York University
EECS 2011 Winter 2018 – Problem Set 2
Instructors: James Elder, Suprakash Datta

This problem set will not be graded, but can be used to prepare for the midterm. You are free to work together on these if you prefer. Solutions will be posted Tuesday, Feb 20th.

1. Choosing a data structure

   State in one or two words the simplest ADT and implementation we have discussed that would meet each requirement.

   (a) O(1) time removal of the most recently added element
   ADT: Implementation:

   (b) O(1) average time addition, removal, access and modification of (key, value) pairs with unique keys
   ADT: Implementation:

   (c) O(1) time insertion and removal when you are given the position
   ADT: Implementation:

   (d) O(1) time index-based access and modification and amortized O(1) addition of elements
   ADT: Implementation:

   (e) O(log n) time insertion of (key, value) entries and O(log n) removal of entry with smallest key
   ADT: Implementation:

   (f) O(1) time removal of the least recently added element
   ADT: Implementation:

2. Binary Trees

   You are to design a recursive algorithm **btDepths(u, d)**, where u is a node of a binary tree and d is the depth of u. Your algorithm will determine the minimum and maximum depths of the external nodes descending from u. Note that if u has no parent (i.e., is the root of the whole tree), then $d = 0$. You can assume that each node v of the tree supports the following four binary tree accessor methods: **left(v), right(v), hasLeft(v)** and **hasRight(v)**. You can also assume that u is not null. Your algorithm should run in $O(n)$ time, where n is the number of nodes descending from u.

   Input: A non-null node **u** of a binary tree, and its depth d.

   Output: An object **depths** consisting of the two integer fields **depths.min** and **depths.max**, containing the minimum and maximum depth over all external nodes descending from u.

   (a) (20 marks) Your algorithm (in pseudocode or Java):
   Algorithm btDepths(u, d):

   (b) (5 marks) Provide a brief justification for why you think your algorithm is $O(n)$.

3. Suppose you have a stack S containing n elements and a queue Q that is initially empty. Describe (in pseudocode or English) how you can use Q to scan S to see if it contains a certain element x, with the additional constraint that your algorithm must return the elements back to S in their original order. You may not use an array or linked list only S and Q and a constant number of reference variables.

4. Describe the structure and pseudo-code for an array-based implementation of the array list ADT that achieves O(1) time for insertions and removals at index 0, as well as insertions and removals at the end of the array list.

5. Describe how to implement an iterator for a circularly linked list. Since hasNext() will always return true in this case, describe how to perform hasNewNext(), which returns true if and only if the next node in the list has not previously had its element returned by this iterator.

6. Describe (in pseudocode or English) an O(n) recursive algorithm for reversing a singly linked list L, so that the ordering of the nodes becomes opposite of what it was before.

7. Let T be a tree with n nodes. Define the lowest common ancestor (LCA) between two nodes v and w as the lowest node in T that has both v and w as descendents (where, by definition, a node is a descendent of itself). Given two nodes v and w, describe (in pseudocode or English) an efficient algorithm for finding the LCA of v and w. Assume that each node is extended to include an instance variable *depth* that contains the depth of the node. What is the running time of your algorithm?

8. Given a min heap T and a key k, give an algorithm to compute all of the entries in T with key less than or equal to k. The algorithm should run in time proportional to the number of entries returned.