CMSC 420:Fall 2020 Dave Mount

Midterm Exam

The exam is asynchronous and online. It is open-book, open-notes, open-Internet, but it must be done on your own without the aid of other people or software. The total point value is 100 points. Good luck!

Problem 1. (10 points)

(1.1) (5 points) Consider the 2-3 tree shown the figure below. Show the **final tree** that results after the operation `insert(6)`. When rebalancing, use only splits, *no adoptions* (key rotations).

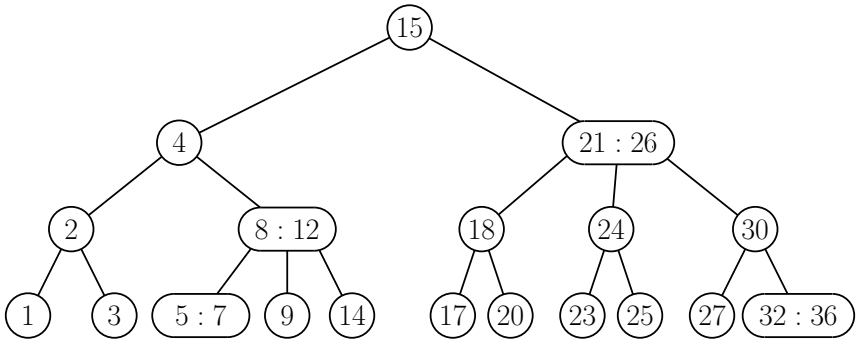

Figure 1: 2-3 tree insertion and deletion.

(1.2) (5 points) Returning to the original tree, show the **final tree** that results after the operation `delete(20)` When rebalancing, you may use *both merge and adoption* (key rotation).

In both cases, you may draw intermediate subtrees to help with partial credit, but don't waste too much time on this.

Problem 2. (25 points) Short answer questions. Except where noted, explanations are not required but may be given for partial credit.

(2.1) (4 pts) Suppose that you have a binary tree with n nodes that is *not* full. As a function of n, what is the *minimum* and *maximum* number of leaves this tree could have?

(2.2) (5 pts) You have a set of n distinct keys, where n is a large even number. You randomly permute all the keys and insert the first $n/2$ of them into a standard (unbalanced) binary search tree. You then take the remaining $n/2$ keys, sort them in ascending order, and insert them into this tree. The final tree has n nodes. As a function of n, what is the expected height of this tree? (Select the best from the choices below.)

(a) $O(\log n)$
(b) $O((\log n)^2)$
(c) $O(\sqrt{n})$

(d) $O(n)$

(2.3) (3 pts) You have a valid AVL tree with n nodes. You insert two keys, one smaller than all the keys in the tree and the other larger than all the keys in the tree, but you do no rebalancing after these insertions. **True or False**: The resulting tree is a valid AVL tree. (Briefly explain.)

(2.4) (4 pts) You have a splay tree containing n nodes for some large value n. Let x and y be two keys in the dictionary. You repeat the operations `splay(x)` followed by `splay(y)` m times, where m is much greater than n. What is the *worst-case* time complexity for all of these operations? (Select one)

(a) $O(m)$
(b) $O(m \log m)$
(c) $O(m \log n)$
(c) $O(n \log m)$
(d) $O(mn)$
(e) None of the above

(2.5) (3 pts) By mistake, two keys in your treap happen to have the same priority. Which of the following is a possible consequence of this mistake? (Select one)

(a) The `find` algorithm may abort, due to dereferencing a `null` pointer.
(b) The `find` algorithm will not abort, but it may return the wrong result.
(c) The `find` algorithm will return the correct result if it terminates, but it might go into an infinite loop.
(d) The `find` algorithm will terminate and return the correct result, but it may take longer than $O(\log n)$ time (in expectation over all random choices).
(e) There will be no negative consequences. The find algorithm will terminate, return the correct result, and run in $O(\log n)$ time (in expectation over all random choices).

(2.6) (3 pts) A scapegoat tree containing n keys has height $O(\log n)$ . . . (select one):

(a) Always—the height is guaranteed.
(b) In expectation, over the algorithm's random choices.
(c) In expectation, assuming that keys are inserted in random order.
(d) In the amortized sense—the average height will be $O(\log n)$ over a long sequence of operations.
(e) Maybe yes, maybe no—there is just no way of knowing.

(2.7) (3 pts) Given an arbitrary B-tree of order 5 with n keys (for some large value of n), what is the smallest number of keys that might be stored in any node of the tree?

(a) 1
(b) 2
(c) 3
(d) 4
(e) 5

Problem 3. (20 points) In this problem we will consider an enhanced version of a skip list. As usual, each node `p` stores a key, `p.key`, and an array of next pointers, `p.next[]`. To this we add a parallel array `p.span[]`, which contains as many elements as `p.next[]`. This array is defined as follows. If `p.next[i]` refers to a node `q`, then `p.span[i]` contains the distance (number of nodes) from `p` to `q` (at level 0) of the skip list.

For example, see Fig. 2. Suppose that `p` is third node in the skip list (key value "10"), and `p.next[1]` points to the fifth element of the list (key value "13"), then `p.span[1]` would be $5 - 3 = 2$, as indicated on the edge between these nodes.

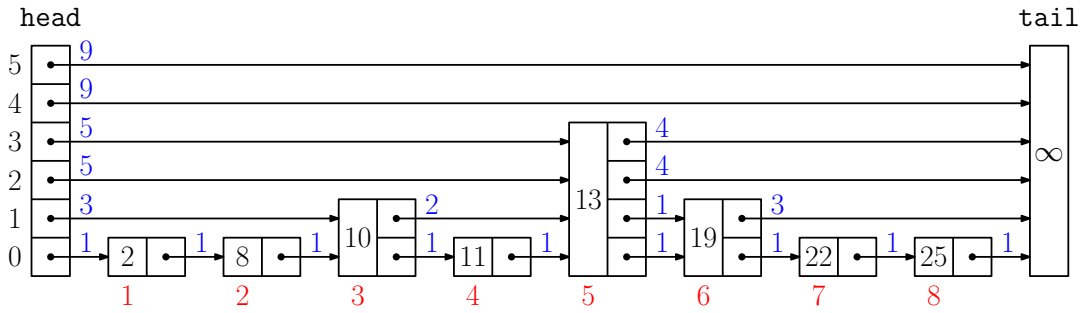

Figure 2: Skip list with span counts (labeled on each edge in blue).

(3.1) (10 points) Present pseudo-code for a function `int countSmaller(Key x)`, which returns a count of the number of nodes in the entire skip list whose key values are strictly smaller than `x`. For example, in Fig. 2, the call `countSmaller(22)` would return 6, since there are six items that are smaller than 22 (namely, 2, 8, 10, 11, 13, and 19).

Your procedure should run in time expected-case time $O(\log n)$ (over all random choices). Briefly explain how your function works.

(3.2) (10 points) Present pseudo-code for a function `Value getMinK(int k)`, which returns the value associated with the kth smallest key in the entire skip list. For example, in Fig. 2, the call `getMinK(5)` would return 13, since 13 is the fifth smallest element of the skip list. You may assume that $1 \leq k \leq n$, where n is the total number of nodes in the skip list.

Your procedure should run in time expected-case time $O(\log n)$ (over all random choices). Briefly explain how your function works.

Problem 4. (25 points) Consider the following possible node structure for 2-3 trees, where in addition to the keys and children, we add a link to the parent node. The root's parent link is `null`.

```
class Node23 {                  // a node in a 2-3 tree
    int       nChildren         // number of children (2 or 3)
    Node23    child[3]          // our children (2 or 3)
    Key       key[2]            // our keys (1 or 2)
    Node23    parent            // our parent
}
```

Assuming this structure, answer each of the following questions:

(4.1) (5 points) Present pseudocode for a function `Node23 rightSibling(Node23 p)`, which returns a reference to the sibling to the immediate right of node `p`, if it exists. If `p` is the rightmost child of its parent, or if `p` is the root, this function returns `null`. (For example, in Fig. 3, the right sibling of the node containing "2" is the node containing "8:12". Since the node containing "8:12" is the rightmost node of its parent ("4"), it has no right sibling.)

Your function should run in $O(1)$ time.

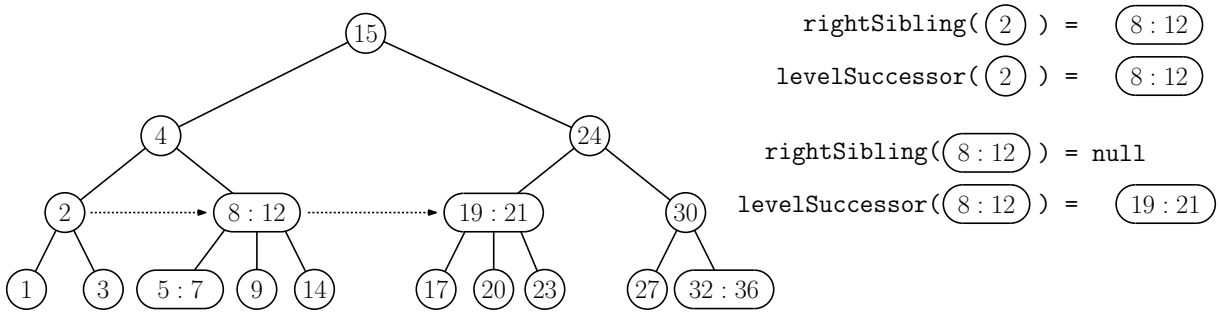

Figure 3: Sibling and level successor in a 2-3 tree.

(4.2) (10 points) For a node `p` in a 2-3 tree, its *level successor* is the node to its immediate right at the same level. Give pseudocode for a function `Node23 levelSuccessor(Node23 p)`, which returns a reference to `p`'s level successor, if it exists. If `p` is the rightmost node on its level (including the case where `p` is the root), this function returns `null`. (For example, in Fig. 3, the level successor of the node containing "2" is the node containing "8:12", and the level successor of "8:12" is the node containing "19:21".)

Your function should run in $O(\log n)$ time. If you like, you may use `rightSibling`.

(4.3) (10 points) Suppose we start at any node `p` in a 2-3 tree with n nodes, and we repeatedly perform `p = levelSuccessor(p)` until `p == null`. What is the (worst-case) total time needed to perform all these operations? (Briefly justify your answer.)

Problem 5. (20 points) A *social-distanced bit vector* (SDBV) is an abstract data type that stores bits, but no two 1-bits are allowed to be consecutive. It supports the following operations (see Fig. 4):

- `init(m)`: Creates an empty bit vector `B[0..m-1]`, with all entries initialized to zero.
- `boolean set(i)`: For $0 \le i \le m$ (where m is the current size of `B`), this checks whether the bit at positions i and its two neighboring indices, $i-1$ and $i+1$, are all zero. If so, it sets the ith bit to 1 and returns `true`. Otherwise, it does nothing and returns `false`. (The first entry, `B[0]`, can be set, provided both it and `B[1]` are zero. The same is true symmetrically for the last entry, `B[m-1]`.)

  For example, the operation `set(9)` in Fig. 4 is successful and sets `B[9] = 1`. In contrast, `set(8)` fails because the adjacent entry `B[7]` is nonzero.

There is one additional feature of the SDBV, its ability to *expand*. If we ever come to a situation where it is impossible set any more bits (because every entry of the bit vector is

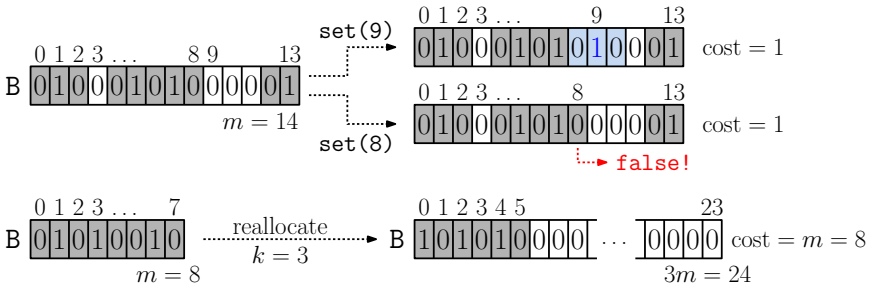

Figure 4: Social-distanced bit vector. (Shaded entries cannot be set to one, due to social-distancing.)

either nonzero or it is adjacent to an entry that is nonzero), we *reallocate* the bit vector to one of three times the current size. In particular, we replace the current array of size m with an array of size $3m$, and we copy all the bits into this new array, compressing them as much as possible. In particular, if k bits of the original vector were nonzero, we set the entries $\{0, 2, 4, \ldots, 2k\}$ to 1, and all others to 0 (see Fig. 4).

The cost of the operation `set` is 1, unless a reallocation takes place. If so, the cost is m, where m is the size of the bit vector *before* reallocation.

Our objective is to derive an amortized analysis of this data structure.

(5.1) (4 points) Suppose that we have arrived at a state where we need to reallocate an array of size m. As a function of m, what is the minimum and maximum number of bits of the SDBV that are set to 1? (Briefly explain.)

(5.2) (4 points) Following the reallocation, what is the minimum number of operations that may be performed on the data structure until the next reallocation event occurs? Express your answer as a function of m. (Briefly explain.)

(5.3) (2 points) As a function of m, what is the cost of this next reallocation event? (Briefly explain.)

(5.4) (10 points) Derive the amortized cost of the SDBV. (For full credit, we would like a tight constant, as we did in the homework assignment. We will give partial credit for an asymptotically correct answer. Assume the limiting case, as the number of operations is very large and the initial size of the bit vector is small.)

Throughout, if divisions are involved, don't worry about floors and ceilings.