York University | EECS 4101/5101 | February 26, 2019

Homework Assignment #5
Due: March 7, 2019 at 11:30 a.m.

1. Suppose we start out with a perfectly balanced external BST containing n_0 keys whose height is exactly $\lceil \log_2 n_0 \rceil$. The value of n_0 is known. Assume that we only want to perform a sequence of DELETE and FIND operations on this tree (no INSERTS). The argument to a DELETE operation is a pointer to the leaf that must be deleted.

Our goal is to design a data structure so that in any sequence of FIND and DELETE operations, the *worst-case* time for a FIND is $O(\log n)$ (where n is the number of keys stored in the tree when the FIND is performed) and the *amortized* time for each DELETE is $O(1)$.

The idea is that a DELETE does not actually remove the leaf. Instead, it simply marks the leaf (by setting a bit stored in the leaf), to indicate that the leaf's key has been deleted. Then, periodically, the entire tree is rebuilt to again make it a perfectly balanced BST containing all the unmarked leaves.

(a) Describe how to rebuild the tree efficiently. If the old tree contains n leaves, how much time does it take to do the rebuild? Give your answer in terms of n using Θ notation and *briefly* justify your answer.

(b) Explain how would you decide *when* to rebuild the tree in order to achieve the time bounds described above.

(c) Show that the amortized time per DELETE is $O(1)$.

(d) Show that the worst-case time per FIND is $O(\log n)$ when the tree has n (unmarked) keys.

2. Consider a 2-dimensional set ADT. It stores a set of objects. Each object has two fields called x and y. The values of these fields are drawn from two ordered sets X and Y. (Thus, we can think of the ADT storing a subset of $X \times Y$.) It supports the following three operations.

- INSERT adds a new pair to the set. Assume that the given pair is not already in the set.
- DELETE removes an object from the set. (You can assume that the argument is a pointer to the part of the data structure that represents the object.)
- QUERY, which takes two arguments $ymin$ and $ymax$ in Y, with $ymin \leq ymax$. It returns a pair whose y value is between $ymin$ and $ymax$ and whose x value is maximal among all such pairs. (Geometrically, if the x and y values represent Cartesian coordinates, you can think of this operation as returning the rightmost point in the horizontal stripe bounded by the lines $y = ymin$ and $y = ymax$. If there are multiple rightmost points, then the operation can return any of them.)

(a) Describe how to implement this ADT efficiently. All three operations should run in $O(\log n)$ time in the worst case when the set contains n objects.

Hint: use a red-black tree, where the keys represent one of the two fields, and each node is augmented to store some information about the other field.

(b) Give an example of an interesting application where this data structure would be useful. (Specify what would be stored in the data structure, what the x and y fields would represent about each object, and why you would want to do a query of the type described.)