

GTU Department of Computer Engineering

CSE 222/505 - Spring 2023

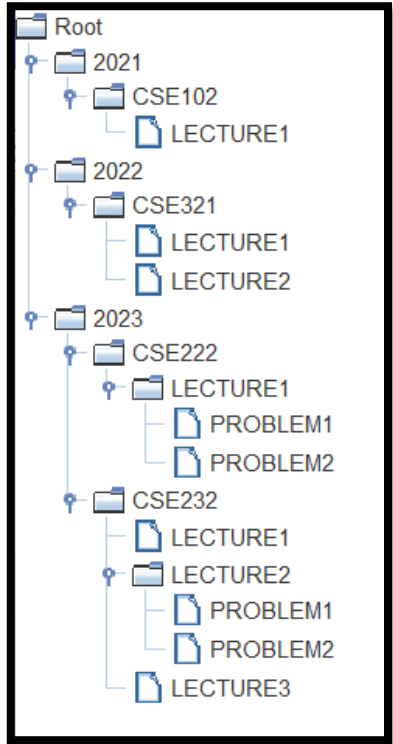
Homework 5

Due date: May 04, 2023 – 23:59

You are expected to write a Java program that performs the following tasks. Make sure your program has a proper OOP structure.

A) 40 pts. Read some information from a .txt file and represent it as a tree structure. In the .txt file, each row represents a data point, and it is split into categories by ";" character. You can assume that the .txt file has a grid structure, and each column might have an arbitrary number of unique values. You should parse this information into a tree (not particularly a binary tree) and then print it.

Example:

Input (content of the txt file)	Output
<pre>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</pre>	

Steps:

- Read the txt file line by line and save the information in a 2D String array.
- Use JTree (from javax.swing library) to create a tree structure. The tree should have a root with a label "Root" as seen in the example above.

- Append each line of the 2D String array as a node to the tree. The data in the first column should be children of the Root, and the data in the second column should be the children of the nodes which represent first column, so on and so forth.
- Make sure you don't create duplicates in the tree. For instance, in the example txt file above there are multiple 2023's but in the tree, there is only one node that represents 2023.
- You should work according to the number of data columns in each line. As can be seen, it may vary.
- Use JFrame structure (which is from the same library as JTree) then embed your JTree into JFrame.
- Print your JFrame as an image, as seen in the example above. For the usage of JTree and JFrame, you can find a simple example [here](#).

Important Notes and Assumptions:

- You can assume that each data line has at least 1 column.
- You can assume that txt file contains capital letters, numbers and ";" only and the data lines are proper, i.e. there is always a ";" in between two columns.
- You cannot use any auxiliary array to keep track of the data points which were added to the tree. You should work on the tree itself to check it.
- Don't forget to add JRE System Library [JavaSE-11] in order to use javax.swing library. Check [this page](#) to see how to add a library by using build path.
- The .txt file should be named as "tree.txt" and it should be in the same folder as the .java files. The file path must be dynamic, i.e. it should run on any computer without any change.
- You don't have to save the image.

B) 10 pts. Apply BFS algorithm to find an input in the tree. You should implement the search algorithm by yourself and do not use any external libraries. Print whether the input is found or not, along with the sequence of steps. The input should be given by the user, which is a String.

Examples:

```
Using BFS to find 'CSE232' 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 (Found!)
```

```

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.

```

C) 10 pts. Repeat part B with DFS. You can use the same input.

Examples:

```

Using DFS to find 'CSE232' in the tree...
Step 1 -> Root
Step 2 -> 2023
Step 3 -> CSE232 (Found!)

```

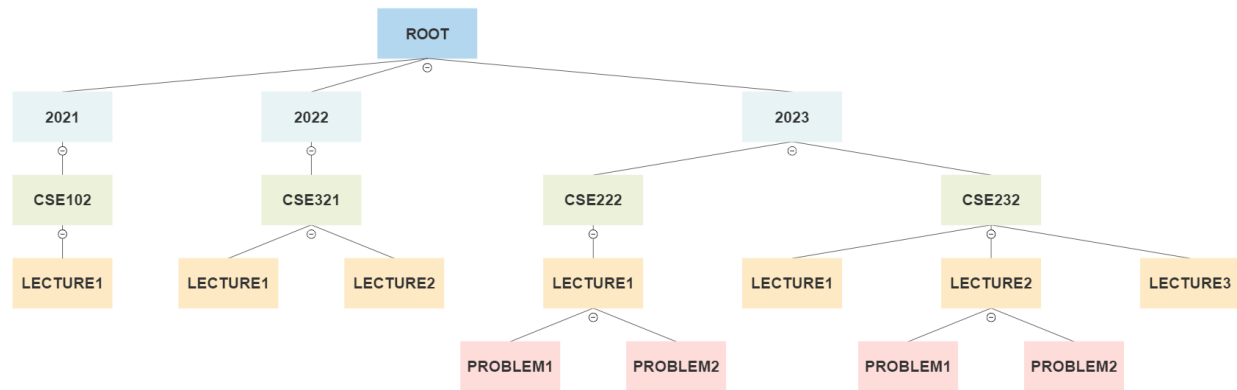
```

Using DFS to find 'CSE2332' in the tree...
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.

```

D) 10 pts. Repeat part B with a post order traversal algorithm. You can use the same input.

Hint: The top-down view of the given example is shown below. You should implement your algorithm considering this.



Examples:

```

Using Post-Order traversal to find 'CSE232' in the tree...
LECTURE1
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!)
  
```

```

Using Post-Order traversal to find 'CSE2332' in the tree...
LECTURE1
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
Not found.
  
```

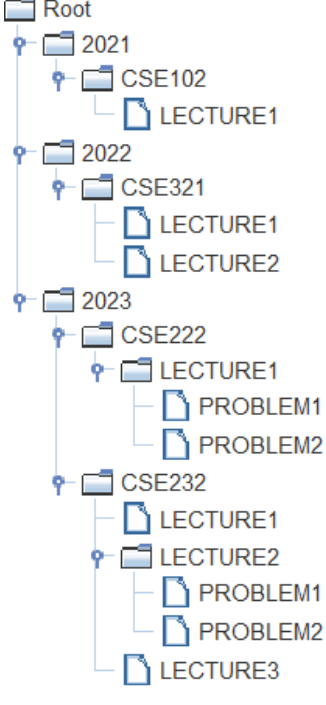
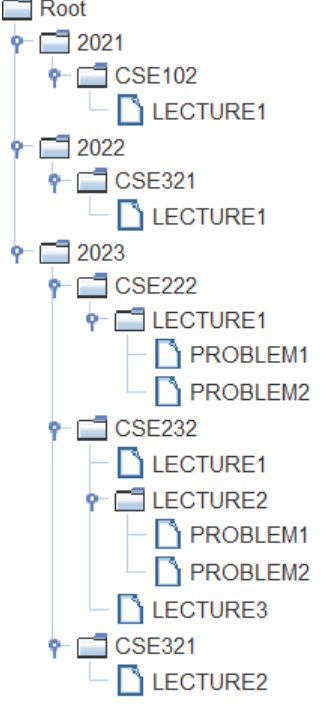
E) 30 pts. Move a problem/lecture/course (any node which is not root or not the child of root) from one year to another. If the problem/lecture/course exists in the destination year, overwrite it and inform the

user about it. After the moving operation, print the tree again. The user should give multiple Strings for the source (for example: 2022, CSE321, LECTURE1 or 2021, CSE102) and a single String (for example: 2023 or 2020).

Don't forget that you are *moving a node*, not *copying* it. Make sure you delete related node(s) if necessary. For instance, in the example above, if we move CSE102 course from 2021 to 2023, then 2021 will be completely empty and therefore it should be deleted as well. Similarly, if the destination year doesn't exist in the tree, then it should be created.

Hint: You can implement a new and simple search algorithm to find the source node.

Examples:

The Original Tree (The tree from part A)	The Edited Tree (Moved 2022->CSE321->LECTURE2 to 2023)
 <pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 CSE321 --> LECTURE2 2023 --> CSE222 CSE222 --> LECTURE1 CSE222 --> PROBLEM1 CSE222 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 CSE232 --> LECTURE2 CSE232 --> PROBLEM1 CSE232 --> PROBLEM2 CSE232 --> LECTURE3 </pre>	 <pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 2023 --> CSE222 CSE222 --> LECTURE1 CSE222 --> PROBLEM1 CSE222 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 CSE232 --> LECTURE2 CSE232 --> PROBLEM1 CSE232 --> PROBLEM2 CSE232 --> LECTURE3 2023 --> CSE321 CSE321 --> LECTURE2 </pre>

The Original Tree (The tree from part A)	The Edited Tree (Moved 2022 -> CSE321 to 2020)

The Original Tree (The tree from part A)	The Edited Tree (actually it was not edited) (Moved 2022 -> CSE222 to 2020)
	Console Output Cannot move 2022->CSE222 because it doesn't exist in the tree.

The Original Tree (The tree from part A)	The Edited Tree (Moved 2023->CSE232->LECTURE1->PROBLEM2 to 2022)
<pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 CSE321 --> LECTURE2 2023 --> CSE222 CSE222 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 CSE232 --> LECTURE2 LECTURE2 --> PROBLEM1 LECTURE2 --> PROBLEM2 CSE232 --> LECTURE3 </pre>	<pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 CSE321 --> LECTURE2 2022 --> CSE232 CSE232 --> LECTURE2 LECTURE2 --> PROBLEM2 2023 --> CSE222 CSE222 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 CSE232 --> LECTURE2 LECTURE2 --> PROBLEM1 LECTURE2 --> PROBLEM2 CSE232 --> LECTURE3 </pre>

The Original Tree (Be careful, this one is different from the other examples.)	The Edited Tree (Moved 2023->CSE321->LECTURE1 to 2022)
<pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 CSE321 --> LECTURE2 2023 --> CSE321 CSE321 --> LECTURE1 2023 --> CSE222 CSE222 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 CSE232 --> LECTURE2 LECTURE2 --> PROBLEM1 LECTURE2 --> PROBLEM2 CSE232 --> LECTURE3 </pre>	<pre> graph TD Root --> 2021 Root --> 2022 Root --> 2023 2021 --> CSE102 CSE102 --> LECTURE1 2022 --> CSE321 CSE321 --> LECTURE1 CSE321 --> LECTURE2 2022 --> CSE222 CSE222 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 2023 --> CSE232 CSE232 --> LECTURE1 LECTURE1 --> PROBLEM1 LECTURE1 --> PROBLEM2 CSE232 --> LECTURE2 LECTURE2 --> PROBLEM1 LECTURE2 --> PROBLEM2 CSE232 --> LECTURE3 </pre>
	Console Output Moved 2023->CSE321->LECTURE1 to 2022. 2022->CSE321->LECTURE1 has been overwritten.

General Information

- You shouldn't use any libraries other than the ones below:
 - o Java.swing
 - o Java.io.File
 - o Java.io. FileNotFoundException
 - o Java.util.Scanner
 - o Java.util.Arrays
 - o Java.util.Queue
 - o Java.util.LinkedList
 - o Java.util.Map / Java.util.HashMap
 - o Java.util.Stack

PS: If you think a library other than the ones above is essential, contact us.

- Do not use any methods from the imported libraries to do the work you are asked to do. For example, if JTree has a method to construct a tree with a given input or a method to implement BFS, you cannot use them. All of the tasks explained above should be implemented by you. You can only use some data structures like queue, stack, etc. or a class to represent/display data like JTree or JFrame. You cannot also use static variables to avoid parameter passing. This applies to all of your homeworks.
- You are free to use data structures / programming techniques that you have learned in the class till now. So, it is up to use recursion or not, for instance.
- Make sure your code works with arbitrary input. Your solution will be graded by using a different .txt file. There might be more columns (for example, maybe there will be SOLUTION1 and SOLUTION2 under a node named PROBLEM1). You should read the data dynamically.

Grading Details

No OOP Design	-100 points
No error handling	-50 points
No report (a report that explains the details and the manner of work of your solution)	-90 points
No javadoc documentation	-50 points
Cheating (yes, including ChatGPT)	-200 points
Declaring the input within the code, instead of reading it from user	-20 points
Using any library other than the ones stated above	-50 points
Not implementing part A (Not printing the tree means part A doesn't work)	-100 points (you cannot get points from the other parts if you didn't implement part A).
Not printing the steps in part B	-10 points (no steps -> no grade)
Not printing the steps in part C	-10 points (no steps -> no grade)
Not printing the steps in part D	-10 points (no steps -> no grade)

Not implementing the search algorithm correctly in part E	-5 points.
Not deleting the source node from the tree in part E	-5 points
Deleting the source node but not deleting its parent (if needed) in part E	-5 points
Not adding the node to the destination in part E	-5 points
Not adding a new parent (if needed) in part E	-5 points
Not implementing overwrite operation in part E	-5 points

Contact

Res. Asst. Sibel Gülmez (sgulmez2018@gtu.edu.tr)