TreeFromTxt class(Part A):

This class has returnTreeFromTxt method that takes the path as argument.

Path is the tree.txt in the assignment and given to this function in main. It uses a fixed size(100) 2D string array to split the input line by line and store the values. From this string, I created a hashmap named nodes which includes the nodelabel and a treeNode.(For ex: root->2022 key will have the node stores “2022” as object and the which is the child node of the node contains “root” object node.) I assigned a parent node for each line of the input( each array element of the 2D String array) and for all the values I updated the nodeLabel by concatenating the current value and calculated the parent node object by whether the nodelabel is in the hashmap or not. If it is in the hashmap, I updated the parentNode by giving the key of the nodeLabel, if it is not in the hashmap i created a new treenode(making it a child of the current parrent), updated the parrentNode by making it the last added node and I added that node to the hashmap. After all the values are added to the tree, function returns the root treeNode.

Search class (Part B, C, D):

This class has the 3 search algorithms specified in pdf.

findBFS:

This function creates a deque, adds the root to the deque and while the deque is not empty, it polls the next element from it and checks whether it is the node containing the object it is searching. If it is not, function simply adds the children of the current node that is checked. So the deque won’t be empty when the children of the parent node is added to the deque and it continuously polls the nodes from the deque to check the node. This way function checks all the nodes. And since we used Deque it is breadth-first.

findDFS:

Same logic as in the BFS but this time I used a stack to make it Depth-first. It Checks the last added first and this makes it depth-first.

findPostTraversal:

I used a recursive helper method to implement this function, it traverses to the tree until it finds a leaf recursively (I called the function again in the for loop by updating the boolean flag that represents whether the object is found, I used this for loop to check all the children of the nodes.) and then it checks for equality, This way when we encounter a parrent node, it won’t be checked until all the children of it checked previously. I calculated the steps by making a step array sized 1 to ease the calculation of the step value because of the recursive call in a for loop would initialize the wrong value each time.

I haven’t done part E.

Main class:

Main function:

It just prints the JTree which is very straightforward process I found in given link, and it continuously asks for a search input to test all 3 of the search methods.