.length; i++) {
            if (i != src) {
                dist[i] = Integer.MAX_VALUE;
            }
        }
        //Algorithm
        // O(V*E)
        for (int i = 0; i < V - 1; i++) { //O(V)
            //edge - O(E)
            for (int j = 0; j < graph.size(); j++) {
                Edge e = graph.get(j);
                //u,v,wt
                int u = e.src;
                int v = e.dest;
                int wt = e.wt;
                if (dist[u] != Integer.MAX_VALUE && dist[u] + wt < dist[v]) { // in java if something added in infinity value then it
                    dist[v] = dist[u] + wt;
                }
            }
        }

        //print
        for (int i = 0; i < dist.length; i++) {
            System.out.print(dist[i] + " ");
        }
        System.out.println();
    }

    public static void main(String[] args) {
        int V = 5;
//        ArrayList<Edge> []graph = new ArrayList[V];
        ArrayList<Edge> graph = new ArrayList<>();
        createGraph(graph);
        bellmanFord(graph, 0, V);
    }
}
```

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;

public class GraphPractice {
    static class Node {
```

```
        int data;
        Node left;
        Node right;

        Node(int data) {
            this.data = data;
            this.left = null;
            this.right = null;
        }
    }
//    public static int minimumDepth(Node root){ // this is going to traverse whole binary tree
//        if (root == null){
//            return 0;
//        }
//        // If the root has no children, the minimum depth is 1
//        if (root.left == null && root.right == null){
//            return 1;
//        }
//        // If the root has only one child, return the minimum depth of the child subtree
//        if (root.left==null){
//            return minimumDepth(root.right)+1 ; // Depth of right subtree from root
//
//        }
//        if (root.right==null){
//            return minimumDepth(root.left)+1; // Depth of left subtree from root
//
//        }
//        // If the root has two children, return the minimum depth of the two subtrees
//        return Math.min(minimumDepth(root.left),minimumDepth(root.right)) + 1; // returned the minimum of left subtree depth and rig
//
//        // we +1 minimum of left and right subtree depth. so it gives the depth of itself to parent node by adding 1 in minimum of l
//    }
public static int minimumDepth(Node root) { // this is going to traverse level order and trigger level wise if the node has no child
        int minDepth = 0;
        if (root == null) {
            return minDepth;
        }
        Queue<Node> q = new LinkedList<>();
        q.add(root);
        while (!q.isEmpty()) {
            minDepth++;
            int levelSize = q.size();
            for (int i = 0; i < levelSize; i++) {
                Node currNode = q.remove();
                if (currNode.left == null && currNode.right==null){
                    return minDepth;
                }
                if (currNode.right!=null){
                    q.add(currNode.right);
                }
                if (currNode.left!=null){
                    q.add(currNode.left);
                }
            }
        }
        return minDepth;
    }
public static void main(String[] args) {
/*
                    1
                  /  \
                8     2
              / \    /
             6    5 9
*/
        Node root = new Node(1);
        root.left = new Node(8);
        root.right = new Node(2);
        root.left.left = new Node(6);
        root.left.right = new Node(5);
        root.right.left = new Node(9);
        System.out.println(minimumDepth(root) - 1);
    }
}
```

Challenges Experienced

Faced issues in corner cases of root node’s child.

Resources Used

Youtube, ChatGPT, Alpha, GeeksForGeeks