GEBZE TECHNICAL UNIVERSITY

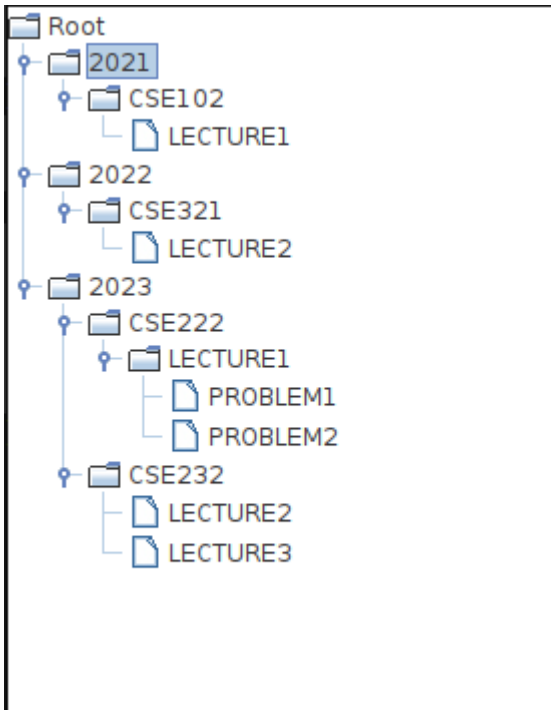
DATA STRUCTURE

HOMEWORK 5 REPORT

225008003102

SÜHA BERK KUKUK

İ) Create Tree



- private String FILENAME = "src/data.txt";: This line initializes a String variable FILENAME to the file path of the data file that the program will read data from.

- private static final String DELIMITER = ";";: This line initializes a static final String variable DELIMITER to the delimiter used in the data file to separate nodes.

- private JTree jt;: This line declares a JTree object jt that will be used to represent the tree.

- private JFrame f;: This line declares a JFrame object f that will be used to contain the JTree.

- public Tree(): This is the constructor method for the Tree class.

- ArrayList<String> data = new ArrayList<>();: This line creates an ArrayList object data to store the data read from the file.

- try {...} catch (FileNotFoundException e) {...}: This code block attempts to read data from the file specified by FILENAME using a Scanner object. If the file is not found, an exception is caught, and a stack trace is printed to the console.

- while (myScanner.hasNextLine()) {...}: This while loop reads each line of the file and adds it to the ArrayList data.

- Map<String, DefaultMutableTreeNode> information = new HashMap<>();: This line creates a Map object information to store the tree nodes.

- DefaultMutableTreeNode root = new DefaultMutableTreeNode("Root");: This line creates a DefaultMutableTreeNode object root that will be the root of the tree.

- for (String line : data) {...}: This for loop iterates through each line of the ArrayList data.

- String[] nodes = line.split(DELIMITER);: This line splits each line of the file into an array of strings using the

ii) Breath First Search

```
Using BFS to find'LECTURE2' 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 LECTURE2
Step ->11 LECTURE1
Step ->12 LECTURE2(found)
```

```
Using BFS to find'NOT' 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 LECTURE2
Step ->11 LECTURE1
Step ->12 LECTURE2
Step ->13 LECTURE3
Step ->14 PROBLEM1
Step ->15 PROBLEM2
Node not found!
```

- System.out.println("Using BFS to find"+""" +search+ """+ " in the Tree."); - This line just prints a message indicating that BFS is being used to find a node with a specific name in the tree.

- int step = 1; - This initializes a counter variable step to keep track of each step taken by BFS in searching for the node.

- Queue<DefaultMutableTreeNode> queue = new LinkedList<>(); - This creates a Queue object called queue of type DefaultMutableTreeNode (which is a node in the tree), using the LinkedList implementation.

- queue.add((DefaultMutableTreeNode)jt.getModel().getRoot()); - This adds the root node of the tree to the queue for processing. The getRoot() method returns the root node of the JTree object, which is then cast to DefaultMutableTreeNode to match the type of queue.

- while (!queue.isEmpty()) { - This initiates a loop that runs as long as there are still nodes to be processed in the queue.

- DefaultMutableTreeNode node = queue.poll(); - This removes and returns the head of the queue (i.e., the first node that was added to the queue) and assigns it to the variable node.

- if (node.toString().equals(search)) { - This checks if the name of the current node (node.toString()) is equal to the name of the node being searched for (search).

- System.out.println("Step ->"+step+" "+node.toString()+"(found)"); - If the name of the node matches the search criteria, this line prints a message indicating the step number, the name of the node, and that it has been found.

- return; - This immediately exits the method if the node is found.

- else - If the node name does not match the search criteria, the code inside the else block is executed.

- System.out.println("Step ->"+step+" "+node.toString()); - This prints a message indicating the step number and the name of the node being processed.

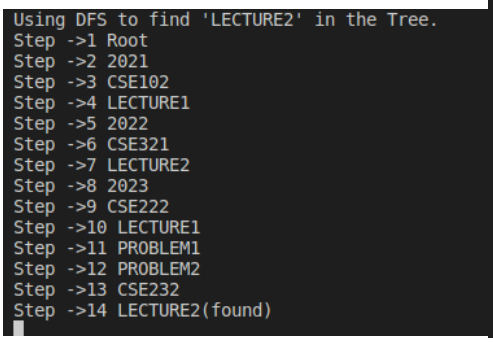
- for (int i = 0; i < node.getChildCount(); i++) { - This loops through all the children nodes of the current node.

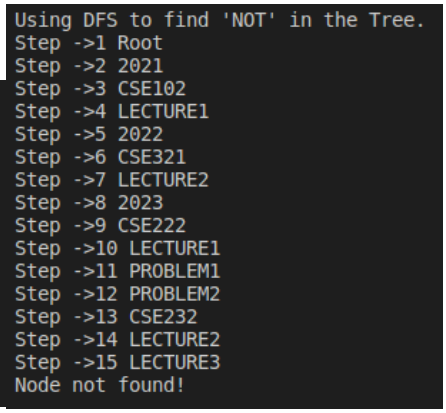
- queue.add((DefaultMutableTreeNode)node.getChildAt(i)); - This adds each child node to the end of the queue for later processing. The getChildAt() method returns the child node at the specified index, which is then cast to DefaultMutableTreeNode to match the type of queue.

- step++; - This increments the step counter to track the next step in the search process.

- System.out.println("Node not found!"); - If the method completes without finding the node, this line is printed to indicate that the node was not found in the tree.

iii) Depth First Search





- The method dfs is declared, which takes a search parameter representing the value to search for in the tree.

- A print statement is executed to indicate that the Depth First Search (DFS) algorithm is being used to search for the given value in the tree.

- A variable step is declared and initialized to 1 to keep track of the current step in the search.

- A new Stack object is created to hold tree nodes during the DFS search.

- The root node of the tree is pushed onto the stack to start the DFS search.

- A while loop is initiated that runs until the stack is empty.

- Inside the while loop, the top node from the stack is popped off and assigned to the node variable.

- An if statement is executed to check if the value of node equals the search value. If it does, a message is printed indicating that the value was found, and the method returns.

- If the value of node does not equal the search value, then a message is printed indicating that the current node is being searched.

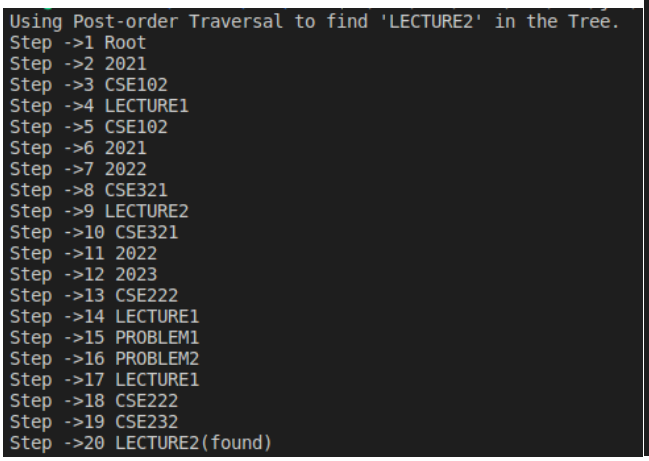
- A for loop is initiated to iterate through all the children of the current node in reverse order, from the last child to the first child.

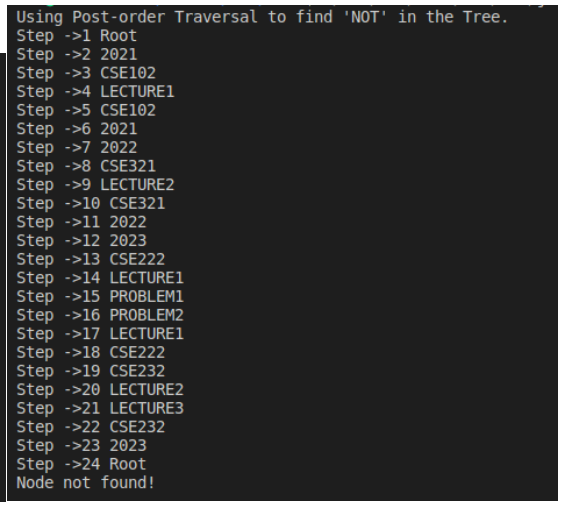
- Inside the for loop, each child node is pushed onto the stack.

- The step variable is incremented.

- If the while loop finishes executing without finding the search value, a message is printed indicating that the search value was not found in the tree.

iv) Post Order Traversal





- The method is defined with a single parameter, search, which is the value of the node being searched for in the tree.

- A message is printed indicating that the post-order traversal algorithm is being used to search for the node.

- A Stack object is created to hold the tree nodes during the traversal.

- The root node of the tree is obtained using the getModel() method of the JTree object and is cast to a DefaultMutableTreeNode object.

- A lastVisitedNode variable is initialized to null, which will be used to keep track of the last node that was visited during the traversal.

- A step variable is initialized to 1, which will be used to keep track of the current step during the traversal.

- The root node is pushed onto the stack to start the traversal.

- While the stack is not empty, the loop continues.

- The top node on the stack is obtained using the peek() method.

- If the value of the current node is equal to the search parameter, then a message is printed indicating that the node has been found and the method returns.

- If the value of the current node is not equal to the search parameter, then a message is printed indicating that the current node is being visited.

- If the current node is a leaf node or if the last visited node is the last child of the current node, then the current node is popped off the stack and the last visited node is set to the popped node.

- If the current node is not a leaf node and the last visited node is not the last child of the current node, then a loop is used to push the current node's children onto the stack in reverse order.

- The step variable is incremented.

- If the search node is not found, a message is printed indicating that the node was not found.

V) Move to Node

I could not implemented.