GEBZE TECHNICAL UNIVERSITY COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

CSE 222 / 2023 SPRING

HOMEWORK 5 REPORT

ETHEM MENGAASLAN

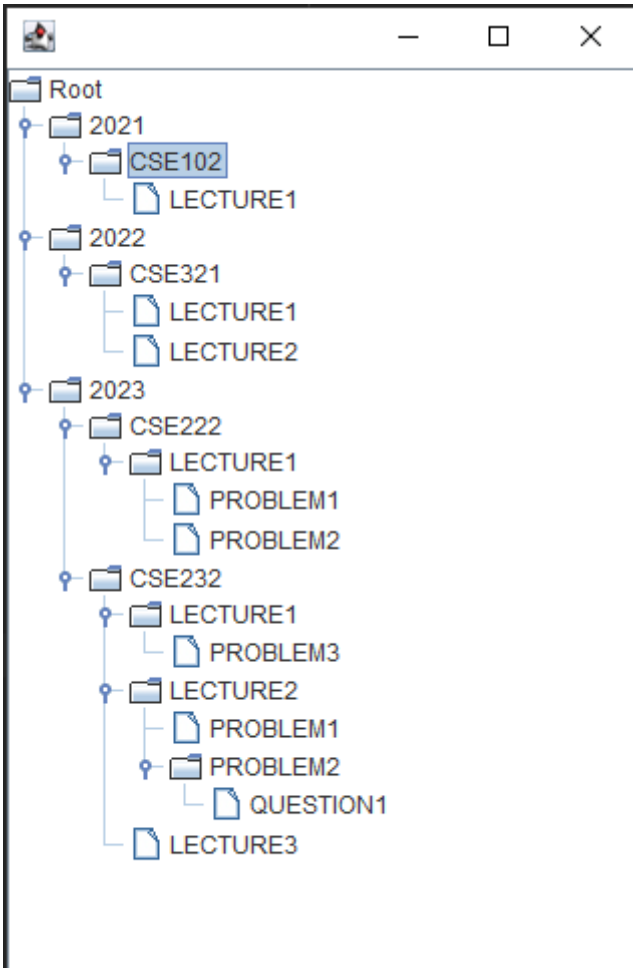
200104004016

TEST & OUTPUT:

tree.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

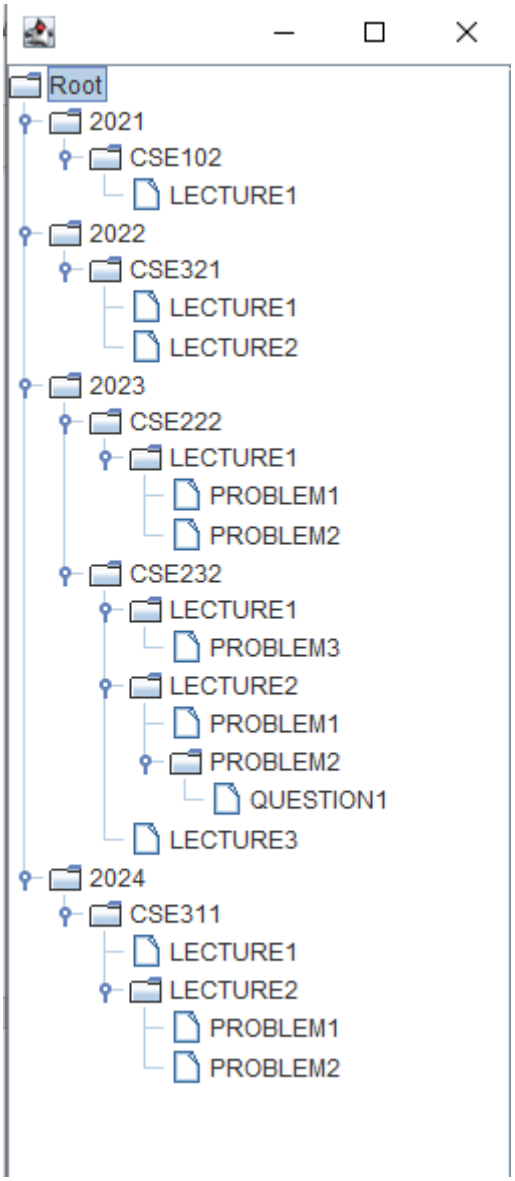
```
2021;CSE102;LECTURE1
2022;CSE321;LECTURE1;LECTURE2
2022;CSE321;LECTURE2
2023;CSE222;LECTURE1;PROBLEM1
2023;CSE222;LECTURE1;PROBLEM2
2023;CSE232;LECTURE1;PROBLEM3
2023;CSE232;LECTURE2;PROBLEM1
2023;CSE232;LECTURE2;PROBLEM2;QUESTION1
2023;CSE232;LECTURE3
```



tree.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

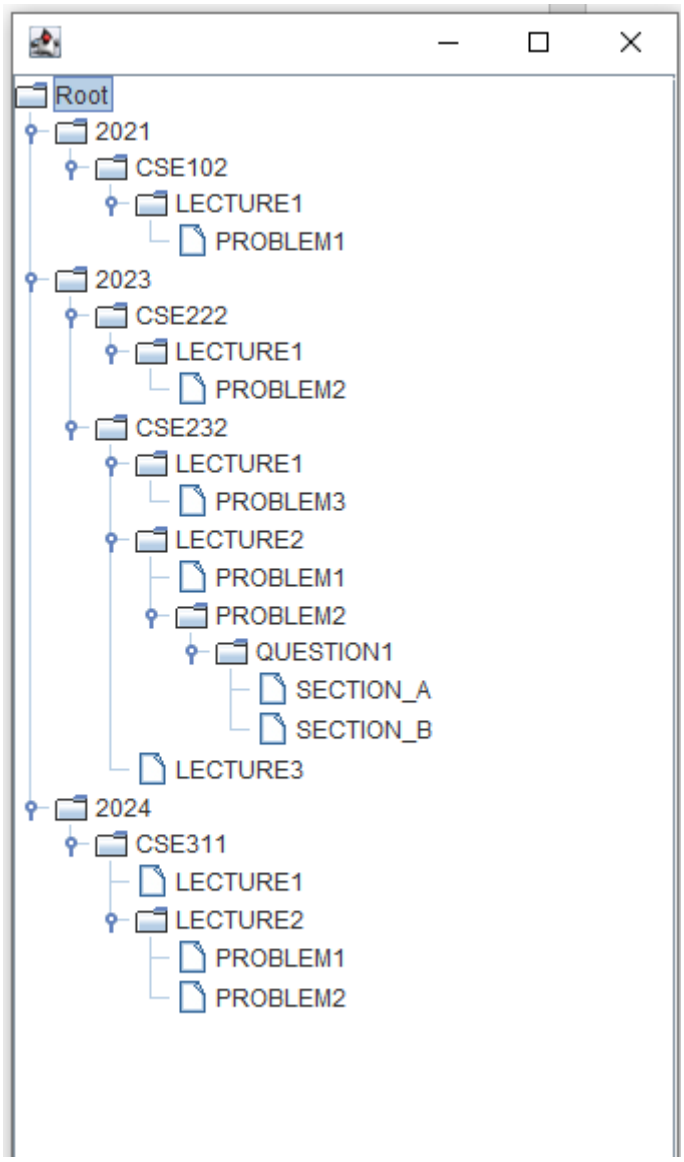
```
2021;CSE102;LECTURE1
2022;CSE321;LECTURE1;LECTURE2
2022;CSE321;LECTURE2
2023;CSE222;LECTURE1;PROBLEM1
2023;CSE222;LECTURE1;PROBLEM2
2023;CSE232;LECTURE1;PROBLEM3
2023;CSE232;LECTURE2;PROBLEM1
2023;CSE232;LECTURE2;PROBLEM2;QUESTION1
2023;CSE232;LECTURE3
2024;CSE311;LECTURE1
2024;CSE311;LECTURE2;PROBLEM1
2024;CSE311;LECTURE2;PROBLEM2
```



tree.txt - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
2023;CSE222;LECTURE1;PROBLEM1
2023;CSE222;LECTURE1;PROBLEM2
2023;CSE232;LECTURE1;PROBLEM3
2023;CSE232;LECTURE2;PROBLEM1
2023;CSE232;LECTURE2;PROBLEM2;QUESTION1
2023;CSE232;LECTURE2;PROBLEM2;QUESTION1
2023;CSE232;LECTURE3
2024;CSE311;LECTURE1
2024;CSE311;LECTURE2;PROBLEM1
2024;CSE311;LECTURE2;PROBLEM2
```



DESIGN:

A) Building the tree:

I have divided the process into several parts:

1) **Reading the file:**
   The input is read from the file, than the elements are parsed and stored in a 2D String array. For this purpose, I implemented FileParser() method, which reads and parses the txt file's input, then returns a 2D String array formed of soon-to-be elements of the tree.

2) **Building the tree:**
   The building process is formed of multiple steps:
   First, each row of the 2D array (this corresponds to each line in the txt file), formed into separate trees, and stored in a list of JTree s. This means at the beginning we have the same amount of trees as the number of lines in the txt file.

   Then, by looping through the subtrees list, the trees are compared to find out if their first nodes are common. If there is two trees with the same root, they are combined to form a single tree with the treeCombiner() tree. This method takes a host tree, and a guest tree, than adds them together, eliminating duplicate nodes by correctly adding them to each other, than returning a JTree.

   Note: At first, I thought of easier ways to do this, but all of a sudden, I wanted to create a generic reusable method for the purpose of combining two trees to form one.

   Than the final JTree is returned.

3) **Displaying the tree:**
   Finally, the tree is displayed using displayTree() method.

B) Search algorithms:

1) BreadthFirstSearch():

The method is designed to perform a breadth-first search of a tree data structure to find a specific value. It begins by initializing two queues: "toBeVisited" and "visitedNodes". The "toBeVisited" queue initially contains the root node of the tree. A counter is used to keep track of the number of steps taken during the search. The method continues to visit nodes in the tree until there are no more nodes to visit. Each visited node is added to the "visitedNodes" queue, and the node's information is printed to the console. If the node contains the value being searched for, the method prints a message indicating that the value has been found and returns "true". If the value is not found, the method returns

"false". Any unvisited child nodes of the current node are added to the "toBeVisited" queue, and the search continues.

2) DepthFirstSearch():

The DepthFirstSearch method takes in a JTree object and a String value to search for. It initializes a stack to hold nodes to be visited and a queue to hold visited nodes. The root node of the tree is added to the stack, and a step counter is initialized. The method then begins a loop that continues until the stack is empty. Within this loop, the top node is removed from the stack, added to the queue of visited nodes, and printed to keep track of the steps. If the node contains the desired value, the method returns true. If not, any unvisited child nodes are added to the stack. The loop continues until there are no more nodes to visit. If the value is not found, the method returns false.

3) PostOrderTraversal():

The postOrderTraversal method performs a post-order traversal of a JTree by calling the postOrderTraversalRecursive helper method on the root node of the tree. The helper method recursively traverses the given DefaultMutableTreeNode in post-order, printing the current node being visited to keep track of steps, and checking if the given value is found in any of its nodes. If the value is found, the method returns true and prints a success message. If the value is not found, the method returns false and prints a "not found" message. The helper method uses a for loop to iterate over each child node of the current node and recursively call itself on each child node. The method stops when it reaches a null node, and returns false if the value is not found.