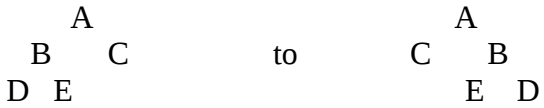
Programming Project 6 - Due: **Tuesday, June 7** at **6:45 PM**

You have learned so much powerful stuff these last few weeks, you can't wait to put all that power to the test. Before that, you're not really convinced those recursive methods operating on trees really work, so you plan to write one for yourself and see.

Part 1:

You will implement a recursive method that changes a binary tree (any binary tree; not necessarily balanced, not necessarily a BST) as if it was flipped left to right.

As an example, the transformation below would be performed when the tree is flipped.



Hint: the code is minimal but some thinking is required. As always, best done on paper first. To test your code you can make up a tree by manually linking nodes. Then flip it and verify the result (you can do a tree traversal and print each node).

Part 2:

You will implement a min-max queue class which has two pop methods. PopMin removes and returns the min priority item in the queue, while popMax removes the max priority one. e.g. If we push the following names in any order (“bob”, “alice”, “zalice”, “mary”) then popMin would return “alice” while popMax would return “zalice”.

In the implementation of your class you may use **as a client** any of the Java library classes (or combination of) we have seen.

Part 3:

You will write a program that, given a word, finds other valid English words that have exactly the same letters. The program relies on an English dictionary to verify whether or not a string is a valid English word.

a) You plan to make a set of all the words in the dictionary (all converted to lowercase). Then, given a word, you plan to make “words” from all the combinations of the letters and check whether each is in the set. If yes, it is a valid English word and you add it to the result. For example, for the word “cat”, the combinations are “act”, “atc”, “cat”, “cta”, “tac”, “tca”. “act” is in the dictionary and thus valid.

Analyze the runtime of this algorithm and give the big O. Write your brief and clear answer in the report. **No code to write at all.**

b) After some thinking you realize that if you could somehow group dictionary words having the same letters into separate groups, then you could find all the matching words for a given word by only searching through that word's group. This approach reminds you very much of the way a hash table using separate chaining is implemented. **The only Java API data structures you can use are arrays and java.util.LinkedList.**

Hints:

- Make sure you understand the problem and your approach really well before you write even one line of code. Start by looking at the given test code. The algorithm and code you will write is not complicated, but can get complicated really fast if you are not clear about what you are doing. Make sure you understand the concept of hashing and how separate chaining was implemented.
- A common problem when performing a mod operation (e.g. h % size) is when h is negative. You can handle negative results by adding size.
- Test code has been given to you. Start first with some simple tests and, as you gain confidence that your code works, run all the functional tests and then the stress tests to ensure your solution is efficient.

Part 4:

In the report file you will do a brief (2 lines each) complexity analysis of Parts 1 through 3 a) and give the big O for each.

Deliverables:

You should submit a zip file named project6_first_last.zip (where first and last are your first and last name) containing **ONLY the files below.**

Project6.java

report.txt : a text (not Word, Power Point, ...) file containing ONLY:
a) A 1 to 5 lines paragraph from you saying "I have tested this program and there are no known issues." if you believe that to be the case, or a brief description of known issues in case your program has known problems or you could not fully implement it.
b) The complexity analysis for Parts 1 to 3 a)

How you get points:

– Part 1	15 points
– Part 2	20 points
– Part 3	a) 10 points
	b) 55 points
– Part 4	included in 1 to 3 above

How you lose points:

- If you do not follow the given directions and decide to make changes "for fun". Specifically, **do not change the code given to you**. **Just fill in the missing parts.**
- If your implementation is inefficient. **Your solution should be efficient** to a level seen in class for this ADT.
- If your code does not follow good coding practices.
- If any of your code prints anything at all on the console. Remove all your print outs, debug statements, etc. Clean up your code and do not leave clutter behind.
- If your code has no comments where needed. Comment your code appropriately. Brief and to the point.
- If you submit your whole workspace or executable files. Submit only the files the project asks for.