COMS21103: Problem set 5

2015/2016

Remark: Most problem sets for the next few weeks will contain at least one starred problem, which is more challenging. If any of the problems seem unclear, please post a question on the Blackboard discussion board.

1. Using the analysis from the Bloom filters lecture, if we want to use a Bloom filter for a Web cache which stores up to 1,000 URLs picked from a universe of 10,000,000 URLs, with a 0.1% probability of false positives, how many bits of storage should we use for the Bloom filter?

   Answer (sketch): From a direct calculation, we can use 18,778 bits in total to achieve this. Note that the universe size is irrelevant.

2. Imagine the sequence of integers $5, 4, 11, 15, 9, 10, 3, 12, 6, 13$ are inserted into a skip list (in that order). By tossing a (real) fair coin, simulate the operations involved. Draw a diagram of the resulting skip list.

   Answer (sketch): There is no single answer to this :).

3. Imagine that you have a skip list data structure, as described in the lecture, except that your coin-flipper is broken! What is the worst case time complexity of the operations Insert, Delete and Find if the coin-flips are fixed to be,

   $$\text{HTHTHT}\ldots \text{(the first coin flip is always a heads)}$$

   Answer (sketch): In this case, the skip list will always insert every item into exactly two levels. Therefore the skip list will be composed of two identical linked lists. The worst case complexity of each operation is $O(n)$ (where n is the number of elements in the linked list).

4. Instead imagine that your coin-flipper is broken so that the coin-flips are always,

   $$\text{HHTTHHTTHHTT}\ldots \text{(the first coin flip is always a heads)}$$

   How many levels will there be in the tree? What is the worst case time complexity of the operations Insert, Delete and Find now?

   Answer (sketch): In this case, the first item inserted will be placed in three levels (HHT). The second item to be inserted will appear in one level (T). This process will continue to alternate. I.e. the i-th insert will appear in three levels if i is odd and one level otherwise. When there are n elements in the linked list, at least $\lfloor n/2 \rfloor$ of these are in three levels. Therefore, looking for (or deleting or inserting) the largest element takes $\Theta(n)$ time.

5. Prove the claim made in the lecture on self-balancing trees that the height of a 2-3-4 tree is always $O(\log n)$. You may use the fact that a 2-3-4 tree is always perfectly balanced when proving this.

   Answer (sketch): Consider a 2-3-4 tree of height $h > 0$ (where h does not include the dummy leaves). We will prove by induction that for $i \leq h$, level i in the tree (all nodes at depth i) contains at least 2^i nodes. For $i = 0$, there is exactly one node as required. Assume that at level $(i-1)$ there are at least $2^{(i-1)}$ nodes. If $i < h$, each node at level $(i-1)$ has at least 2 children (though it may have 3 or 4). This is because the tree is perfectly balanced. Therefore level i has at least 2^i nodes. This completes the proof by induction. We now observe that level h contains at least 2^h nodes, each holding at least one element. As the tree contains n elements, we have that $2^h \leq n$. By taking logs of both sides, we have that $h \in O(\log n)$ as required.

6. ($\star$) Explain how to augment the 2-3-4 tree data structure discussed in the lecture on self-balancing trees to support the Predecessor and RangeFind operations. Descriptions of these operations are given at the start of said lecture. The time complexity of the Predecessor operation should be $O(\log n)$. The time complexity of the RangeFind operation should be $O(\log n + m)$ where m is the number of elements returned. That is it should be *output sensitive*.

7. ($\star$) Explain how to augment the skip list data structure discussed in the lecture to support the Predecessor and RangeFind operations. Descriptions of these operations are given at the start of said lecture. The time complexity of the Predecessor operation should be $O(\log n)$. The time complexity of the RangeFind operation should be $O(\log n + m)$ where m is the number of elements returned. That is it should be *output sensitive*.