Project 5

Due Date: Wednesday 13 April by 11:59 PM

General Guidelines.

The method signatures provided in the skeleton code indicate the required methods. You may need additional methods or classes that will not be directly tested but may be necessary to complete the assignment, and you are welcome to add those into the classes.

Unless otherwise stated in this handout, you are welcome to add to/alter any provided java files as well as create new java files as needed, but make sure that your code works with the provided test cases. Your solution must be coded in Java. Also, keep in mind that your solution should not depend on any changes you make to any files that are not submitted. For example, the test code is not submitted as we use our own test code to grade your project, so if your solution depends on changes you made to the test code, then it will not pass our test code. Again: Only changes made to submitted files will be reflected in the grading, and we do not want you to submit anything that is not specifically mentioned in the submission guidelines.

In general, you are not allowed to import any additional classes in your code without explicit permission from your instructor! The Math class is fine for import.

Note on academic dishonesty: Please note that it is considered academic dishonesty to read anyone else's solution code, whether it is another student's code, code from a textbook, or something you found online. You MUST do your own work! It is also considered academic dishonesty to share your code with another student. Anyone who is found to have violated this policy will be subject to consequences according to the syllabus and university policy.

Note on grading and provided tests: The provided tests are to help you as you implement the project and (in the case of auto-graded sections) to give you an idea of the number of points you are likely to receive. Please note that the points indicated when you run these tests locally are not your final grade. Your solution will be graded by the TAs after you submit. Please also note that these test cases are not likely to be exhaustive and find every possible error. Part of programming is learning to test and debug your own code, so if something goes wrong, we can help guide you in the debugging process but we will not debug your code for you.

Project Overview.

This project has three parts. In each of the parts, you must implement a data structure. More specifically, you are provided skeleton code for each one and must implement specific operations for the data structures.

Part 1. Hashtable (20 points)

In the file called `Hashtable.java`, you must complete the implementation of a Hashtable that uses linear probing to handle collisions. Please pay close attention to the specific guidelines for resizing and hashing so that you can pass the provided tests.

Guidelines

- First, familiarize yourself with the Pair.java class that is provided for you. This is a general class for key-value pairs that will be used throughout the project.
- Second, familiarize yourself with the Hashtable.java class that is provided. Much of the basic set up is done for you. You just need to fill in the code for the operations.
- I recommend you have a private method for finding the hash value for a key. You should just use the `hashcode` function (which exists for all Java objects) along with modular hashing.
- I recommend having a private method for resizing the table. Make sure that you skip over deleted items when you resize, and don't forget to rehash all the elements.
- Instead of requiring the table sizes to be prime (which would necessitate a certain amount of overhead), I have provided a simple function to get a number that is somewhat likely (but not guaranteed) to be prime. Use this when you resize.
- You will have to figure out how you want to manage deleted items. I suggest either denoting deleted items in the Pair class (which you can change as needed) or keeping a boolean array to keep track of what has been deleted.
- Although I've tried to simplify the generics for you by providing the skeleton code, you will still need to do a little casting. The generic types will be treated as Objects, which will need to be cast to whatever the actual type is before you can do anything useful with them.

Required Methods to be Implemented

Method Signature	**Description**
`public V get(K key)`	`return the value associated with`

	the given key or null if the key does not exist in the table
public void put(K key, V val)	Insert the (key, value) pair into the table or, if the key already exists in the table, update the value. If, after the insert, n/m* exceeds alphaHigh (0.5), then resize the table to getNextNum(2*m).
public V delete(K key)	Delete the key and associated value from the table. Don't forget that you don't want gaps in your clusters, so be careful how you do this. Return the deleted value OR null if the key was not in the table. After the delete, if m/2 >= 11 AND n/m* < alphaLow (0.125), then resize the table to getNextNum(m/2).
public boolean isEmpty()	Return true if the table is empty and false otherwise. This function should be very easy to implement.
public int size()	Return the number of key-value pairs in the table. This function should be very easy to implement.

*Don't forget that since n and m are integers, you will need to cast this load factor to a double to get accurate results.

Part 1 Grading.
The tests total 20 points. You can see what the tests are testing for in the test code and in the output.

Part 2. Priority Queue (20 points)

In the file called `MinPQ.java`, implement the operations for a min priority queue.

Guidelines

- Familiarize yourself with the provided skeleton code.

- Note that the item that is being put into the PQ is generic, but it must implement Comparable. That means that you can assume that there is a `compareTo` method for that class that can be used for ranking. See https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html for details.
- For the `sink` function: if the two children of a node are equal and the node in question is larger than the children, swap with the left child (not the right child).
- Make sure your `delMin` and `getMin` functions throw the `EmptyQueueException` when appropriate.
- Pay attention to the details for resizing.
- You should write helper functions to do `sink`, `swim`, `swap`, and `resize`.
- Base your indexing at 0. That is, the minimum should be at index 0.

Required Methods to be Implemented

Method Signature	**Description**
`public void insert(T item)`	`Insert the item into the PQ. If the array is full before the insert, resize it to be twice as large.`
`public T delMin() throws EmptyQueueException`	`Delete and return the minimum from the PQ. If the PQ is empty when the function is called, throw the exception. If the PQ's size drops below ¼ of the array capacity after the delete, resize the array to be ½ its size.`
`public T getMin() throws EmptyQueueException`	`Return but do not delete the minimum from the PQ. If the PQ is empty when the function is called, throw the exception.`
`public int size()`	`Return the number of items in the PQ. This should be very simple.`
`public boolean isEmpty()`	`Return true if the PQ is empty and false otherwise. This should be very simple.`

Part 2 Grading

The tests total 20 points. You can see what the tests are testing for in the test code and in the output.

Part 3. Search Tree

In the file called `Tree.java`, implement a search tree. You have two options. You can implement a binary search tree, or you can implement a *right-leaning* red-black tree. In both cases, the required methods to be implemented are the same, but their implementations will be different because the structures are different. And (as you probably guessed), option 2 is significantly more difficult, which is why that option can earn you extra credit.

Option 1.

If you correctly implement the functions for the binary search tree, you can get full credit, but your implementations must be correct and efficient. The BST tests are worth 5 points each (so 20 points total).

Option 2.

If you correctly implement the functions for the right-leaning red-black tree, you can get full credit + 12 extra points, but our implementations must be correct and efficient. The RBT tests are worth 8 points each (so 32 points total).

Guidelines

- Both trees should maintain BST order as discussed in class. Note that the K generic type extends Comparable, so you can use `compareTo`.
- In both cases, you need to be able to efficiently get the *height* and *size* of individual nodes in the tree, as well as the *height* and *size* of the tree itself. It is not a very efficient method to calculate this each time by searching the tree, and such a method will not be given full credit. Note that the Node class provided already has variables for *height* and for *N*. The key is to update these as you do the inserts.
- For the right-leaning red-black trees:
    - A right-leaning red-black tree is like a left-leaning red-black tree, but all the red links must go to the right instead of the left.
    - You may use the code provided in the slides. You will need the same basic transformations for this tree.
    - The first thing you should do is look at the cases for left-leaning inserts (Cases 1-4 in the slides) and change them to fit a right-leaning tree. (Think of this tree as a mirror image of the left-leaning tree.)
    - The second thing you should do is make sure you fully understand the code for the transformations. Like I said, you don't need new code for the

rotations or the color flip, but you will have a very difficult time if you don't actually understand what the code is doing, so trace through it first.

Required Methods to be Implemented

Method Signature	Description
`public void put(K key, V val)`	Insert (key, val) into the tree or update val if key is already in the tree.
`public V get(K key)`	Get the value associated with key in the tree or return null if the key does not exist.
`public boolean isEmpty()`	Return true if the tree is empty or false otherwise. Should be simple.
`public int size()`	Return the number of nodes in the tree. Should be simple.
`public int height()`	Return the height of the tree. This should take O(1) time.
`public int height(K key)`	Return the height of the subtree whose root node contains the given key. If the key does not exist in the tree, return -1. The time for this should be the same as the time it takes to find the node.
`public boolean contains(K key)`	Return true if the key is in the tree and false otherwise.
`public int size(K key)`	Return the size of the subtree whose root contains the given key. Return -1 if the key does not exist. The size should include the node itself. Also, the time for this should not be worse than the time it takes to find the node.

Part 3 Grading

The tests total 20-32 points depending on the option you choose, but the testing method here is different. You are provided with input and output files. The input files contain commands and the output files contain the results of running those commands. The test code does all this for you, but the way your code is evaluated is by comparing the output of your code with the expected output by looking at the difference between the two files.

- For debugging purposes, test 0 is the easiest one and can easily be drawn by hand (even as a RBT).
- Both versions use the same input files, but the output files are different.
- To test a BST implementation: type the following command into the terminal:

```
./runBSTTests.sh
```

- If your output looks like this, it is correct:

```
Running Test 1...
Running Test 2...
Running Test 3...
Running Test 4...
```

- Incorrect output will result in other things being printed out. Those things indicate differences between the expected output file and your output file. You should be able to compare them manually. Your output for a given test will be in a file called `output<testNum>.txt`. (For example, `output0.txt` for Test 0.) The expected output will be in a file called `test_bst_output<testNum>.txt.` (For example, `test_bst_output0.txt` for Test 0.)
- To test an RBT implementation, the only differences are:
    - The command to run the tests is: `./runRBTTests.sh`
    - The expected output will be in a file called `test_rbt_output<testNum>.txt` (e.g. `test_rbt_output0.txt` for Test 0).

Other notes about grading for the whole project:

- If you do not follow the directions (e.g. importing classes without permission, using unauthorized extra space, etc.), you may not receive credit for that part of the assignment.
- Good Coding Style includes: using good indentation, using meaningful variable names, using informative comments, breaking out reusable functions instead of repeating them, etc.
- If you implement something correctly but in an inefficient way, you may not receive full credit.

- In cases where efficiency is determined by counting something like array accesses or grid accesses, it is considered academic dishonesty to *intentionally* try to avoid getting more access counts without actually improving the efficiency of the solution, so if you are in doubt, it is always best to ask. If you are asking, then we tend to assume that you are trying to do the project correctly.
- If you have questions about your graded project, you may contact the TAs and set up a meeting to discuss your grade with them in person. Regrades on programs that do not work will only be allowed under limited circumstances, usually with a standard 20-point deduction. All regrade requests should be submitted according to the guidelines in the syllabus and within the allotted time frame.

Submission

To submit your code, please upload the following files to **lectura** and use the **turnin** command below. Once you log in to lectura and transfer your files, you can submit using the following command:

```
turnin csc345p5 Hashtable.java Pair.java MinPQ.java Tree.java
```

Upon successful submission, you will see this message:

```
Turning in:
      Hashtable.java -- ok
      Pair.java-- ok
      MinPQ.java-- ok
      Tree.java – ok
All done.
```

Testing your code on lectura.

It is always a good idea to compile and run your code on lectura before you submit. In order to do that, you need to transfer all files to lectura that are necessary for testing (but only *submit* the ones that are required for submission.) Once the files are submitted, you can use the following commands in the terminal as needed.

`javac <name of file to be compiled>` – this will compile the given file
`javac *.java` – this will compile all .java files in the current folder
`java <name of file to be run>` – this will run the compiled file

For testing Part 3, follow the instructions above for running the tests.