

GEBZE TECHNICAL UNIVERSITY

**CSE 222
DATA STRUCTURES AND ALGORITHMS**

**HOMEWORK 5
REPORT**

**OGUZ MUTLU
1801042624**

1. DETAILED SYSTEM REQUIREMENTS

This homework

- For Input of the system should accept a .txt file as input. Each row in the file represents a data point, and the data is split into categories by the ";" character. Each column in the file can have an arbitrary number of unique values. In each part in this assignment accept different inputs from the user. For example B,C,D parts we have to use console input from taking an input from the user
- Parsing: The system should parse the data from the .txt file and represent it as a tree structure. The tree structure should not necessarily be a binary tree, but it should be a tree structure. And parsing txt file we have to parse txt into 2d string array which sizes grows dynamically
- Printing: The system should be able to print the parsed tree structure in a JTree and JFrame format. This could be as a visual tree diagram, or as a list of nodes and their relationships to one another.
- Error Handling: The system should handle errors gracefully. If the input file is not formatted correctly, the system should provide informative error messages to the user. If the file is not found or there is a problem reading the file, the system should also provide informative error messages.

2.PROBLEM SOLUTION APPROACH AND IMPLEMENTATION

Tree() this function creates a tree from the given 2d string, This function use iterative method behind its idea is,

- first we create a root node of the tree which is root then this function check first row of the array in every splitted character node.add() method invokes by method after first column was written other columns writtten to that node
- I move root reference to the child node and after this reference move iteration (we assign root = child) and then we invoke add function on the root node. It adds every column by root iteration with help of this function

insertTheTree() this function looks if given node is insertable to the tree, tree already include given node data it must skip that node this method check that situation with helper isInclude method

moveTreeObject() if new node is already added to the tree then this function iterates through the children to reach the data node inside the tree root of the tree node data node to insert or iteratively moveif data node object found in the tree returns founded node reference

BFS()

- First I accessed the child nodes starting from root, then I wrote a for loop that will visit each child nodes of root first.
- Within this for loop, I assigned child node I visited to a queue, then I visited the 2nd and 3rd child nodes if exists,
- starting with the first child I get childs child nodes , and assigned these values to the queue in that order. While doing these, at the same time,
- I checked whether the value I was looking for and the value I would assign to the queue at the moment matched, and if the result was correct, I finished the process.

DFSRecursion()

I use stack data structure for this method also write recursive but I choose stack solution in the homework you can also check the code and recursive part.

- Recursively searches the subtree rooted at the given node using the Depth-First Search algorithm. The method visits each node in the subtree by first exploring the rightmost branch until it reaches a leaf node, then backtracks and explores the next unvisited branch.
- The method stops the traversal and returns true when it finds a node with user object equal to the given input. If no such node is found, the method returns false.

DFSStack()

I use stack data structure for this method also write recursive but I choose stack solution in the homework you can also check the code and recursive part.

- Searches the tree using the Depth-First Search algorithm with a stack.
- Starting from the root node, the method traverses the tree by visiting each node's children first before moving on to the next sibling.
- The method stops the traversal and returns true when it finds a node with user object equal to the given input. If no such node is found the method returns false.

PostOrderTraversal()

- Recursively traverses the tree using post-order traversal and searches for the given user input.
- The method starts by visiting the leftmost child, then its siblings, and then the parent.
- If the given user input is found in the tree, it prints the steps taken and returns true.
- If the given user input is not found in the tree, it prints the steps taken and returns false.

move()

This function includes 4 methods as helper

- First I invoke **createNode()** method for create source nodes reference, I keep new tree for source path of the node then if last node has child node I take all last node reference and remove them
- -then our source path is arrange properly then we invoke add() function add function include 1 important function inside whichh is **iterateRoot()** method

this methods usage purpose

- Iterates over a given root node to find a destination node represented by a string of comma-separated values.
- If the destination node is found, it returns the node. Otherwise, it returns null.
- If the destination node is not found, it inserts a new node to the tree and returns the newly created node.

After iteration done our reference which first assigned to the root is now assigned to the destination path in add function we arrange overwritten issue(let call our reference is tempReference), tempReference shows us the destination year but if there are same in the sourcePath and destination year we have to handle that issue I write simple check function iterates all child nodes of two part and then if exactly same nodes are found then I wrote to the console “overwritten” if not I add source path into the new destination path.

- Other method is **cleanTree()** method this method only deletes years which has not any child anymore

3.TEST CASES AND RESULTS

1. Showing the Tree Structure Readed From the Text

```
oguz@oguz-ubuntu: ~/Desktop/github_homeworks/DataStru...
at java.base/java.io.FileInputStream.<init>(File
at java.base/java.util.Scanner.<init>(Scanner.ja
at partA.readFromTxt(partA.java:27)
at testMain.main(testMain.java:14)
oguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
/hw5$ javac *.java
oguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
/hw5$ java testMain
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE102;LECTURE1
3 2022;CSE321;LECTURE1
4 2022;CSE321;LECTURE2
5 2023;CSE222;LECTURE1;PROBLEM1
6 2023;CSE222;LECTURE1;PROBLEM2
7 2023;CSE232;LECTURE1
8 2023;CSE232;LECTURE2;PROBLEM1
9 2023;CSE232;LECTURE2;PROBLEM2
10 2023;CSE232;LECTURE3
```

Root

- 2021
 - CSE102
 - LECTURE1
- 2022
 - CSE102
 - LECTURE1
 - CSE321
 - LECTURE1
 - LECTURE2
- 2023
 - CSE222
 - LECTURE1
 - PROBLEM1
 - PROBLEM2
 - CSE232
 - LECTURE1
 - LECTURE2
 - PROBLEM1
 - PROBLEM2
 - LECTURE3

- Assingment example is in the below.

```
Go Run Terminal Help
partA.java testMain.java 1
tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE321;LECTURE1
3 2022;CSE321;LECTURE2
4 2023;CSE222;LECTURE1;PROBLEM1
5 2023;CSE222;LECTURE1;PROBLEM2
6 2023;CSE232;LECTURE1
7 2023;CSE232;LECTURE2;PROBLEM1
8 2023;CSE232;LECTURE2;PROBLEM2
9 2023;CSE232;LECTURE3
10
4.PostOrderTraversal
5.Move
^Coguz@oguz-ubuntu:~/Desk
ks/hw5$ javac *.java
^[[Aoguz@oguz-ubuntu:~/De
orks/hw5$ java testMain
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
```

Root

- 2021
 - CSE102
 - LECTURE1
- 2022
 - CSE321
 - LECTURE1
 - LECTURE2
- 2023
 - CSE222
 - LECTURE1
 - PROBLEM1
 - PROBLEM2
 - CSE232
 - LECTURE1
 - LECTURE2
 - PROBLEM1
 - PROBLEM2
 - LECTURE3

2.BFS Results

The screenshot shows a terminal window with a Java program running a BFS search. The program's output is as follows:

```
4.PostOrderTraversal
5.Move
2
Enter an input to search
CSE232
Using BFS to find CSE232
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!)
-----HW 5-----
-----
```

To the right, a tree diagram illustrates the search path. The root is 'Root', which points to '2021'. '2021' points to 'CSE102', which points to 'LECTURE1'. '2021' also points to '2022', which points to 'CSE321', which points to 'LECTURE1' and 'LECTURE2'. '2021' also points to '2023', which points to 'CSE222', which points to 'LECTURE1' (containing 'PROBLEM1' and 'PROBLEM2') and 'CSE232'. 'CSE232' points to 'LECTURE1' and 'LECTURE2' (containing 'PROBLEM1', 'PROBLEM2', and 'LECTURE3'). The node 'CSE232' is highlighted in blue, indicating it was the node found by the search.

- Not found example

The screenshot shows a terminal window with a Java program running a BFS search. The program's output is as follows:

```
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.
```

To the right, a tree diagram illustrates the search path. The root is 'Root', which points to '2021'. '2021' points to 'CSE102', which points to 'LECTURE1'. '2021' also points to '2022', which points to 'CSE321', which points to 'LECTURE1' and 'LECTURE2'. '2021' also points to '2023', which points to 'CSE222', which points to 'LECTURE1' (containing 'PROBLEM1' and 'PROBLEM2') and 'CSE232'. 'CSE232' points to 'LECTURE1' and 'LECTURE2' (containing 'PROBLEM1', 'PROBLEM2', and 'LECTURE3'). The node 'LECTURE3' is highlighted in blue, indicating it was the last node visited before the search concluded that the target was not found.

3.DFS Results

```
partA.java testMain.java 1 tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE321;LECTURE1
3 2022;CSE321;LECTURE2
4 2023;CSE222;LECTURE1;PROBLEM1
5 2023;CSE222;LECTURE1;PROBLEM2
6 2023;CSE232;LECTURE1
7 2023;CSE232;LECTURE2;PROBLEM1
8 2023;CSE232;LECTURE2;PROBLEM2
9 2023;CSE232;LECTURE3
10

-----HW 5-----
-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
3
Enter an input to search with DFS.
CSE232
Step 1 -> Root
Step 2 -> 2023
Step 3 -> CSE232(Found!)
-----HW 5-----
```

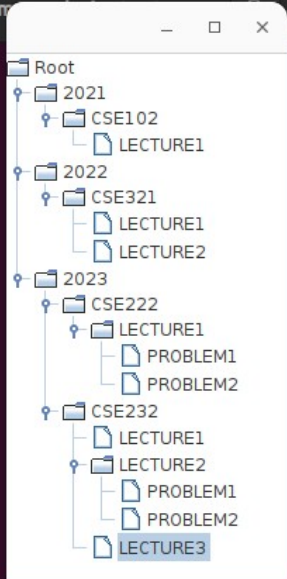
```
partA.java testMain.java 1 tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE321;LECTURE1
3 2022;CSE321;LECTURE2
4 2023;CSE222;LECTURE1;PROBLEM1
5 2023;CSE222;LECTURE1;PROBLEM2
6 2023;CSE232;LECTURE1
7 2023;CSE232;LECTURE2;PROBLEM1
8 2023;CSE232;LECTURE2;PROBLEM2
9 2023;CSE232;LECTURE3
10

3
Enter an input to search with DFS.
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.
-----HW 5-----
```


4.PostOrderTravelsal Results

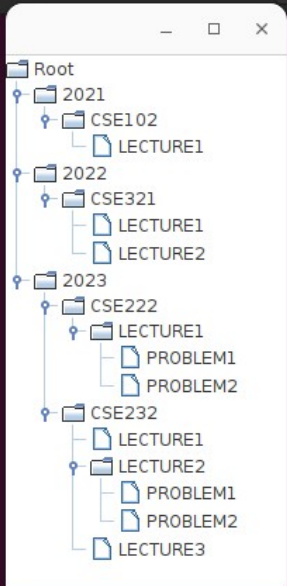
```
partA.java testMain.java 1 tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE321;LECTURE1
3 2022;CSE321;LECTURE2
4 2023;CSE222;LECTURE1;PROBLEM1
5 2023;CSE222;LECTURE1;PROBLEM2
6 2023;CSE232;LECTURE1
7 2023;CSE232;LECTURE2;PROBLEM1
8 2023;CSE232;LECTURE2;PROBLEM2
9 2023;CSE232;LECTURE3
10

3.DFS
4.PostOrderTraversal
5.Move
4
Enter an input to search with BFS.
CSE232
Step 1 -> LECTURE1
Step 2 -> CSE102
Step 3 -> 2021
Step 4 -> LECTURE1
Step 5 -> LECTURE2
Step 6 -> CSE321
Step 7 -> 2022
Step 8 -> PROBLEM1
Step 9 -> PROBLEM2
Step 10 -> LECTURE1
Step 11 -> CSE222
Step 12 -> LECTURE1
Step 13 -> PROBLEM1
Step 14 -> PROBLEM2
Step 15 -> LECTURE2
Step 16 -> LECTURE3
Step 17 -> CSE232(Found!)
-----HW 5-----
```



```
partA.java testMain.java 1 tree.txt
1 2021;CSE102;LECTURE1
2 2022;CSE321;LECTURE1
3 2022;CSE321;LECTURE2
4 2023;CSE222;LECTURE1;PROBLEM1
5 2023;CSE222;LECTURE1;PROBLEM2
6 2023;CSE232;LECTURE1
7 2023;CSE232;LECTURE2;PROBLEM1
8 2023;CSE232;LECTURE2;PROBLEM2
9 2023;CSE232;LECTURE3
10

5.Move
4
Enter an input to search with BFS.
CSE2332
Step 1 -> LECTURE1
Step 2 -> CSE102
Step 3 -> 2021
Step 4 -> LECTURE1
Step 5 -> LECTURE2
Step 6 -> CSE321
Step 7 -> 2022
Step 8 -> PROBLEM1
Step 9 -> PROBLEM2
Step 10 -> LECTURE1
Step 11 -> CSE222
Step 12 -> LECTURE1
Step 13 -> PROBLEM1
Step 14 -> PROBLEM2
Step 15 -> LECTURE2
Step 16 -> LECTURE3
Step 17 -> CSE232
Step 18 -> 2023
Step 19 -> Root
Not found.
```



5.Move Results

The terminal window shows the execution of a Java program. The user has entered the following commands: 3.DFS, 4.PostOrderTraversal, 5.Move, and then the program prompt ^Coguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru... followed by ks/hw5\$ java testMain. The program output shows a menu with 5 options: 1.Show the tree, 2.BFS, 3.DFS, 4.PostOrderTraversal, and 5.Move. The user has selected option 5. The program then prompts for the source of the object (at least 2022,CSE321,LECTURE2) and the destination of the object (year target 2023). The file explorer windows show the directory structure before and after the move operation. In the 'before' state, the directory structure is: Root -> 2021 -> CSE102 -> LECTURE1; Root -> 2022 -> CSE321 -> LECTURE1; Root -> 2023 -> CSE222 -> LECTURE1, PROBLEM1, PROBLEM2; Root -> 2023 -> CSE232 -> LECTURE1, LECTURE2, PROBLEM1, PROBLEM2, LECTURE3. In the 'after' state, the directory structure is: Root -> 2021 -> CSE102 -> LECTURE1; Root -> 2022 -> CSE321 -> LECTURE1; Root -> 2023 -> CSE222 -> LECTURE1, PROBLEM1, PROBLEM2; Root -> 2023 -> CSE232 -> LECTURE1, LECTURE2, PROBLEM1, PROBLEM2, LECTURE3; Root -> 2023 -> CSE321 -> LECTURE2.

```
oguz@oguz-ubuntu: ~/Desktop/github_homeworks/DataStru...
3.DFS
4.PostOrderTraversal
5.Move
^Coguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
ks/hw5$ java testMain
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
5
Enter source of object (at least 2022,CSE321,LECTURE2
Enter destination of object : (year target 2023
```

The terminal window shows the execution of the same Java program. The user has entered the following commands: 3.DFS, 4.PostOrderTraversal, 5.Move, and then the program prompt ^Coguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru... followed by ks/hw5\$ java testMain. The program output shows a menu with 5 options: 1.Show the tree, 2.BFS, 3.DFS, 4.PostOrderTraversal, and 5.Move. The user has selected option 5. The program then prompts for the source of the object (at least 2022,CSE321,LECTURE2) and the destination of the object (year target 2020). The file explorer windows show the directory structure before and after the move operation. In the 'before' state, the directory structure is: Root -> 2021 -> CSE102 -> LECTURE1; Root -> 2022 -> CSE321 -> LECTURE1, LECTURE2; Root -> 2023 -> CSE222 -> LECTURE1, PROBLEM1, PROBLEM2; Root -> 2023 -> CSE232 -> LECTURE1, LECTURE2, PROBLEM1, PROBLEM2, LECTURE3. In the 'after' state, the directory structure is: Root -> 2021 -> CSE102 -> LECTURE1; Root -> 2023 -> CSE222 -> LECTURE1, PROBLEM1, PROBLEM2; Root -> 2023 -> CSE232 -> LECTURE1, LECTURE2, PROBLEM1, PROBLEM2, LECTURE3; Root -> 2020 -> CSE321 -> LECTURE1, LECTURE2.

```
oguz@oguz-ubuntu: ~/Desktop/github_homeworks/DataStru...
3.DFS
4.PostOrderTraversal
5.Move
^Coguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
ks/hw5$ java testMain
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
5
Enter source of object (at least 2 source and 2022,CSE321
Enter destination of object : (year target) 2020
```

Writing to a console of not existing node

The terminal window shows a sequence of commands and their outputs:

```
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
5
Enter source of object (at least 2 source and divided by comma) :
2022,CSE222
Enter destination of object : (year target)
2020
Cannot move 2022 -> CSE222 because it doesn't exist in the tree.
-----HW 5-----
1.Show the tree
2.BFS
```

The two file explorer windows show a tree structure with the following nodes:

- Root
 - 2021
 - CSE102
 - LECTURE1
 - 2022
 - CSE321
 - LECTURE1
 - LECTURE2
 - 2023
 - CSE222
 - LECTURE1
 - PROBLEM1
 - PROBLEM2
 - CSE232
 - LECTURE1
 - LECTURE2
 - PROBLEM1
 - PROBLEM2
 - LECTURE3

writing to a console of overwriting node

The terminal window shows a sequence of commands and their outputs:

```
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
5
Enter source of object (at least 2 source and divided by comma) :
2022,CSE102
Enter destination of object : (year target)
2021
Moved 2022->CSE102 to 2021
2022->CSE102 has been overwritten.
-----HW 5-----
1.Show the tree
2.BFS
```

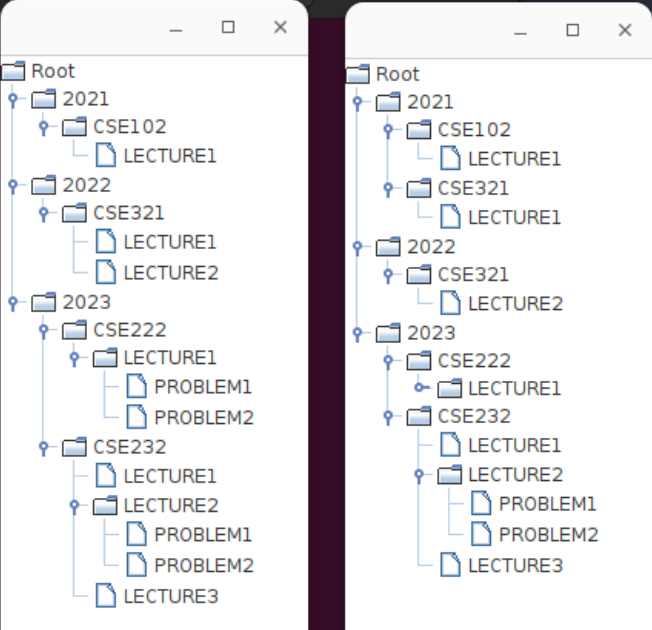
The two file explorer windows show a tree structure with the following nodes:

- Root
 - 2021
 - CSE102
 - LECTURE1
 - 2022
 - CSE102
 - LECTURE1
 - CSE321
 - LECTURE1
 - LECTURE2
 - 2023
 - CSE222
 - LECTURE1
 - PROBLEM1
 - PROBLEM2
 - CSE232
 - LECTURE1
 - LECTURE2
 - PROBLEM1
 - PROBLEM2
 - LECTURE3

Test Case 1

- Move node to another year **PASS**

```
oguz@oguz-ubuntu: ~/Desktop/github_homeworks/DataStru...
Step 16 -> LECTURE3
Step 17 -> CSE232
Step 18 -> 2023
Step 19 -> Root
Not found.
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
1
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
5
Enter source of object (at least 2 source and divided by comma) :
2022,CSE321,LECTURE1
Enter destination of object : (year target)
2021
```



Test Case 2

- Enter Invalid Entry when moving, **PASS**
- Enter less than 2 parsed string in source input **PASS**
- Enter non numeric string value in destination input **PASS**
- Invalid entry for menu **PASS**

```
/hw5$ java testMain
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
6.Exit
ads
Invalid Entry Program Terminates
oguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
/hw5$
```

```
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
6.Exit
5
Enter source of object (at least 2 source and divided by comma) :
CSE222
Enter destination of object : (year target)
2022
Invalid Entry Program Terminates
```

```
-----HW 5-----
1.Show the tree
2.BFS
3.DFS
4.PostOrderTraversal
5.Move
6.Exit
5
Enter source of object (at least 2 source and divided by comma) :
2022,CSE102
Enter destination of object : (year target)
asd
Invalid Entry Program Terminates
oguz@oguz-ubuntu:~/Desktop/github_homeworks/DataStru...
```