Com S 228 Fall 2013

Assignment 4: Trees of Entries of Key Sequence and Value

Due at 11:50 pm on Wednesday, November 20

Introduction

This assignment gives you opportunities to write a completely new generic entry tree class according to the following requirements. These requirements are selected to help you develop independent skills of writing methods for manipulating entry trees. Thus, you are provided with only a minimal set of template code. In addition, this assignment gives you opportunities to sharpen your testing skills with JUnit by writing JUnit test code for the entry tree class. Note that we have designed this assignment specially for this course, so some terms used in this assignment may be different from what other instructors have used before. Doing this assignment will help you develop the ability to understand trees and to write and debug well-structured programs.

An entry tree is a rooted tree with each node having any number of unordered children that are each given a unique key of type `K`. The root node is a dummy node with no key (key is null). Since the children of each node have different keys, a unique sequence of keys is obtained by walking down any path from the root to any other node and concatenating the keys at these nodes in order. So each path from the root to any other node corresponds to a unique sequence of keys. In addition to the key type `K`, there is another data type `V` called a value type. The root has no value (value is null), whereas every leaf node that is not the root has a value. Any internal node that is not the root either has a value or has no value. A pair of key sequence and value is called an entry. An entry tree is used to represent a mapping from key sequences to values with the property that if a node other than the root has a value, then the key sequence on the path from the root to the node is mapped to the value. Entry trees are useful as a data structure for representing a dictionary with the key type `K` being `Character` and the value type `V` being `String` and an Internet IP address directory in which each unique IP address as a sequence of `Integer` is assigned to an organization as a value.

An entry tree is implemented by using a linked structure (see a picture of an example entry tree in an attachment). In this structure, the children of a node are stored in a doubly linked list with no dummy node and not circular. Each node has a `child` link referring to the first node of the doubly linked list for its children. The `child` link of the node is `null` if it has no child. The nodes in the doubly linked list are connected together with links `next` and `prev`. The `prev` link of the first node in any doubly linked list is always `null`; the `next` link of the last node in any doubly

linked list is always `null`. Each node has a `parent` link; the `parent` link of the root node is always `null`. The `parent` link of every child node refers to its parent node. In addition, each node has two types of data: key of type `K` and value of type `V`. The `key` and `value` fields of the root node are always `null`. Every other node has a `key` field that is not `null`. If a node has no value, then its `value` field is `null`. The `value` field of every leaf node that is not the root is never `null`.

You should first look at the attached input and output files before you read the specification of each method below in detail.

Requirements for class EntryTree<K, V>

Write a public class named `EntryTree` with five public methods. The specification of each method is given below. You should introduce common private methods to avoid duplication of code. Note that a recursive method is easier to write than an iterative method. Both types of methods are acceptable for this assignment. We suggest that you first write the `showTree` method so that you can check if the tree constructed by other methods is correct by printing out the tree. In addition, you need to write a JUnit test set for each public method except `showTree`.

```
public V search(K[] keyarr)
```

The method returns `null` if `keyarr` is `null` or its length is 0. If any element of `keyarr` is `null`, then the method throws a `NullPointerException` exception. The method returns the value of the entry with the key sequence specified in `keyarr` or `null` if this tree contains no entry with the key sequence. An example is given in an attachment to illustrate this method.

```
public K[] prefix(K[] keyarr)
```

The method returns `null` if `keyarr` is `null`, or its length is 0. If any element of `keyarr` is `null`, then the method throws a `NullPointerException` exception. A prefix of the array `keyarr` is a key sequence in the subarray of `keyarr` from index 0 to any index $m \geq 0$; the corresponding suffix is a key sequence in the subarray of `keyarr` from index $m + 1$ to index `keyarr.length - 1`. The method returns an array of type `K[]` with the longest prefix of the key sequence specified in `keyarr` such that the keys in the prefix are, respectively, with the nodes on the path from the root to a node. The length of the returned array is the length of the longest prefix. Note that if the length of the longest prefix is 0, then the method returns `null`. An example is given in the attachment to illustrate this method.

```
public boolean add(K[] keyarr, V aValue)
```

The method returns `false` if `keyarr` is `null`, its length is 0, or `aValue` is `null`. If any element of `keyarr` is `null`, then the method throws a `NullPointerException` exception. The method locates the node P corresponding to the longest prefix of

the key sequence specified in `keyarr` such that the keys in the prefix label the nodes on the path from the root to the node. If the length of the prefix is equal to the length of `keyarr`, then the method places `aValue` at the node P and returns `true`. Otherwise, the method creates a new path of nodes (starting at a node S) labelled by the corresponding suffix for the prefix, connects the prefix path and suffix path together by making the node S a child of the node P, and returns `true`. An example is given in the attachment to illustrate this method.

`public V remove(K[] keyarr)`

The method returns `null` if `keyarr` is `null` or its length is 0. If any element of `keyarr` is `null`, then the method throws a `NullPointerException` exception. The method returns `null` if the tree contains no entry with the key sequence specified in `keyarr`. Otherwise, the method finds the path with the key sequence, saves the `value` field of the node at the end of the path, sets the `value` field to `null`. If any leaf node is the only child of its parent and has `null` in its `value` field, then the leaf node is removed. This step is repeated until no leaf node in the resulting tree meets this condition. Finally, the method returns the saved old value.

`public void showTree()`

The method prints the tree on the console in the output format shown in the attachment. If the tree has no entry, then the method just prints out the line for the dummy root node.

Requirements for class Dictionary

Write a class named `Dictionary` with a `main()` method for using the `EntryTree` class to process a file of commands (one per line). Each command line consists of the name of a public method of the `EntryTree` class followed by its arguments in string form if the method has arguments. The name of the file is available to the `main()` method from its `args` argument at index 0. You can assume that the command file is correct in format. The `main()` method creates an object of the `EntryTree` class with `K` being `Character` and `V` being `String`, reads each line from the command file, decodes the line into `String` parts, forms corresponding arguments, and call the public method from the `EntryTree` object with the arguments, and prints out the result on the console. Note that the name of a public method in the `EntryTree` class on each command line specifies that the public method should be called from the `EntryTree` object. A sample input file of commands and a sample output file are attached.

The sample output file was produced by directing the console output to a file. The output from your program does not need to look exactly like the sample output file.

Requirements for inner class Node

The `Node` class needs to implement the `EntryNode<K, V>` interface in a code template for automated testing.

Submission

You are required to include, in your submission, the source code for each of the two classes: `EntryTree` and `Dictionary` and the source code for the `EntryNode` interface. A short template is given in package cs228hw4.zip. You need to write proper documentation with JavaDoc for each method in the `EntryTree` class. You are required to submit JUnit tests and are not allowed to share your JUnit tests on Blackboard Learn.

Write your class so that its package name is edu.iastate.cs228.hw4. Your source files (.java files) will be placed in the directory edu/iastate/cs228/hw4 (Linux) or edu\iastate\cs228\hw4 (Windows), as defined by the package specified above. Be sure to put down your name after the @author tag in each class source file. Your zip file should be named Firstname_Lastname_HW4.zip. You may submit a draft version of your code early to see if you have any submission problem with Blackboard Learn. We will grade only your latest submission.