CMSC 420: Spring 2022

Homework 2: Search Trees

Handed out Tue, Feb 22. Due **Wed, Mar 2, 11:59pm**. Point values are tentative and subject to change.

Important! Solutions will be discussed in class on Thu, Mar 3, so **no late submissions will be accepted**. Turn in whatever you have completed by the due date.

Problem 1. (12 points) Consider the AVL tree shown in Fig. 1.

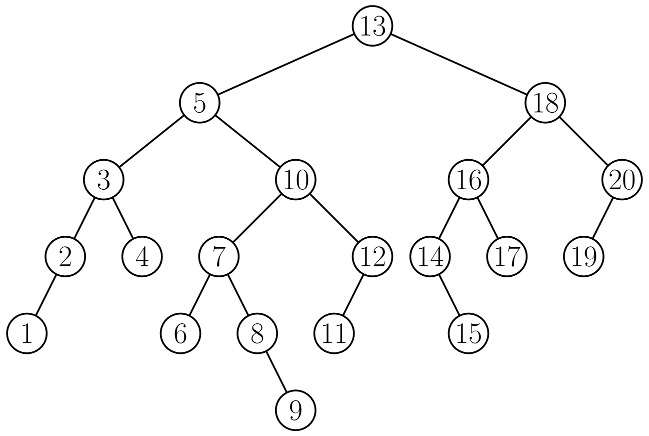

Figure 1: AVL Trees.

(a) (5 points) Draw the tree again, indicating the balance factors associated with each node.

(b) (7 points) Show the tree that results from the operation `delete(19)`, after all the rebalancing has completed. (We only need the final tree. You can provide intermediate results for partial credit.)

Problem 2. (12 points) Consider the AA trees shown in Fig. 2.

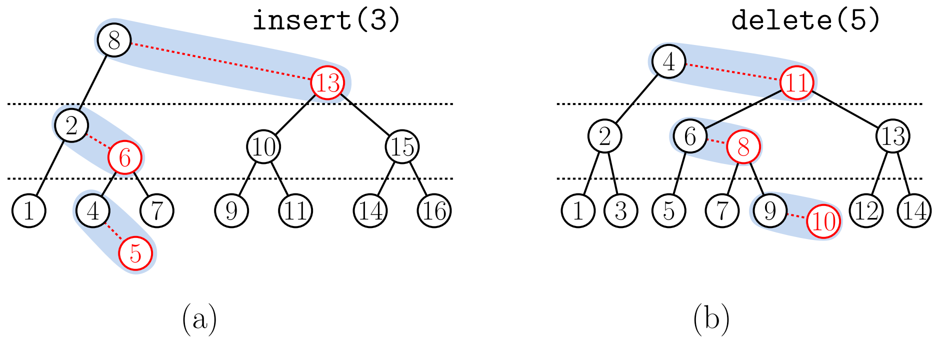

Figure 2: AA Trees.

(a) (6 points) Show the result of performing the operation `insert(3)` into the tree in Fig. 2(a).

(b) (6 points) Show the result of performing the operation `delete(5)` from the tree in Fig. 2(b).

(In both cases, we only need the final tree. You can provide intermediate results for partial credit. If you don't have two different colored pens, you can indicate red nodes with dashed edges and/or encircle circle groups of nodes as we do in our figures.)

Problem 3. (7 points) Recall the right rotation operation for a binary tree (given in Lecture 5).

```
Node rotateRight(Node p) {
    Node q = p.left
    p.left = q.right
    q.right = p
    return q
}
```

Suppose that we wish to apply this to a *threaded binary tree* using inorder threads (defined in Lecture 3). Explain what modifications (if any) are needed to perform a right rotation at node `p` so that after your modified function executes, all child links and threads are properly set. You may assume that the call is valid, in particular, `p` is non-null and `p`'s left-child link is standard parent-child pointer, and not a thread. Present your modified pseudocode and briefly explain why it is correct.

Note 1: Recall that the boolean's `leftIsThread` and `rightIsThread` are used to indicate that the left/right child link is a thread. These values may also need to be updated.

Note 2: It is possible that the rotation operation cannot be defined because it involves global knowledge of the tree structure beyond what is accessible through node `p`. If this is so, please explain why this is the case.

Problem 4. (7 points) Recall the code (shown below) for the operations `skew` and `split` for AA-trees (from Lecture 7).

```
AANode skew(AANode p) {               |   AANode split(AANode p) {
    if (p == nil) return p            |       if (p == nil) return p
    if (p.left.level == p.level) {    |       if (p.right.right.level == p.level) {
        AANode q = p.left             |           AANode q = p.right
        p.left = q.right              |           p.right = q.left
        q.right = p                   |           q.left = p
        return q                      |           q.level += 1
    }                                 |           return q
    else return p                     |       }
}                                     |       else return p
                                      |   }
```

Also recall that each invocation of the `insert` function, the last line is "`return split(skew(p))`". There is an interesting phenomenon that sometimes occurs. It is illustrated in Fig. 3.

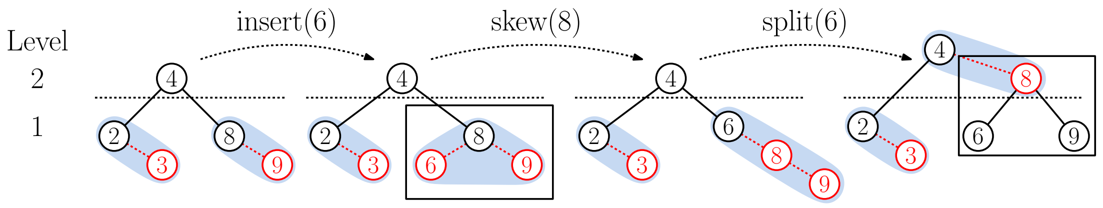

Figure 3: Ineffective skew-split combination.

Observe that the invocation of `skew(8)` results in a right rotation at `8` and returns a reference to node `6`. The subsequent invocation of `split(6)` results in a left rotation at `8` (thus undoing the previous rotation). It then promotes `8` to the next higher level. These two rotations effectively undo each other, and we call this skew-split combination *ineffective*.

In an effort to improve the efficiency of the AA tree, your task is to write an "effective" variant of skew-split. Your function, called `effectiveSkewSplit(p)` must be functionally equivalent to `split(skew(p))`. That is, it must have the same effect on the tree's structure, and it must return the same result. The only difference is that, it detects when an ineffective skew-split is about to occur and avoids doing the two rotations.

Present pseudocode for your function and explain why it is correct. As with skew and split, your function should run in $O(1)$ time.

Problem 5. (12 points) Each node of a 2-3 tree may have either 2 or 3 children, and these nodes may appear anywhere within the tree. Let's imagine a much more rigid structure, where the node types alternate between levels. The root is a 2-node, its two children are both 3-nodes, their children are again 2-nodes, and so on (see Fig. 4). Generally, depth i of the tree consists entirely of 2-nodes when i is even and 3-nodes when i is odd. (Remember that the *depth* of a node is the number of edges on the path to the root, so the root is at depth 0.) We call this an *alternating 2-3 tree*. While such a structure is too rigid to be useful as a practical data structure, its properties are easy to analyze.

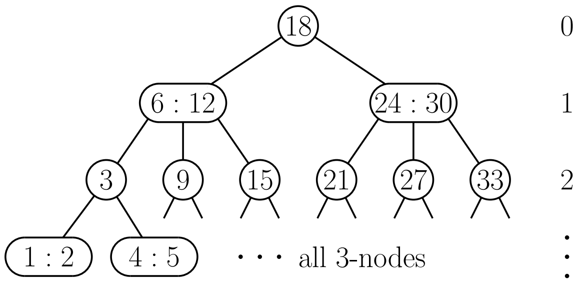

Figure 4: Alternating 2-3 tree.

(a) (6 points) For $i \geq 0$, define $n(i)$ to be the number of nodes at depth i in an alternating 2-3 tree. Derive a closed-form mathematical formula (exact, not asymptotic) for $n(i)$. Present your formula and briefly explain how you derived it.

By "closed-form" we mean that your answer should just be an expression involving standard mathematical operations. It is *not* allowed to involve summations or recurrences, but it is allowed to include cases, however, such as

$$n(i) \;=\; \begin{cases} \ldots & \text{if } i \text{ is even} \\ \ldots & \text{if } i \text{ is odd.} \end{cases}$$

(b) (6 points) For $i \geq 0$, define $k(i)$ to be the number of keys stored in the nodes at depth i in an alternating 2-3 tree. (Recall that each 2-node stores one key and each 3-node stores 2 key). Derive a closed-form mathematical formula for $k(i)$. Present your formula and briefly explain how you derived it. (The same rules apply for "closed form", and further your formula should stand on its own and not make reference to $n(i)$ from part (a).)

Challenge Problem 1: Continuing Problem 5 on the alternating 2-3 tree, for $i \geq 0$, define $N(i)$ to be the total number of nodes in all depths from 0 through i, and define $K(i)$ to be the total number of keys in all depths from 0 through i. Derive a closed-form mathematical formula for $N(i)$ or $K(i)$ (your choice). Present your formula and briefly explain how you derived it. As before, your formula should stand on its own and not make reference to $n(i)$ or $k(i)$.)

Challenge Problem 2: This problem is actually quite simple, but the "challenge" is to familiarize yourself with the section of the Latex lecture notes, Lecture X01, that discusses the amortized analysis of quake heaps.) In this problem we are going to do a walk-through of the amortized analysis of Quake Heaps for a single example of `extract-min` shown in Fig. 5.

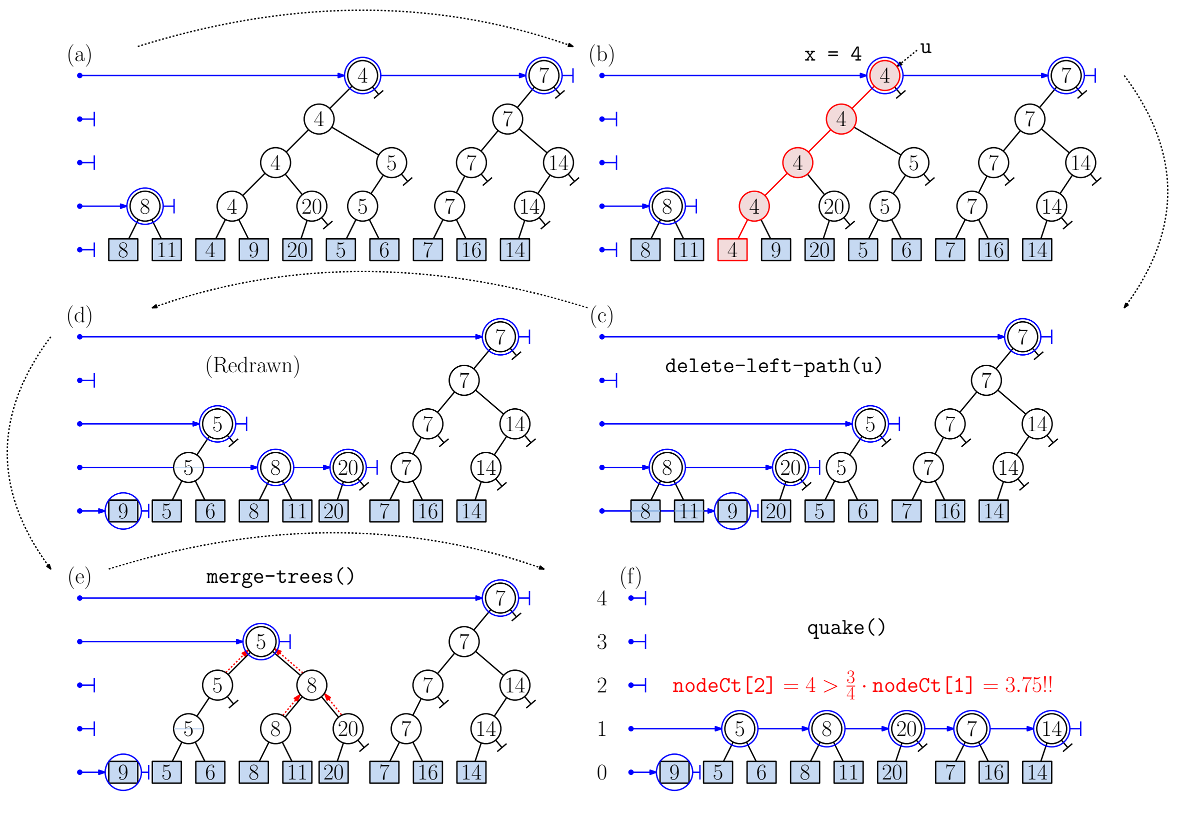

Figure 5: Quake Heap amortized analysis.

Recall that the *potential* for the data structure is defined to be $\Psi = N + 2R + 4B$, where N is the total number of nodes (both internal nodes and roots), R is the number of root nodes, and B is the number of nodes that have exactly one child (so called "bad nodes").

(a) Consider the initial quake heap shown in Fig. 5(a). For this tree, what are the values of N, R, B, and the potential Ψ?

(b) First, the algorithm searches all the roots to find the smallest key. Let T_b denote the actual cost, namely number of roots roots visited. What is T_b? Has the potential changed at all?

(c) Second, we delete all the nodes along the left path leading to the minimum root 4 (see Fig. 5(b)-(c)). Let T_c denote the actual number of nodes deleted. What is the value of T_c, and what is the net change to N, R, and B as a result? (Note that we have created some new roots in the process. If the number decreases, then the net change is negative.) Let $\Delta\Psi_c$ denote the net change to the potential. What is $\Delta\Psi_c$ and what is the total contribution $T_c + \Delta\Psi_c$ to the amortized cost?

(d) Next, we perform merge-trees (see Fig. 5(e)). Let T_d denote the actual cost of the number of new nodes created by the merging process. What is the value of T_d, and what is the net change to N, R, and B as a result? Let $\Delta\Psi_d$ denote the net change to the potential. What is $\Delta\Psi_d$ and what is the total contribution $T_d + \Delta\Psi_d$ to the amortized cost?

(e) Finally, we perform the quake operations (see Fig. 5(f)). Let T_e denote the actual cost of the number of new nodes deleted by the quake operations. What is the value of T_e, and what is the net change to N, R, and B as a result? Let $\Delta\Psi_e$ denote the net change to the potential. What is $\Delta\Psi_e$ and what is the total contribution $T_e + \Delta\Psi_e$ to the amortized cost?

(f) In summary, what is the total actual costs from this operation $T = T_b + \cdots + T_e$, what is the total change in potential $\Delta\Psi = \Delta\Psi_c + \cdots \Delta\Psi_e$, and what is the final amortized cost $T + \Delta\Psi$? (Note that total amortized cost may be negative, since we have improved the structure more than the actual amount of work needed to perform the operations.)