CMPT 225, Spring 2018, Assignment 5

Due date: March 27, 2018, 5:30 PM (before class)

1. Programming exercise. Implement a method that, given an input binary tree T with n nodes, performs a sequence of up to n rotations (which can be left or right rotations) to transform T into a caterpillar (a tree for which every left child is a leaf). **(25 pts.)**

   Details and suggestions:

   Start by implementing a *Position* class that supports the following methods: *left* (left child), *right* (right child), *parent* (parent position), *isRoot*, *isLeaf* (see Lecture 6).

   Implement a *BinaryTree* class that contains Positions as its nodes (see Lecture 6).

   Implement a *Rotation* method that takes as input a tree node and a direction (left or right) and outputs the rotated tree.

   This exercise is easier to do in the linked-list implementation of a binary tree than in an array-based one, but you can decide which one you prefer.

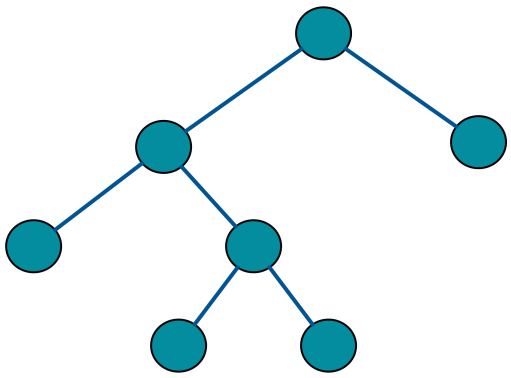
   You may find it useful to convert T to an intermediate (canonical) form C, for instance, a caterpillar (see Lecture 6).

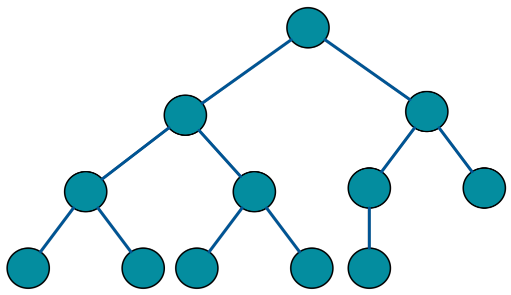
   You do not need to worry about any data in your tree, only the structure matters.

   In your main function, demonstrate the process on the trees given in the test cases.

   Bonus: Find the **shortest** sequence of rotations to transform T into a caterpillar.

   Test cases (create these trees and demonstrate that your program works on them):





2. Programming exercise. Implement a priority queue class based on a sorted array. **(20 pts.)**

   Details and suggestions:

   You may assume that the priority queue (and hence, array) size is at most 100.

   You may assume that the keys are integers and the values are single characters. I suggest creating a class called *Pair* that contains an int as key and a char as value.

   Your priority queue needs to keep track of the last position used in the array.

   Provide an implementation of your algorithm in a file called SortedArrayPQ.cpp.

   The main function should demonstrate the functionality on several test cases.

   Test case: perform the following operations, showing the state of the priority queue each time.

   insert(5,a)

   insert(4,b)

   insert(7,i)

   insert(1,d)

   removeMin()

   insert(3,j)

   insert(6,c)

   removeMin()

   removeMin()

   insert(8,g)

   removeMin()

   insert(2,h)

   removeMin()

   removeMin()

3. Your classmate claims that the order in which a fixed set of entries is inserted into an AVL tree does not matter - the same AVL tree results every time. Give a small example that proves he or she is wrong. **(10 pts.)**

4. Explain how to use a red-black tree to sort n comparable elements in $O(n \log n)$ time in the worst case. **(10 pts.)**

5. Show that, given only the less-than operator ($<$) and the Boolean operators *and* (&&) and *not* (!), it is possible to implement all the other comparators: $>$, $<=$, $>=$, $==$, $!=$. Suggestion: call the elements to be compared x and y. **(5 pts.)**

6. Let H be a heap storing 15 entries using the vector representation of a complete binary tree (recall that in this representation, the root is stored at index 1 and the left and right children of the node at index i are stored at indices $2i$ and $2i + 1$ respectively). What is the sequence of indices of the vector that are visited in a preorder traversal of H? What about an inorder traversal of H? What about a postorder traversal of H? **(10 pts.)**

7. Illustrate the performance of the heap-sort algorithm on the input sequence (2,5,16,4,10,23,39) by showing the state of the heap after each insert and removeMin operation and marking the path used for each up/downheap operation. **(10 pts.)**

8. **Bonus:** In the previous question, use the bottom-up construction to build the heap. **(5 pts.)**