CS 6301.001. Implementation of data structures and algorithms
Long Project 3: Skip Lists vs RBT

Ver 1.0: Initial description (Monday, March 9).
Ver 1.1: Some minor corrections (Wednesday, March 11)
Due: 11:59 PM, Sun, April 5.

Max excellence credits: 3.0

- Submission procedure is same as the same as that of prior projects.
- For each group, only its last submission is kept and earlier submissions are discarded.
- Your code must be of good quality, well commented, and pass all test cases within time limits to earn excellence credits.

Project Description

In this long project, implement skip lists and red-black tree, and then perform comparison study of both the data structures over a large number (million) of add, contain, remove operations.

1. Skip Lists

Implement the following operations of skip lists. Some are optional. Starter code is provided. Do not change the signatures of public methods declared. You can add additional fields, nested classes, and methods as needed. Driver code is also provided along with the several testcases.

```
add(x): Add a new element x to the list. If x already exists in the skip
list, replace it and return false.  Otherwise, insert x into the skip
list and return true.

contains(x): Does list contain x?

remove(x): Remove x from the list. If successful, removed element is returned.
Otherwise, return null.

size(): Return the number of elements in the list.

isEmpty(): Is the list empty?


ceiling(x):  Find  smallest  element  that  is  greater  or  equal  to  x.
(optional)

first(): Return first element of list. (optional)
```

floor(x): Find largest element that is less than or equal to x. (optional)

get(n): Return element at index n of list. First element is at index 0. Call either getLinear or getLog. (optional)

getLinear(n): O(n) algorithm for get(n). (optional)

getLog(n): O(log n) expected time algorithm for get(n). **This method is optional, but code it correctly to earn an EC.** Need to implement get(n) to if you implement this.

iterator(): Iterator for going through the elements of list in sorted order. (optional)

last(): Return last element of list. (optional)

rebuild(): Reorganize the elements of the list into a perfect skip list. A search operation in a perfect skip list, will emulate binary search in a sorted array. (optional)

2. Red-black Tree

Extend BST to Red-Black Tree. Implement add and remove operations. You need to implement the algorithms discussed in the class. If you implement any other algorithm, you will not get any credit. Do not declare field for parent in a node. Also, as discussed in the class, just create one nil black node. If you create a new nil node for every leaf node, your space usage will double. Also, implement method verifyRBT() to verify whether the tree satisfies all the properties of RB tree. Starter code (RedBlackTree.java) is provided. Driver code will be provided. Some of the test cases given for skip lists can be used to test your code for RB tree.

Implement RBT efficiently. The performance of your implementation will be compared with other teams' implementation and **excellence credits will be awarded based on the relative performance.** The rubric for awarding EC is as follows:
Top 5 performing teams will get 1 EC
Top 6 to 10 performing teams will receive 0.75 EC
Top 11 to 15 performing teams will receive 0.5 EC

3. Comparison Study

Do a comparison study of the performances of your implementation of skip list, your implementation of RB tree and java's TreeSet over a large number of add/contain/remove operations. Do the study over large tree sizes such as 4M, 16M, 64M, 256M till you run out of memory. You can also use any of the net01 through net45.utdallas.edu machines to run your experiments. Write a good report on your comparison study and the results. **An excellent comparison study with a good report will earn you an EC.**