CPS 130 Homework 13 - Solutions

1. (CLRS 14.1-5) Given an element \(x\) in an \(n\)-node order-statistic tree and a natural number \(i\), how can the \(i\)th successor of \(x\) in the linear order of the tree be determined in \(O(\log n)\) time?

Solution: The data structure should support the following two operations: \(\text{OS-Rank}(T, x)\), which returns the position of \(x\) in the linear order determined by an inorder tree walk of \(T\) in \(O(\log n)\) time, and \(\text{OS-Select}(x, i)\), which returns a pointer to the node containing the \(i\)th smallest key in the subtree rooted at \(x\) in \(O(\log n)\) time. The \(i\)th successor of \(x\) is given by \(\text{OS-Select}(x, \text{OS-Rank}(T, x) + i)\), which will also run in \(O(\log n)\) time.

2. (CLRS 14.2-1) Show how the dynamic-set queries MINIMUM, MAXIMUM, SUCCESSOR and PREDECESSOR can each be supported in \(O(1)\) worst-case time on an augmented order-statistic tree. The asymptotic performance of other operations should not be affected. (Hint: Add pointers to nodes.)

Solution:

3. In this problem we consider a data structure for maintaining a multi-set \(M\). We want to support the following operations:

- \(\text{Init}(M)\): create an empty data structure \(M\).
- \(\text{Insert}(M, i)\): insert (one copy of) \(i\) in \(M\).
- \(\text{Remove}(M, i)\): remove (one copy of) \(i\) from \(M\).
- \(\text{Frequency}(M, i)\): return the number of copies of \(i\) in \(M\).
- \(\text{Select}(M, k)\): return the \(k\)'th element in the sorted order of elements in \(M\).

If for example \(M\) consists of the elements
\[< 0, 3, 3, 4, 4, 7, 8, 8, 9, 11, 11, 11, 11, 11, 13 >\]
then \(\text{Frequency}(M, 4)\) will return 2 and \(\text{Select}(M, 6)\) will return 7.

Let \(|M|\) and \(||M||\) denote the number of elements and the number of different elements in \(M\), respectively.

(a) Describe an implementation of the data structure such that \(\text{Init}(M)\) takes \(O(1)\) time and all other operations take \(O(\log ||M||)\) time.

Solution: The idea is to store the distinct elements of the multi-set in a red-black tree. For each node \(x\) in the tree which stores the value \(k\) maintain a counter \(c(x) = \) how many elements in the multi-set are equal to \(k\). \(\text{Init}(M)\) simply initializes the red-black tree. \(\text{Insert}(M, i)\) first searches for \(i\) in the tree: if it exists, it increments its counter, otherwise it inserts it and sets its counter to 1. \(\text{Remove}(M, i)\) searches for \(i\) in the tree and if it exists, it decrements its counter, and if the counter becomes
0 it deletes that node from the tree. \( \text{Frequency}(M, i) \) searches for \( i \) and returns its counter.

In order to implement \( \text{Select}(M, k) \) we need to augment the tree with extra information such that each node can find out its rank. This is basically the same problem as augmenting a red-black tree in order to answer order statistics queries in \( O(\log n) \) time. We store in each node \( x \) a field \( \text{size}(x) \) which is the total number of nodes in the subtree rooted at \( x \), which can be computed as

\[
\text{size}(x) = \text{size}({\text{left}(x)}) + \text{size}({\text{right}(x)}) + \text{counter}(x).
\]

(b) Design an algorithm for sorting a list \( L \) in \( O(|L| \log |L|) \) time using this data structure.

**Solution:** Insert each element from the list into this data structure and then select each element.

\[
\text{For } i = 1 \text{ to } |L| \text{ Insert}(M, L[i]) \\
\text{For } i = 1 \text{ to } |L| \text{ Select}(M, i).
\]

As the tree will contain \( |L| \) distinct elements, each call of \( \text{Insert()} \) or \( \text{Select()} \) will take \( O(\log |L|) \) time.