The University of Saskatchewan

Saskatoon, Canada

Department of Computer Science

CMPT 280– Intermediate Data Structures and Algorithms

Assignment 5

Date Due: March 3, 2020, 9:00pm

Total Marks: 20

1 Submission Instructions

- Assignments must be submitted using Moodle.
- Responses to written (non-programming) questions must be submitted in a PDF file, plain text file (`.txt`), Rich Text file (`.rtf`), or MS Word's `.doc` or `.docx` files. Digital images of handwritten pages are also acceptable, provided that they are **clearly** legible.
- Programs must be written in Java.
- If you are using IntelliJ (or similar development environment), do not submit the Module (project). Hand in only those files identified in Section 5. Export your `.java` source files from the workspace and submit only the `.java` files. **Compressed archives are not acceptable.**
- No late assignments will be accepted. See the course syllabus for the full late assignment policy for this class.

2 Background

2.1 Hash Tables

In this question you will work with an implementation of a *chained hash table*. `lib280-asn5` contains the class `KeyedChainedHashTable280`. You will use this class to solve a problem. `KeyedChainedHashTable280` is an example of a *keyed dictionary* where each item is associated with a unique *key*. A keyed hash table computes the hash of an item from **only** the item's key, even though the item itself might contain other data. Additional background and details about chained hash tables are part of this week's lectures and are a component of this week's tutorials.

3 Your Tasks

Question 1 (20 points):

Almost every modern video game in the roleplaying genre provides the player with a *quest log* which is essentially a list of tasks that the player's character has to perform to advance the story or to obtain rewards.[1] In this question we are going to create a data structure for a quest log based on a chained hash table. For our purposes, a *quest log entry* will consist of the following pieces of information:

- Name of the quest.
- Name of the area of the world in which the quest takes place.
- Recommended minimum character level that should attempt the quest.
- Recommended maximum character level that should attempt the quest.

For the purposes identifying quests, **quest log entries are keyed on the name of the quest.** A class called `QuestLogEntry` that can hold these pieces of information is provided. Notice how the `key()` method of the `QuestLogEntry` class returns the name of the quest.

For this question, you are provided with a complete IntelliJ module called `QuestLog-Template` in which you will modify one of the classes. It includes the `QuestLogEntry` class and some `.csv` (comma-separated value) files which will be used as input data. The `QuestLog-Template` module requires access to the `lib280-asn5` project, set this up as a module dependency as per the self-guided tutorial on the class website for setting up an IntelliJ module to use `lib280`. You've already done this sort of thing previously with other assignments.

The entirety of your work will be to finish implementing methods in the `QuestLog` class provided in the `QuestLog-Template` project, and to write a couple of interesting tests. Note that the `QuestLog` class is a specialized extension of `KeyedChainedHashTable280`, so it is a chained hash table. Here is a list of what you have to do:

(a) Complete the implementation of the `QuestLog.keys()` method. This method must return an array of the keys (i.e. the quest names) of each `QuestLogEntry` instance in the hash table. The keys may appear in the returned array in any order.

(b) Complete the implementation of the `QuestLog.toString()` method. This method should return a printable string consisting of the complete contents of all of the `QuestLogEntry` objects in the hash table *in alphabetical order by the quest names*, one per line. Here is an example string returned by `toString()` from a quest log containing four entries:

[1] Back in '80s and early '90s, games didn't have quest logs. If you wanted to remember what you were supposed to do, or something that a character in the game said, you **wrote it down** with an ancient mystical device called a **pencil**. And there was no Internet with detailed wikis for every game to get hints from if you got stuck!

```
Defeat Goliad : Candy Kingdom, Level Range: 20-25
Locate the Lich's Lair : Costal Wasteland, Level Range: 35-40
Make an Amazing Sandwich : Finn's Treehouse, Level Range: 1-5
Win Wizard Battle : Wizard Battle Arena, Level Range: 2-4
```

Remember that the hash table makes no promises whatsoever about the ordering of the quest log entries in the chains of the hash table. Hint: the `keys()` method from part (a) will be handy for this method, as will knowing that `Arrays.sort()` can sort the elements of an array.

(c) Complete the implementation of the `QuestLog.obtainWithCount()` method. This method takes a quest name as input and returns a `Pair280` object (found in `lib280.base`) which must contain the `QuestLogEntry` object from the quest log which matches the given quest name (if it exists) and the number of `QuestLogEntry` objects that were examined while searching for the desired one. The latter number must be present whether the quest name was found in the quest log or not.

Hints: A `Pair280` object has two generic type parameters, the first of which specifies the type of the first element of the pair, and the second of which specifies the type of the second element in the pair. For example, if I wanted a pair consisting of an `Integer` and a `Float` I might write:

```
Pair280<Integer, Float> p = new Pair280<Integer, Float>( 5, 42.0 );
```

The components of the pair can be accessed using the `firstItem()` and `secondItem()` methods:

```
System.out.println(p.firstItem());    // prints the integer 5
System.out.println(p.secondItem());   // the floating point number 42.0
```

(d) Now take a look at the `main()` program in `QuestLog.java`. As given, it already does the following things:

1. Creates a new, empty `QuestLog` instance called `hashQuestLog`.
2. Creates a new, empty `OrderedSimpleTree280` instance (a binary search tree) called `treeQuestLog` that can hold items of type `QuestLogEntry`.
3. A `.csv` file containing the data for a number of `QuestLogEntry` objects is opened, and read in, creating a `QuestLogEntry` instance for each quest, and adding each such instance to both both the `hashQuestLog` and `treeQuestLog` data structures.
4. The complete contents of `hashQuestLog` and `treeQuestLog` are printed out using their respective `toString()` methods. You'll know when your `Questlog.toString()` method from part (b) is working when its output matches that of `treeQuestLog.toStringInorder()`.

At the end of `main()` are two TODO markers. For the first one, you need to write code that calls `hashQuestLog.obtainWithCount()` for each quest in the hashed quest log and determines the average number of `QuestLogEntry` objects that were examined over all such calls.

Finally, for the second TODO marker, you have to do the same thing for `treeQuestLog`. You can do this by calling the `searchCount()` method of `treeQuestLog` for each quest stored in the log. Note that `searchCount()` requires that you pass in the actual `QuestLogEntry` object that you are looking for rather than just the quest name (you can obtain these from the hashed quest log). `searchCount()` returns the number of items that were examined while trying to position the tree's cursor at the given `QuestLogEntry` object.

Once you have computed the average number of `QuestLogEntry` objects examined for each of the two data structures, print out the results. Something like this will do:

```
Avg. # of items examined per query in the hashed quest log with 4 entries: 1.25
Avg. # of items examined per query in the tree quest log with 4 entries: 2.0
```

(e) Run your completed `main()` program for each of the `.csv` files provided in the project (just change the filename in quotes that is passed to `FileReader`). Each `.csv` file contains in its filename the

number of quest entries in the file. For each `.csv` file, record the reported average number of items examined per query in each data structure. In a text file called `a5q1.txt` (or other acceptable file format) answer the following questions:

1. List the reported averages that you recorded for each .csv input file in a table that looks something like this (filling in the rest of the table of course):

Filename	Avg. Queries for `hashQuestLog`	Avg. Queries for `treeQuestLog`
quests4.csv	1.25	2.0
quests16.csv		
quests250.csv		
quests1000.csv		
quests100000.csv		

2. If you had to choose a simple function (i.e. from the list of functions used in big-*O* notation) to characterize the behaviour of the the average number of items examined per query for the **hashed** quest log as the number of quests (n) in the log increases, what would it be?
3. If you had to choose a simple function (i.e. from the list of functions used in big-*O* notation) to characterize the behaviour of the the average number of items examined per query for the **tree** quest log as the number of quests (n) in the log increases, what would it be?
4. If your primary use of the quest log was to display all of the quests in the log in alphabetical order, would you prefer the hashed quest log or the tree quest log? Why?
5. If your primary use of the quest log was to periodically look up the details of specific quests in no particular order, would you prefer the hashed quest log or the tree quest log? Why?

4 Files Provided

QuestLog-Template.zip: A complete IntelliJ module containing everything you need for Question 1.

5 What to Hand In

QuestLog.java: Your completed quest log based on a hash table (parts (a), (b), and (c) of Q1), and completed additions to `main()` (part (d) of Q1).

a5q1.txt: Your answers to the questions posed in part (e) of Q1.