**SD LAB 6: Graphs, trees & Containers.**



Student: Johnson Domacasse

Date: 27 Nov 2023

Student#: 4471709

Teacher: Lake Lakeman

Contents

[Introduction: 3](#_Toc153649291)

[Binary trees: 3](#_Toc153649292)

[Graphs and Containers: 4](#_Toc153649293)

[References: 5](#_Toc153649294)

# Introduction:

*The purpose of this document is to show or justify the decisions that were made during this assignment. Below you will find the sections that talk about trees and graphs. Inside these sections there will be the design choices I made, along with a reflection on these design choices.*

# Binary trees:

For this implementation, We focus on two things. The algorithm to find the depth of the binary tree**[1]**, and the way we create the tree, based on the input files. For this implementation.

For the algorithm**,** I have decided to use the Depth-First search algorithm (DFS) over the breadth-first search algorithm (BFS) for the following reasons**[3]**:

* **“DFS is more suitable when there are solutions away from the source”:** In this case we are trying to find the depth of a binary tree, we don’t know whether the depth is one level lower or hundreds. If BFS were to be used, then it would take longer since it would need to visit all of the siblings before it can reach the bottom leaf nodes.
* **“DFS requires less memory than BFS”:** Memory isn’t a big problem in this case but the less we use the better I would say
* **“when the target is far from the source, DFS is preferred”:** Again, BFS could’ve been a better option in this case if the depth level was fairly low. However since we do not know if the depth is a level lower, or hundreds, we use DFS.

With that being said a DFS algorithm function was implemented**[4].**

For reading the input file, a different approach was taken. Reading the user input was easy, we would also need to store this information somehow. The following drawing shows the thought process:

A piece of paper with writing on it

Description automatically generated

From here, the tree was drawn, where it was realized that the nodes needed to be saved in some type of container. From there the options were weighed. This was implemented using vectors because of their ability to dynamically resize themselves during run time. Initially the approach was using an array because the size of the tree is already given. This was to get some practice in with vectors. This implementation gives us the option to have our storage grow dynamically in an efficient way at the cost of memory**[5].**

# Graphs and Containers:

For this implementation, similar to the tree, we have to build the graph by connection each node with another node based on the input file that was given.

For this implementation, I will be using the BFS algorithm and I am going to implement this similarly to what can be found in the geeksforgeeks implementation**[6]**. Instead of creating the graph with a function like before, this time we will create it using a class.

The class holds members such as a Node struct to hold the information of each node in the graph. In this case the number of the node and the connections between nodes. The class itself with hold a list of those nodes and the amount of nodes. Finally we finish of by creating a constructor for the class, a function that can create connections between nodes, the BFS algorithm function.

We then create **bidirectional** connections between each node. Since they are bidirectional you need to have a connection not only between node 1 and node 2 but also node 2 and node 1.

Similar to the class, the algorithm function was also made using inspiration from our reference**[6].** In this implementation we keep track of a list of the visited nodes and also the shortest route. We create a q to traverse through the graph as long as the Q is not empty. In here we check if the neighbour of the current node has **not** been visited. If this is the case, we update the shortest route to this neighbour. We push this to keep the loop going. In the end we return the shortest path to node 1 to node N.

From there in the main, we simply read the amount of test cases, and for each test case we create a graph(object), pass the amount of nodes into the graph. From there, based on the amount of connections that are to be made (M), we make connections and then we use the algorithm to find the route.

A piece of paper with writing on it

Description automatically generated

Using the above figure for example, the arrow points at node 3. Assuming you came from node 1, the you would know that node 2 is already visited (doesn’t go through the loop). Node 4 is not visited, so we add that into the visited list as true. As soon as we get to node 4 in the q (since its pushed as neighbour) we add in into the visited list. The same goes for node 6. However with node 6 (end node) we have to take the shortest path of our current node (which is 3 at node 3) and increment it. so in this case the shortest path would be from 1 - 2 - 3 - 6. By the end when all the neighbours were visited then we return the shortest path to 6. Which is 3.

# References:

*This section is to give all of the references that I have used in the making of this assignment.*

**[1]** – GeeksforGeeks. (2023e, September 27). *Binary tree data structure*. GeeksforGeeks. <https://www.geeksforgeeks.org/binary-tree-data-structure/>

**[2]** – MyCodeSchool. (2014, January 18). *Data Structures: Binary Tree*. MyCodeSchool. <https://www.youtube.com/watch?v=H5JubkIy_p8&t=838s&pp=ygULYmluYXJ5IHRyZWU%3D>

**[3]** – GeeksforGeeks. (2023, October 3). *DFS & BFS Algorithms*. GeeksforGeeks. <https://www.geeksforgeeks.org/find-the-maximum-depth-or-height-of-a-tree/>

**[4]** - GeeksforGeeks. (2023b, October 3). *DFS in binary tree*. GeeksforGeeks. <https://www.geeksforgeeks.org/find-the-maximum-depth-or-height-of-a-tree/?ref=lbp>

**[5]** - *Vectors*. cplusplus.com. (n.d.). <https://cplusplus.com/reference/vector/vector/>

**[6] -** GeeksforGeeks. (2023a, June 9). *Breadth first search or BFS for a graph*. GeeksforGeeks. <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>