COMPUTER SCIENCE 21A (SPRING, 2020)
DATA STRUCTURES AND ALGORITHMS

PROGRAMMING ASSIGNMENT 2 – WORKTIME ANALYSIS

Overview:

You have been hired by the Brandeis Computer Science department to implement a program that analyzes their employees' worktime. A user will provide you with a comma separated value file that holds an employee's work entries. Each work entry contains an activity name. Users will want to bring up information based on an activity name. However, work entries may not always be recorded with the same exact activity name. For example, a teaching assistant employee may record grading related entries with the activity names "grading pa", "grading test", "grading exam", etc. Therefore, someone who is interested in retrieving information on grading in general will have to search all these different activities mentioned previously. Therefore, a user of this application is very likely to make similar, repeated searches. To build this searching application, you will be using a *splay tree* because it is a data structure that takes advantage of locality.

Program Behavior:

The program will read data from a CSV file into Java `WorkEntry` objects. *The* `WorkEntry` *class and CSV-handling will be implemented for you.* Then, these `WorkEntry` objects must be prepared in data structure(s) that will optimize searching and deleting from the data set. Lastly, the user will be able to search and remove entries through a simple command line.

Command Line Requirements:

You will have to provide functionality for the following commands. You should read these instructions carefully along with the **Example Output** on pages 2-3 to fully understand the required behavior.

- `search "activity"` - this command retrieves all of the work entries with a certain activity *surrounded in quotation marks*. If you search for an activity, the program should print all the WorkEntry objects that match this activity followed by the total time spent on that activity or **nothing** if the activity has no associated work entries.
- `list l` - this **l**exicographically lists all possible activities that can be searched.
- `list r` - this command lists all possible activities ordered in such a way that activities similar to the most **r**ecent tree operations are displayed first. Recent tree operations will include searches, insertions, and deletions.
- `del i` - This command allows the user to remove an entry from the data set. The user can input del followed by an integer i to specify the index of the entry to delete displayed by the last search command *if the last search command was successful.* For repeated

deletes, the index will be assumed to refer to the indices following the effects of the latest delete. If one called ‘del 0’ twice, the second ‘del 0’ would delete the entry that was at index 1 before the first ‘del’ call. This is further explained in the **Example Output** on pages 2-3.

- If the last search command was not successful, then there are no entries to index for a del command. Thus, the tool is in an “invalid state”. Therefore, **you must throw an IllegalStateException** if the user tries to execute the del command when there are no entries to index. For more on this, please see #5 of the **Design Questions** on page 6.
- You do not have to check if the user passed in an integer. However, if there are entries to index, **you must check** that the integer can be associated with one of them. If the index cannot be associated with one of the searched entries, **you must throw** an **IndexOutOfBoundsException**.

Notes:

- The input CSV data and search queries will be *lowercase*.
- The input CSV time data will either be *full or half hours.*
- The activity searches will never contain phrases that contain quotation marks.
- **The commands the user enters will be structured properly (following the form below).**

Example Output:

The following table represents possible work entries of a Computer Science Department teaching assistant. This table has been provided in CSV format as part of the skeleton code and is used by the provided example JUnit test.

Date	Time Spent	Activity	Description
9/18/2019	1.5 h	office hours	helped students with pa1
9/19/2019	0.5 h	grading hw	grading late hw
9/20/2019	3 h	grading exam	graded exam #1
9/22/2019	1 h	met with tas	
9/25/2019	2 h	grading hw	graded pa1
10/1/2019	1 h	office hours	

Here are a few examples of possible commands and their output based on this table.
Comments to explain the output are *italicized* and user input is **bolded**.

```
Enter the CSV file path: pdf_table_csv.txt
> list r                          listing based on latest added node
office hours
met with tas
grading hw
grading exam
> list l                                    listing alphabetically
```

```
grading exam
grading hw
met with tas
office hours
> del 0    haven't called search, illegal state exception thrown
[del] command must be preceded by a [search] command
> search "grading pa"  nothing printed after unsuccessful search
> list r            search did affect the recent "tree activity"
grading hw
grading exam
met with tas
office hours
> search "office hours"     results of WorkEntry's toString()
[9/18/2019] office hours (1.5 h): helped students with pa1
[10/1/2019] office hours (1.0 h):
Total: 2.5 h
> list r  recent tree activity affects the result of this cmd
office hours
met with tas
grading hw
grading exam
> del 2               this is referencing office hours search
[del] command received an invalid index
> del 0                         this will delete entry for 9/18
> search "office hours"
[10/1/2019] office hours (1.0 h):
Total: 1.0 h
> q                     you can type 'q' to stop the program
```

Implementation Details:

Below is the list of the classes and methods you need to implement. **You must provide implementations of all the methods below matching these exact signatures.** These classes may have whichever fields you choose.

You cannot use any Java data structures besides arrays, but feel free to implement additional structures that have been covered in lecture. Any non-test classes that you create should be placed into the `main` package.

You should not be importing any Java-provided libraries.
This includes `java.util.Arrays`.

WorkEntrySearchNode.java

This class provides the functionality of a splay tree node discussed in class (adding, searching, deleting, etc.). The class starts with three fields: a left child pointer, a right child pointer, and parent pointer that are **all public**. **You must keep these fields as public.** The 3-pointer tree node representation follows from lecture, so you should implement the node's methods accordingly. In addition to these fields, you are **required** to have the following methods. You are encouraged to add additional fields and helper methods.

`public WorkEntrySearchNode(String activity)` – constructs a node with a specified activity key. This WorkEntrySearchNode will store a collection of WorkEntry objects that have this specified activity key.

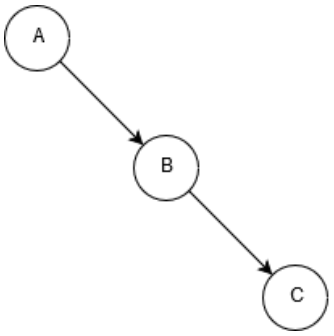
`public int compareTo(WorkEntrySearchNode other)` – this compares two nodes based on their activity keys.

`public WorkEntrySearchNode search(WorkEntrySearchNode node)` - this should search for a node in the splay tree rooted at this node and return the node that matches or the last node encountered in the search. The node that you return should be splayed to the root.

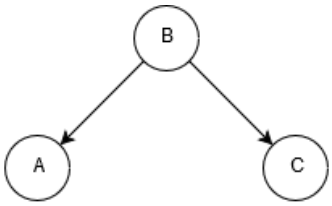
`public WorkEntrySearchNode insert(WorkEntrySearchNode node)` - this should insert a node into the splay tree rooted at this node and return what gets inserted or a node already in the tree that matches this key. The node that you return should be splayed to the root.

`public String toString()` - this should return an inorder traversal of the tree rooted at this node with the nodes' keys **on separate lines.**

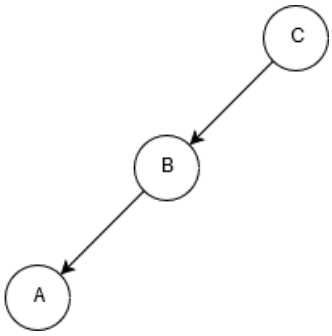
`public String getStructure()` - this should return a String representation using parentheses to display the structure of the tree as described in the examples below.



`(A(B(C)))`



`((A)B(C))`



`(((A)B)C)`

Parenthesis should indicate the left and right subtrees of a given node. The leftmost example demonstrates that A has a right subtree of B which has its own right subtree of C. Similarly, the rightmost example demonstrates that C has a left subtree of B which has its own left subtree of

A. The middle example demonstrates that B has 2 subtrees A and C. **Note that this returns a String that is all on the same line.**

`public void add(WorkEntry e)` - this should add a `WorkEntry` to this node. You should only be passing in WorkEntry objects to this method with a matching activity. Entries should be stored in the same order in which they were added from the input CSV file. This can be seen in the **Example Output** by comparing the input table and the search for "office hours".

`public WorkEntrySearchNode del(int i)` - this should remove the ith `WorkEntry` stored in this node. If the removal results in the node containing no more entries, the node should be removed from the tree it's contained in based on the following guidelines:

Suppose we want to delete node v which is the root.

- If v has a left subtree, splay the maximum of v.left and then attach v.right to the splayed node. Return the splayed node to be the new root of the tree.
- If v does not have a left subtree, return v.right
- If v is a leaf, return null.

If the ith entry does not exist, you **must throw an IndexOutOfBoundsException**.
If the removal of the ith `WorkEntry` stored in this node results in some `WorkEntry` objects remaining in this node, you should return this node.

`public String getEntryData()` – this should return the String representations of the `WorkEntry` objects associated with this node on separate lines followed by the total time spent on these entries **rounded to one decimal place:** this will always be either **.0** or **.5**. If there are no `WorkEntry` objects associated with this node, return the empty String.

`public String getByRecent()` – when a search or insertion is done, nodes similar to the search will be splayed upward in the tree. Therefore, a level order traversal of the tree will list nodes in an order based on their similarity to the most recent search or insertion. This method will return a level order traversal of the tree where each node's key is displayed **on a separate line**. **Do not put nodes on the same tree level on the same String line**. To implement this method, *you will be writing* a First-In-First-Out queue which will be discussed shortly.

Note: You may assume that all of the above methods with the exception of `compareTo()` will be called on the root node of a splay tree.

<u>Recommended</u> methods:

Below are some additional helper methods that we recommend including in your implementation of this class. These will allow you to condense your code for use in the implementation of the required methods of the class that require splaying (e.g. insert, search, del).

`private void splay()` – This method will splay this node to the root of the tree in which it is contained. When implementing this method, make sure to consider the splaying cases discussed in class (zig-zig, zig-zag).

`private void rotateLeft()` – This method performs a single AVL left rotation to this node.

`private void rotateRight()` – This method performs a single AVL right rotation to this node.

Queue.java

This class will hold your implementation of a **generic** FIFO queue that you will use in your implementation of `getByRecent()` in WorkEntrySearchNode.java. You **must** implement the following methods.

`public Queue()` – constructs an empty queue

`public void enqueue(T data)` – enqueues a data element at the end of the queue.

`public T dequeue()` – dequeues the data element at the front of the queue and returns it. If there are no elements in the queue, return null.

`public T front()` – gets the data at the front of the queue (i.e. the one that would be dequeued next). If there are no elements in the queue, return null.

`public int size()` – return the number of elements in the queue.

WorkTimeAnalysisTool.java

This class provides the "command line" functionality that parses user inputs and performs operations on the data provided in the CSV file by wrapping the data structures you create. The following methods **must** be implemented, but feel free to add additional helper methods.

`public WorkTimeAnalysisTool(WorkEntry[] entries)` - this constructs the tool given an array of `WorkEntry` objects.

`public String parse(String cmd)` – this will parse a given command. If the command is a "list" or "search" command, then it should return the result of the list or search. If the command is "del", then this should return null.

Design Strategy:

Here is the approach we recommend that you take when completing this assignment. You do not need to take this approach, but we *strongly encourage* that you at least consider the questions posed below.

1. Implement the `compareTo()`, `search()`, `insert()`, `toString()`, and `getStructure()` methods of `WorkEntrySearchNode`.
   - How will you implement getStructure() and toString()? (*Hint:* You may want to use recursive helper methods and build a String in a StringBuilder object.)

2. Test that your tree is splaying properly thoroughly by utilizing `getStructure()` in `StudentTreeTest.java`

3. Implement the `add()`, `del()`, and `getEntryData()` methods of `WorkEntrySearchNode`.
   - What will you be storing in splay tree nodes?
   - Will a node need additional data structures?

4. Implement the `getByRecent()` method of `WorkEntrySearchNode`. To do this, you will need to implement all the required methods in the `Queue` class.

5. Implement the `parse()` method and fields of `WorkTimeAnalysisTool`.
   - What do you know about the nodes that are returned from the search() and insert() methods of WorkEntrySearchNode? (*Hint:* These are splay nodes, how can you use them?)
   - Which method of WorkEntrySearchNode will you use to implement 'list l'? Which method will you use to implement 'list r'?
   - In which cases will 'del x' throw an IllegalStateException? You should consider cases where there would be no entries to index. (*Hint:* This will depend on previous operations.)

JUnit Tests:

You must write JUnit tests for your implementation of WorkEntrySearchNode. Please write your JUnit tests in the provided file **StudentTreeTest.java** located in the tests package.

You have also been provided with ExampleToolTest.java. This file **should not be edited** because it provides a JUnit test that makes sure your code produces the same output for the inputs given in the example output above. It will help you ensure that you are using the exact Strings that are needed to implement various methods. **If you don't pass these tests, you will most likely fail the grading tests.**

Provided Files:

You have been provided with some files in the main package that you **should not edit.**

- **REPL.java** – this class loads the user's CSV file into a WorkTimeAnalysisTool object and then passes user input to the tool to be parsed using the parse() method.
- **InputFileReader.java** – this is a utility class used by REPL to read a CSV file of entries.
- **WorkEntry.java** - This class holds the information on a work entry *that you should use in the implementation of the assignment*. A work entry is equivalent to one row of a table

like the one displayed in the **Example Output** section. This class is fully documented in the skeleton code and should be read before use in your other classes.

Submission Details:

- For every non-test method you write, you must provide a **proper JavaDoc** comment using `@param`, `@return`, etc. that also **includes the method's runtime.**
- At the top of **every file** that you submit (including tests) please leave a comment with the following format. Make sure to include **each of these 6 items.**

```
/**
 * <A description of the class>
 * Known Bugs: <Explanation of known bugs or "None">
 *
 * @author Firstname Lastname
 * <your Brandeis email>
 * <Month Date, Year>
 * COSI 21A PA2
 */

>Start of your class here<
```

- You must use the skeleton Eclipse project.
- It must be named **LastnameFirstname-PA2.** (Use Eclipse's Refactor > Rename)
- Submit a .zip file of your Eclipse project named **LastnameFirstname-PA2.zip**
- Late submissions will not receive credit.
- **Do not use any Java provided data structures other than arrays.**
- **Do not import any Java libraries.**

Authors

This assignment was written by Chami Lamelas.