

Project 3: Tree Basics

DUE: April 7th, 2023 at 11:59pm

Extra Credit Available for Early Submissions!

Basic Procedures

You must:

- Fill out a readme.txt file with your information (goes in your user folder, an example readme.txt file is provided).
- Have a style (indentation, good variable names, etc.) and pass the provided style checker (See P0).
- Comment your code well in JavaDoc style and pass the provided JavaDoc checker (See P0).
- Have code that compiles with the command: `javac *.java` in your user directory without errors or warnings.
- For methods that come with a big-O requirement (check the provided template Java files for details), make sure your implementation meet the requirement.
- Implement all required methods to match the expected behavior as described in the given template files.
- Have code that runs with the commands: `java KTreeUI InputFile`

You may:

- Add additional methods and class/instance variables (unless specified otherwise by the template files), however they **must be private**.

You may NOT:

- Make your program part of a package.
- Add additional public methods or public class/instance variables, remember that local variables are not the same as class/instance variables!
- Use any built in Java Collections Framework classes anywhere in your program (e.g. no ArrayList, LinkedList, HashSet, etc.).
- Use any arrays or add any additional class/instance variables in `FcnstTreeNode` class.
 - You may not call toArray() methods to bypass this requirement.
- Alter any method signatures defined in this document of the template code. Note: “throws” is part of the method signature in Java, don’t add/remove these.
- Alter provided classes/methods that are complete (e.g. `KTreeUI`, `equals()` in `FcnstTreeNode`, etc.).
- Add any additional import statements (or use the “fully qualified name” to get around adding import statements).
- Add any additional libraries/packages which require downloading from the internet.

Setup

- Download the `p3.zip` and unzip it. This will create a folder `section-yourGMUUserName-p3`;
- Rename the folder replacing `section` with the `004` or `006` based on the lecture section you are in;
- Rename the folder replacing `yourGMUUserName` with the first part of your GMU email address;
- After renaming, your folder should be named something like: `000-yzhong-p3`.
- Complete the `readme.txt` file (an example file is included: `exampleReadmeFile.txt`).

Submission Instructions

- Make a backup copy of your user folder!
- Remove all test files, jar files, class files, etc. You should just submit your java files and your readme.txt
- Zip your user folder (not just the files) and name the zip `section-username-p3.zip` (no other type of archive) following the same rules for `section` and `username` as described above. For example:
 - `000-yzhong-p3.zip --> 000-yzhong-p3 --> JavaFile1.java
JavaFile2.java
JavaFile3.java ...`
- Submit to blackboard.

Grading Rubric

Due to the complexity of this assignment, an accompanying grading rubric pdf has been included with this assignment. Please refer to this document for a complete explanation of the grading, including extra credit for early submissions.

Overview

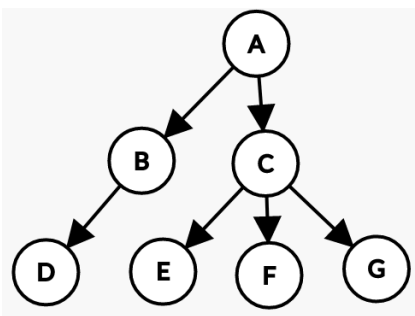
Trees can be implemented using **an array** or a **linked structure** as the internal storage. We will practice both approaches in this project. We will also construct an open-addressing hash table to help us to inspect the tree more conveniently.

Array-based K-ary Tree

Given a fixed K , an array-based K -ary tree uses an internal array to store tree nodes in level order. We will assume the following features:

- Every index corresponds to one node location for a perfect K -ary tree. Root node is at index 0.
- All nodes are stored in the array level by level, starting from the root and going from top down.
- Nodes in the same level are stored from left to right in an ascending order of indexes.
- If a node is not present in the current tree, we reserve an array entry for it and set the entry to be null.
- The internal array has the exact number of entries to hold the perfect tree of the current tree height.

The example below shows a tree in the normal graphical representation and its corresponding array storage.



Assumptions:

- $K=3$.
- B and C are the left two children of A.
- D is the left most child of B.
- Underscore marks (" _ ") indicate null array entries.

A	B	C	_	D	_	_	E	F	G	_	_	_
0	1	2	3	4	5	6	7	8	9	10	11	12

Example 1: K-ary tree and its array-based storage

As we discussed in class, when we use this array-based approach to store tree nodes, math equations can be used to describe the relationship between the array index of a parent and the indexes of its children (if present). For example, assuming a binary tree ($K=2$) and tree root is at index 0, then:

- If parent node is at index p :
 - Index of its left child = $2p+1$; index of its right child = $2p+2$.
- If a child is at index c :
 - Index of its parent = $\text{floor}((c-1)/2)$.

The equations of indexes can be generalized to any K value and you might find them convenient in supporting various tree operations.

Open-addressing Hash Table

We will keep all values in our K -ary tree as a set. A hash table will be constructed to ensure no duplicated values are added into the tree and facilitate an efficient inspection of the tree nodes. The idea is to store pairs of $\langle \text{value}, \text{index} \rangle$ in the hash table to remember which tree node is used to hold the given value. For example, the hash table maintained for the tree from Example 1 would include the following pairs (un-ordered): $\langle A, 0 \rangle$, $\langle B, 1 \rangle$, $\langle C, 2 \rangle$, $\langle D, 4 \rangle$, $\langle E, 7 \rangle$, $\langle F, 8 \rangle$, $\langle G, 9 \rangle$. For this assignment, we will implement an open-addressing hash table and use linear probing to handle collisions.

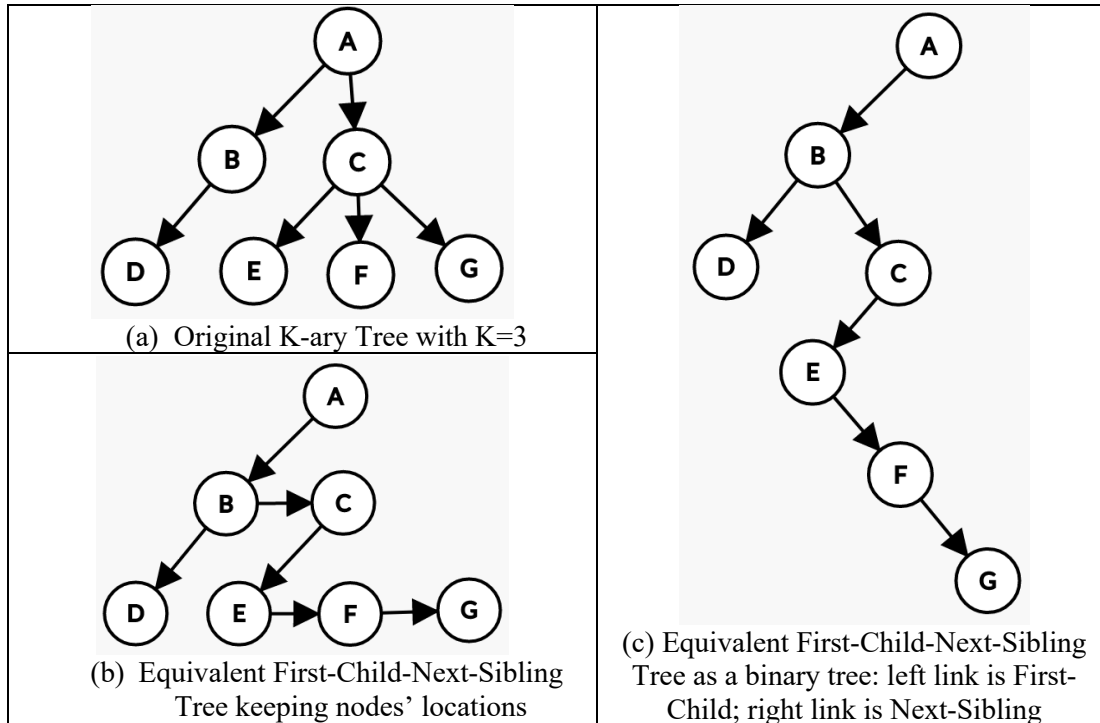
Link-based First-Child-Next-Sibling Tree

One convenient approach to represent K -ary trees with any given K using a **binary** tree node is to represent the tree in a **First-Child-Next-Sibling** format. The idea is to keep only two links in each tree node:

- First-child link: pointing to the first (leftmost non-null) child of this node in the K -ary tree (null if the node is a leaf), and
- Next-sibling link: pointing to the next (right) non-null sibling of this node in the K -ary tree (null if this node is the rightmost child of its parent).

Check our **textbook Ch18.1.2 (pp. 653)** for the definition and more examples of First-Child-Next-Sibling trees.

The example below shows the same K-ary tree as from Example 1 and its First-Child-Next-Sibling tree.



Example 2: K-ary tree and its first-child-next-sibling tree

For this assignment, we will need to create the equivalent first-child-next-sibling tree from a K-ary tree and be able to use the generated first-child-next-sibling tree to simulate tree traversals of the original K-ary tree. Understanding the definition or semantics of the two links in each first-child-next-sibling tree node would be critical.

Implementation/Classes

This project will be built using classes representing a hash table, a K-ary tree, and a first-child-next-sibling tree. All three data structure classes are generic. Here we provide a description of these classes. Template files are provided for each class in the project package and these contain further comments and additional details. You must follow the instructions included in those files.

- **ThreeTenKTree (ThreeTenKTree.java):** The implementation of a generic K-ary tree using an internal array storage.
- **FcnsTreeNode (FcnsTreeNode.java):** The implementation of a first-child-next-sibling representation of a K-ary tree.
- **ThreeTenHashTable (ThreeTenHashTable.java):** The implementation of a hash table using linear probing to handle collisions.
- **KTreeUI (KTreeUI.java):** A textual user interface class with a menu system to invoke different tree operations. This class is provided to you and you should NOT change the file.
- There are a number of methods and/or internal classes already implemented and provided to you in some of the java classes above. Do not change the provided parts but you will need to add JavaDocs for them.

Requirements

An overview of the requirements is listed below, please see the grading rubric and template files for more details.

- **Implementing the classes** - You will need to implement required classes and fill the provided template files.
- **JavaDocs** - You are required to write JavaDoc comments for all the required classes and methods.

- **Big-O** - Template files provided to you contains instructions on the REQUIRED Big-O runtime for many methods. Your implementation of those methods should NOT have a higher Big-O.

How To Handle a Multi-Week Project

While this project is given to you to work on for about three weeks, you are unlikely to be able to complete everything in one weekend. We recommend the following schedule:

- Step 1 (**ThreeTenHashTable**): Before the first weekend (by 03/24)
 - Implement and test methods of **ThreeTenHashTable**.
- Step 2 (**ThreeTenKTree**): First weekend (03/25-03/26)
 - Start implementing **ThreeTenKTree**, everything not related to first-child-next-sibling trees.
- Step 3 (**ThreeTenKTree**, **FcnsTreeNode**): Before second weekend (03/27-03/31)
 - Complete and test methods of **ThreeTenKTree**, including the creation of first-child-next-sibling tree.
 - Start implementing **FcnsTreeNode**.
- Step 4 (**FcnsTreeNode**): Second weekend (04/01-04/02)
 - Complete and test all methods of **FcnsTreeNode** and **ThreeTenKTree**.
- Step 5 (**Wrapping-up**): Last week (04/03-04/07)
 - Additional testing, debugging, get additional help.
 - ☺ Also, notice that if you get it done early in the week, you can get extra credit! Check our grading rubric PDF for details.

Testing

The main methods provided in the template files contain useful example code to test your project as you work. You can use command like "java **ThreeTenHashTable**" or "java **ThreeTenKTree**" to run the testing defined in **main()**.

- **Note:** passing all yays does NOT guarantee 100% on the project! Those are only examples for you to start testing and they only cover limited cases. Make sure you add many more tests in your development. You could edit **main()** to perform additional testing.
- JUnit test cases will not be provided for this project, but feel free to create JUnit tests for yourself. A part of your grade *will* be based on automatic grading using test cases made from the specifications provided.

The provided **KTreeUI** can be run with an input file as the command line argument.

- The file is used to initialize a K-ary tree. A valid file needs to be provided in order to specify the branching factor K and the tree nodes. We provide a number of input files that you can use with **KTreeUI** under the folder **input/**.
- File I/O are all processed in the provided **KTreeUI**. You can find an explanation of the file format in the Appendix of this document.

You can use the provided menu options from **KTreeUI** to invoke different tree operations. We include for sample input files the graphical representation of the K-ary tree and its first-child-next-sibling tree in the Appendix. Since the menu system is straightforward, we only include one sample run. You should be able to use the provided interface and input files to test/verify the implementation of different tree operations.

- **Note:** similarly, passing the checking of all provided sample files does NOT guarantee 100% on the project!
- You should test with more scenarios and input files of your own.

Appendix: Input File Format

The provided **KTreeUI** needs to run with an input file to initialize a K-ary tree. The needed file I/O has been implemented in `KTreeUI.java`. We explain the format of input files here to help you to understand the initialization of trees better. You should also come up with additional input files to test your P2 with more K-ary trees.

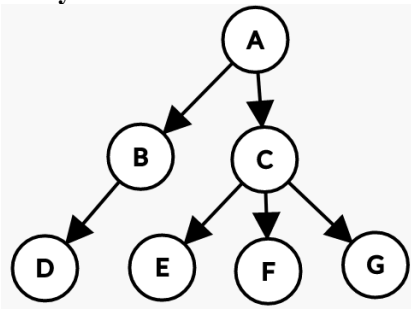
We include multiple sample input files you can use in under **input/** folder. All input files follow the same format as described below:

- Each file starts with a positive integer **K** which defines the branching factor of the tree. You can assume **K** ≥ 2 .
- After the branching factor, the file contains one or more strings, separated by whitespaces. **Assumptions:**
 - Each string is a value we need to add into the K-ary tree.
 - All values are given in the level order of tree nodes in a perfect K-ary tree.
 - A single underscore "_" denotes a null node.

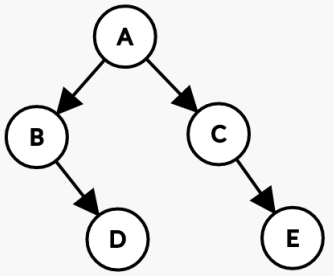
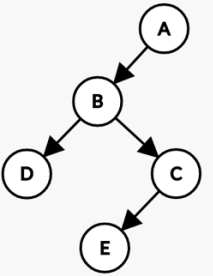
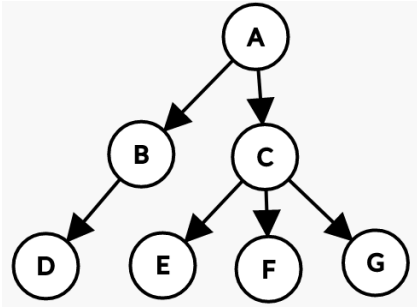
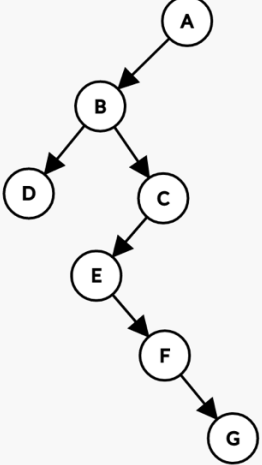
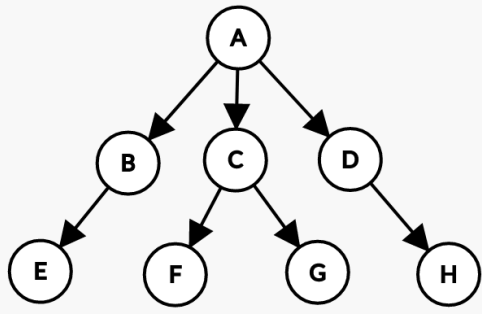
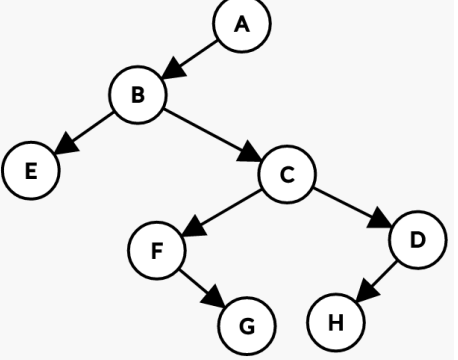
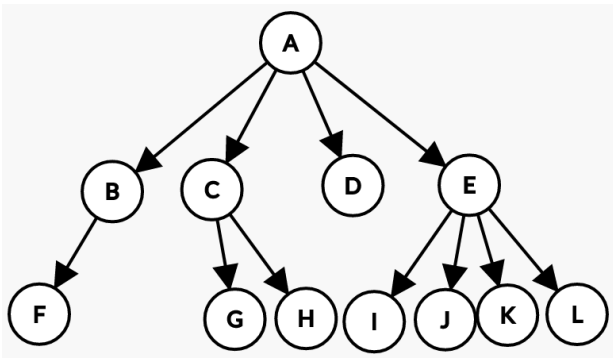
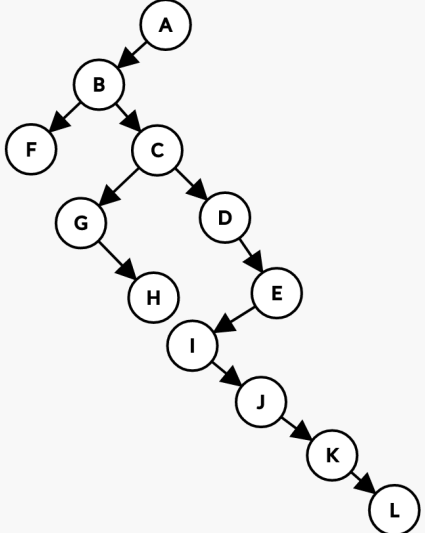
Example input: trinary1.txt

```
3  
A B C _ D _ _ E F G _ _ _
```

K-ary tree corresponding to trinary1.txt:



Appendix: Sample K-ary Trees and FCNS Trees

Input File	K-ary Tree	FCNS Tree
binary1.txt (K=2)		
trinary1.txt (K=3)		
trinary2.txt (K=3)		
quad1.txt (K=4)		

Appendix: Sample Run

```
java KTreeUI input/trinary1.txt
```

Initialize with trinary1.txt

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 1

A
B C null
D null null E F G null null null

Display all nodes in level-order, one level per line, including null nodes.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 2

A B C D E F G

Display all nodes in level-order, excluding null nodes.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 3

FCNS Tree Level-Order:A B D C E F G

FCNS Tree Post-Order:D G F E C B A

Display its FCNS tree: all nodes in level-order and post-order traversals.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree

- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 4

FCNS Tree Simulating K-ary Tree Level-Order:A B C D E F G

FCNS Tree Simulating K-ary Tree Pre-Order:A B D C E F G

FCNS Tree Simulating K-ary Tree Post-Order:D B E F G C A

Use its FCNS tree to simulate tree traversals of the original K-ary tree: level-order, pre-order, and post-order.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 5

Please enter the name of a value/node: C

Path from root to this node:

A-->C

Children of this node:

E F G

Display node info from the original K-tree (path from root and children).

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 5

Please enter the name of a value/node: D

Path from root to this node:

A-->B-->D

Children of this node:

Display node info from the original K-tree (path from root and children).

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 6

Please enter the node/value to remove: X
Value X not present.

Remove a node: node must be present.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
 - 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
 - 3 - Display its FCNS tree
 - 4 - Use FCNS tree to simulate K-ary tree traversals
 - 5 - Show details of one node in tree
 - 6 - Remove a node
 - 7 - Exit
-

Your choice (1-7): 6

Please enter the node/value to remove: A
Cannot remove a value in a non-leaf node.

Remove a node: cannot remove a non-leaf node.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
 - 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
 - 3 - Display its FCNS tree
 - 4 - Use FCNS tree to simulate K-ary tree traversals
 - 5 - Show details of one node in tree
 - 6 - Remove a node
 - 7 - Exit
-

Your choice (1-7): 6

Please enter the node/value to remove: E
Value E removed.

Remove a node: successfully removed.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
 - 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
 - 3 - Display its FCNS tree
 - 4 - Use FCNS tree to simulate K-ary tree traversals
 - 5 - Show details of one node in tree
 - 6 - Remove a node
 - 7 - Exit
-

Your choice (1-7): 1

A

B C null

D null null null F G null null null

K-ary tree after E removed.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node

Your choice (1-7): 2

A B C D F G

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 4

FCNS Tree Simulating K-ary Tree Level-Order:A B C D F G

FCNS Tree Simulating K-ary Tree Pre-Order:A B D C F G

FCNS Tree Simulating K-ary Tree Post-Order:D B F G C A

FCNS tree simulation after
E removed.

Please select from the following options:

- 1 - Display whole K-ary tree (w/ null nodes)
- 2 - Display K-ary tree in LEVEL-ORDER (w/o null nodes)
- 3 - Display its FCNS tree
- 4 - Use FCNS tree to simulate K-ary tree traversals
- 5 - Show details of one node in tree
- 6 - Remove a node
- 7 - Exit

Your choice (1-7): 7

Good-bye!