GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

Homework #5 Report

ÖMER SARIÇAM

200104004009

1)My Solution

Main Method:

The main method contains the primary logic for the program. Firstly, a 10x10 String array is created, and its length is printed. Then, a file is read, and the data is stored in the array. If the data exceeds the array size, the array is resized accordingly.

Afterward, a new root node is created, and the JTree is constructed based on the data in the array. The constructed JTree is then used to perform various operations, which are controlled by user input. The user is presented with a menu containing the various operations, and the selected operation is executed.

Resize Method:

The resize method is used to increase the size of the array when it reaches its limit. The method takes two boolean parameters, indicating whether to increase the row or column size. If the row size needs to be increased, the size is doubled. Similarly, if the column size needs to be increased, the size is doubled. A new array is then created with the new size, and the data from the old array is copied to the new array.

BFSsearchJTree Method:

This method uses Breadth First Search (BFS) algorithm to search a JTree structure for a given input. It takes in two parameters: root and input. The root parameter is the root of the JTree structure that we want to search, and input is the value that we are searching for. First, the method creates a LinkedList called queue to store the nodes that it needs to search through. It starts by adding the root node to the queue. Then, the method enters into a loop that continues until the queue is empty. In each iteration of the loop, it retrieves the first node from the queue using the poll() method. This node is stored in the node variable. Next, the method checks if the value of the current node matches the given input value. If it does, then the method prints a message indicating that the value has been found and returns from the method. If the value of the current node does not match the given input value, then the method proceeds to search the child nodes of the current node. It does this by retrieving all the child nodes of the current node using the children() method and adding them to the queue. This allows the method to search through all the child nodes of the current node before moving on to the next node in the queue. If the method reaches the end of the while loop and has not found the input value, it prints a message indicating that the value was not found. In summary, the BFSSearchJTree method performs a BFS search through the JTree structure to find a given input value. It uses a LinkedList called queue to keep track of the nodes that it needs to search through, and checks each node to see if its value matches the given input value.

DFSsearchJTree Method:

The DFSsearchJTree method performs a depth-first search on the JTree to find the node that matches the given input string. The method takes two parameters: the root node of the JTree and the input string to search for. The search is performed recursively, starting from the root node. The method compares the user object of each node with the input string, and if a match is found, the method prints the path to the node.

PostOrderTraversal Method:
The PostOrderTraversal method performs a post-order traversal on the JTree to find the node that matches the given input string. The method takes two parameters: the root node of the JTree and the input string to search for. The search is performed recursively, starting from the root node. The method traverses the left subtreea, then the right subtrees, and finally the root. The method compares the user object of each node with the input string, and if a match is found, the method prints the path to the node.

moving_process Method:
The moving_process method is used to move a node from one position to another in the JTree. The method takes the JTree as a parameter, and prompts the user to select the node to move and its new location. The method then removes the node from its original position and inserts it into the new position. First, the search_for_moving method is used to check whether the entered path exists in the tree. If it exists, this path is transferred to a new tree using the tree_to_move method. The reason for transferring it to a new tree is the need to take into account child nodes under the entered path. I couldn't think of another data structure to solve this problem. For example, if the input is 2023→CSE231, but the CSE231 node also has LECTURE1 and LECTURE2 nodes, I can add the sub-nodes by creating a new tree. Then, using the path entered by the user, the deleting_themoved_part method is used to remove the path moved by the user from the tree. Finally, the transportation method is used to add this path to the destination entered by the user.