GTU Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework 5 Report

Mehmet Mete Şamlıoğlu
200104004093

1-) SYSTEM REQUIREMENTS

The system was requiring a tree that reads its content of it from the tree.txt file and builds a JTree to visualize. A node of a tree can have more than one child so that is not a binary tree, this was hard to solve because all the implementations in the book were based on a binary tree. Therefore it was something new to implement. The other difficult requirement was moving a node in Tree to somewhere else. This requires inserting and removing from the Tree and also another search algorithm was needed to perform this.

2-) PROBLEM-SOLUTION APPROACH

I will examine all the functions that I used to solve the problems in this homework one by one and explain in here.

public void parseFileString(String filename)

This function has provoked the constructor to parse "the tree.txt" accordingly, it parses text line by line and put the String array into a 2D string array (which is created with ArrayList<String[]>) dynamically. After the parsing operation has been done, it provokes the createTree() method to build a tree.

public void createTree(ArrayList<String[]>NodeList)

It takes a 2D string Array as a parameter and builds a tree by using the strings that have been parsed. In each iteration it creates a child for the following nodes in the string but before it adds the child it controls whether the child already exists in the parentNode to avoid the duplicate child. It performs that duplicate search with the help of the method findNode(). The findNode() method takes two arguments the first one is the root of the tree and the second one is a node that is being searched, it basically checks if the duplicate exist in given node, if duplicate exist it returns -1, otherwise it returns the index of the searched node.

public void updateTree()

After editing the tree (move operations) , the tree should be updated by calling this function.

public void BFSAlgorithm(String node)

This is one of the search algorithms that has been asked in the PDF. Since I did not find the algorithm or an example of it in the book I searched for it on the Internet and examine the algorithm of it. The BFS algorithm basically begins at the root node and then visits all nodes level by level. It means that after the root, it traverses all the direct children of the root. To implement it, it was suggested to use a queue because of the working mechanism of the queue(FIFO). So I created a queue and push the root of the tree in it at the beginning. After that, I created a while loop that checks if the queue is empty or not and this was my end condition for the iteration. In each iteration First I checked if the node is found or not, if it is not found I put the children of the current node into the queue to perform the operation that I mentioned in the beginning. The first child will be controlled last because of the mechanism of the queue(FIFO), so in each iteration, I put the child of the current node into the queue and poll them one by one and check whether it matches with the node that I'm searching or not. I printed each step and also printed the found node in each step.

public void DFSAlgorithm(String node)

DFS was somehow the opposite of the BFS so it was really easy to implement it after BFS. The DFS basicaly begins at the root node and then it explores each branch before backtracking. Since it was the opposite mechanism of BFS It is implemented using stack. Both algorithm can also be solved with recursive functions but solving them with queue and stack was easier compared to recursive functions. So to implement DFS search I used exactly the same algorithm with BFS the only difference was the priority of the childnodes'search so I used stack instead of queue to perform. In each iteration I pushed the nodes of the current node to the stack instead queue and, after that I compared pop them out one by one in each iteration and compared them with the node that I'm searching. So I said it was really easy to implement after the BFS.

public ArrayList<DefaultMutableTreeNode> searchNodePath(ArrayList<String> source)

Since it was really hard to find the path with the searching algorithms above(BFS and DFS) I had to find another algorithm to find the path of the node. Getting the node alone was not enough to perform the move operation because the path of the node that I'm searching could be matched with the node that I'm moving. In these cases, it requires overwriting instead of inserting the node. To check if the move operation will be performed by inserting or overriding I had to find the path. So I parsed the moveFrom path one by one and found the related nodes that matched with the moveFrom path, and after that, I put them into ArrayList one by one. In the end, I obtained all the required nodes to get to the given path.

public void moveNode(String moveFrom, String moveTo)

This was the main function that I used to move a node to some destination, first I found the path of the node by calling the searchNodePath method that I explained above. After that, I checked how many nodes were in this path to continue the operation. If there is only one node in the path I checked If there is a duplicate or not manually and do the required operation after that. If there is more than one node in the math I provoked the moveToPath method and pass the past and the destination point of the node in it after that it performed the desired operation. In the end, I called the update tree method to recreate the tree and display the edited version of it. The only error condition that I checked in this method was if the given path existed or not. All error conditions other than that have been checked in the moveToPath method.

public void moveToPath(ArrayList<DefaultMutableTreeNode> path, String destination)

I used this method to move the nodes whose path has more than one node. In other words, for the nodes whose depth is more than 1. This condition required another function because in some cases it might require overwriting. So the very first thing that I have done in this function is parse the string according to the "->" substring, after that, I checked if the given path exists or not by iterating the root node. After that, I performed two different operations according to the result of the isPathExist boolean value. If the given path exists, I control the every node to check if

it matches with any node in the destination point. If it matches with any node in the destination I overwrite the node in that location. If no node matches with any node in the destination point I just add the node without overwriting it. If destination year does not exist, I created one and add path as a children of it.

3-) Running Results

BFS Searching CSE232

```
1-) BFS Search
2-) DFS Search
3-) Move a node
4-) Exit
Choose the operation: 1
Enter the node:CSE232
Using BFS to find CSE232 in the tree...
Step 1 -> Root
Step 2 -> 2021
Step 3 -> 2022
Step 4 -> 2023
Step 5 -> CSE102
Step 6 -> CSE321
Step 7 -> CSE222
Step 8 -> CSE232(Found!)
```

BFSE Searching CSE2332

```
meterose@DESKTOP-3HDDHUD:/mnt/c/hw5$ java Driver
1-) BFS Search
2-) DFS Search
3-) Move a node
4-) Exit
Choose the operation: 1
Enter the node:CSE2332
Using BFS to find CSE2332 in the tree...
Step 1 -> Root
Step 2 -> 2021
Step 3 -> 2022
Step 4 -> 2023
Step 5 -> CSE102
Step 6 -> CSE321
Step 7 -> CSE222
Step 8 -> CSE232
Step 9 -> LECTURE1
Step 10 -> LECTURE1
Step 11 -> LECTURE2
Step 12 -> LECTURE1
Step 13 -> LECTURE1
Step 14 -> LECTURE2
Step 15 -> LECTURE3
Step 16 -> PROBLEM1
Step 17 -> PROBLEM2
Step 18 -> PROBLEM1
Step 19 -> PROBLEM2
Not found.
```

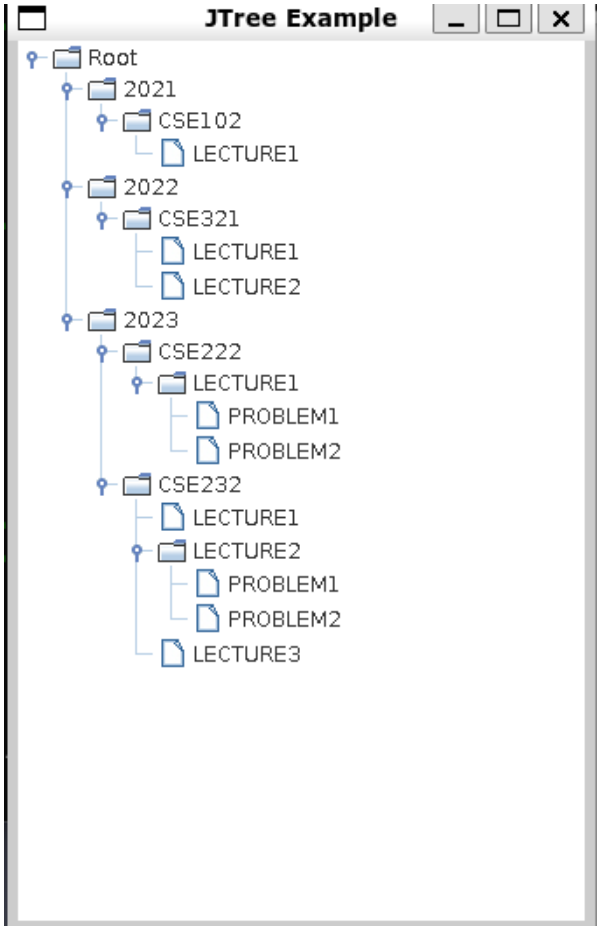
DFS Searching CSE232

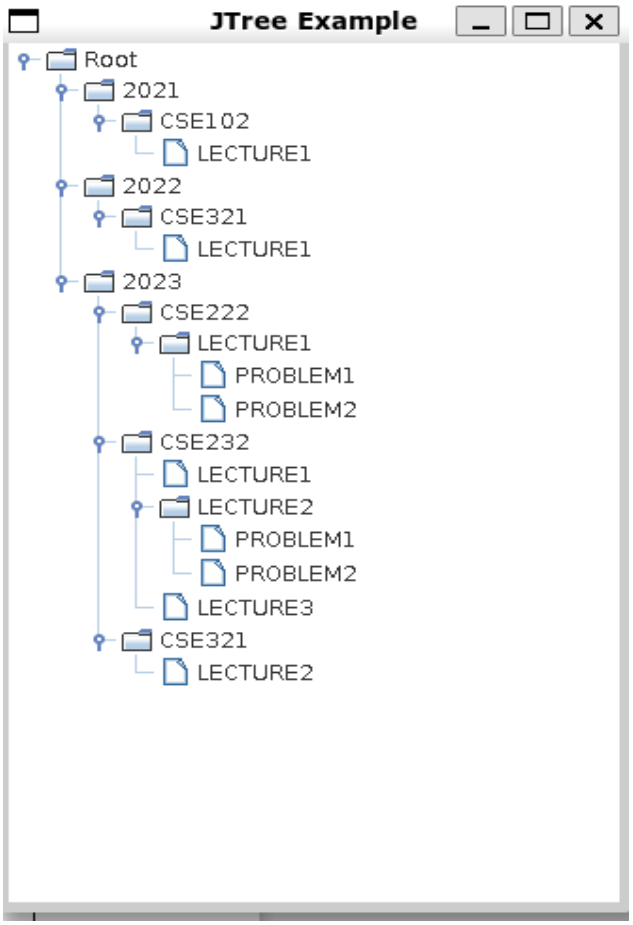
```
meterose@DESKTOP-3HDDHUD:/mnt/c/hw5$ java Driver
1-) BFS Search
2-) DFS Search
3-) Move a node
4-) Exit
Choose the operation: 2
Enter the node:CSE232
Step 1 -> Root
Step 2 -> 2023
Step 3 -> CSE232(Found!)
```

DFS Searching CSE2332

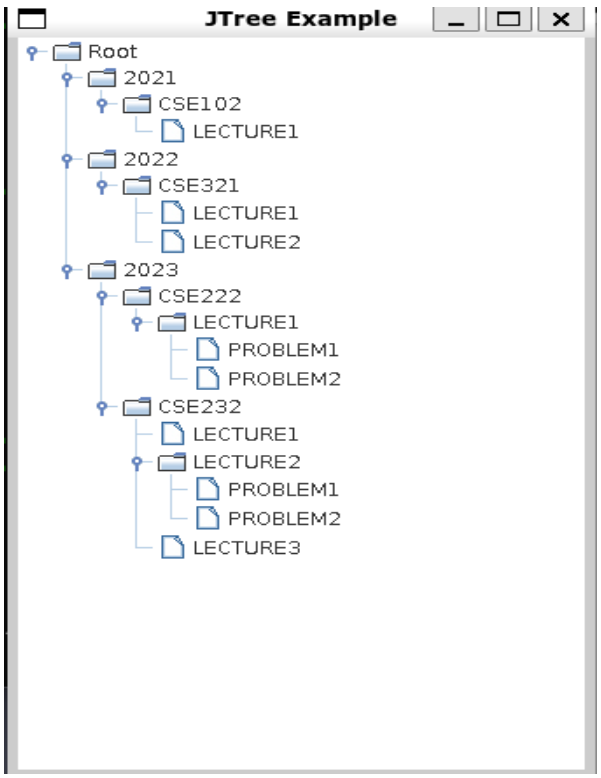
```
1-) BFS Search
2-) DFS Search
3-) Move a node
4-) Exit
Choose the operation: 2
Enter the node:CSE2332
Step 1 -> Root
Step 2 -> 2023
Step 3 -> CSE232
Step 4 -> LECTURE3
Step 5 -> LECTURE2
Step 6 -> PROBLEM2
Step 7 -> PROBLEM1
Step 8 -> LECTURE1
Step 9 -> CSE222
Step 10 -> LECTURE1
Step 11 -> PROBLEM2
Step 12 -> PROBLEM1
Step 13 -> 2022
Step 14 -> CSE321
Step 15 -> LECTURE2
Step 16 -> LECTURE1
Step 17 -> 2021
Step 18 -> CSE102
Step 19 -> LECTURE1
Not found.1-) BFS Search
```

The Original Tree (The tree from part A)	The Edited Tree (Moved **2022->CSE321->LECTURE2** to **2023**)





The Original Tree (The tree from part A)	The Edited Tree (Moved **2022->CSE321** to **2020**)




```
Example Input; Move from:2023->CSE222
Move from: 2022->CSE222
Move to: 2020
Cannot move 2022->CSE222 because it does not exist in the tree.
```

The Original Tree (The tree from part A)	The Edited Tree (Moved **2023->CSE232->LECTURE2->PROBLEM2** to **2022**)