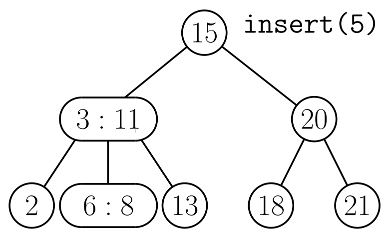
CMSC 420: Spring 2022

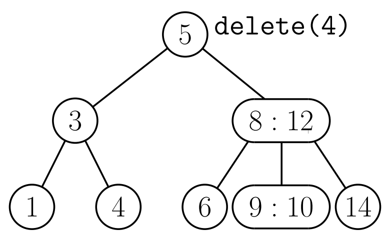
CMSC 420 (0101) - Midterm Exam 1

Problem 1. (10 points) For this problem, use the algorithms presented in class for 2-3 trees. (You will receive partial credit if you produce a valid 2-3 tree, but not using the algorithm from class.) Intermediate results are not required.

(a) (5 points) Show the **final tree** that results after the operation `insert(5)` to the 2-3 tree in the figure below.



(b) (5 points) Show the **final tree** that results after the operation `delete(4)` to the 2-3 tree in the figure below.



Problem 2. (30 points) Short answer questions. Unless requested, explanations are not required, but may always be given to help with partial credit.

(a) (4 points) You have an inorder-threaded binary tree with n nodes. Without knowing the tree structure, is it possible to know how many of the links in this tree are threads (versus normal parent-child links)? If so, indicate how many as a function of n.

(b) (6 points) You have a standard (unbalanced) binary search tree storing the consecutive odd keys $\{1, 3, 5, 7, 9, 11, 13\}$ (which may have been inserted in any order). Into this tree you insert the consecutive keys $\{0, 2, 4, 6, 8, 10, 12, 14\}$ (also inserted in any order). Which of the following statements hold for the resulting tree. (Select all that apply.)

(1) It is definitely a full binary tree

(2) It is definitely **not** a full binary tree

(3) It is definitely a complete binary tree

(4) It is definitely **not** a complete binary tree

(5) Its height is larger than the original by exactly 1

(6) Its height is larger than the original, but the amount of increase need not be 1

(c) (5 points) You have an AA tree that contains an even number of keys n. As a function of n, what is the minimum and maximum number of red nodes that might be in this tree?

(d) (5 points) Suppose that you insert 13 keys into a quake heap and then perform the merge-trees operation. How many roots are there at each of the levels 0–4 in the resulting structure?

(e) (4 points) You insert a node x into a treap having at least three entries, and you observe that after the insertion, x is at the root of the tree. What can you say about the random priority assigned to x?

(1) It is the smallest

(2) It is the largest

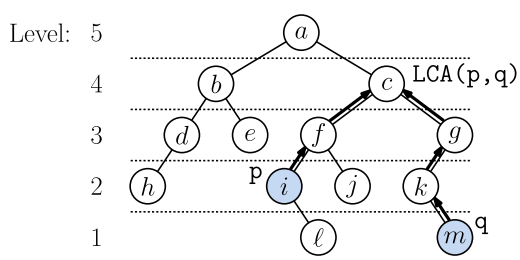
(3) It is the median

(4) You can't infer anything about x's priority

(f) (6 points) A hacker tries to mess with your skip list as follows. First, they insert a large number of keys. After this, they delete all the keys with nodes that contribute to levels 1 and higher, effectively reducing your skip list to a standard linked list.

Is this an issue that the skip-list designer needs to worry about? Take a position (either "*This is an issue*" or "*This is not an issue*") and briefly justify your position. (You may assume the hacker does *not* have access to your random number generator.)

Problem 3. (20 points) Given two nodes `p` and `q` in a binary tree, their *lowest common ancestor*, denoted `LCA(p,q)`, is the common ancestor of these two nodes that is closest to both. (For example, in the figure below `LCA(p,q)` is the node labeled "c".) If `q` is an ancestor of `p` (possibly `p` itself), then `LCA(p,q) = q`.



In this problem, we assume that we are given a binary tree. Each node `p` is associated with the usual child pointers `p.left` and `p.right` as well as a parent pointer, `p.parent`.

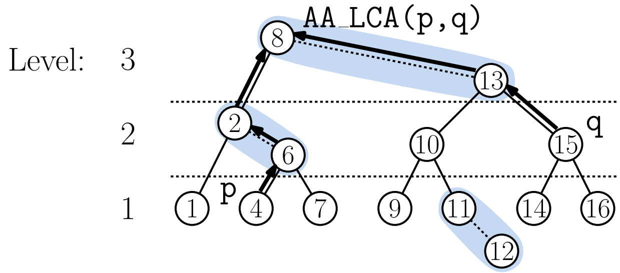
(a) (10 points) Each node `p` is associated with a *level number*, `p.level`, which grows by 1 as we go from child to parent. Present pseudocode for a function

```
Node LCA(Node p, Node q)
```

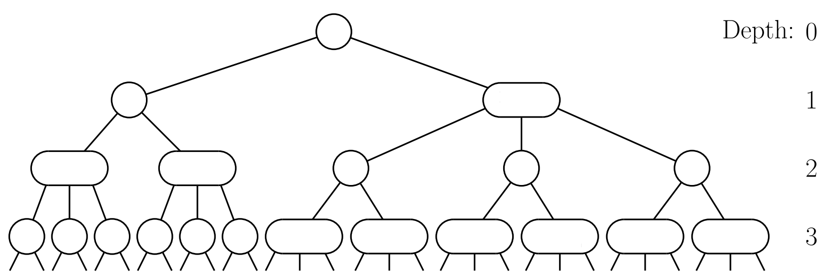
which returns the LCA of `p` and `q`. (For full credit, it should run in time proportional to the height of the tree and should not modify the tree.)

Hint: Move `p` and `q` up in a coordinated manner until they converge at the LCA.

(b) (10 points) Repeat part (a), but with the following twist. The tree is a (valid) AA-tree. This means that a node and its parent might be at the same level, according to the rules of AA trees. Call this function `AA_LCA`.



Problem 4. (20 points) This is a variant of the HW 2 problem called a *skewed alternating 2-3 tree*. The root is a 2-node. Its left child is a 2-node, and its right child is a 3-node. For each successive level the nodes alternate. The children of each 2-node are 3-nodes, and the children of each 3-node are 2-nodes.



(a) (10 points) For $i \geq 0$, define $n(i)$ to be the number of nodes at depth i in a skewed alternating 2-3 tree. Derive a **recurrence** for $n(i)$. Present your recurrence and briefly explain how you derived it.

Hint: I believe that the recurrence is simplest when you work two levels at a time, for example, try to express $n(i)$ in terms of $n(i-2)$. Be sure to give the base case(s).

(b) (10 points) Derive a **closed-form mathematical formula** (exact, not asymptotic) for $n(i)$. Present your formula and briefly explain how you derived it.

As in the homework, your formula should *not* involve summations or recurrences, but can involve multiple cases. You do *not* need to use the result of (a), and you do *not* need to give a formal proof of correctness.

Problem 5. (20 points) In this problem, we will consider variations on the amortized analysis of the dynamic stack. Let us assume that the array storage only *expands*, it never contracts. As usual, if the current array is of size m and the stack has fewer than m elements, a `push` costs 1 unit. When the mth element is pushed, an overflow occurs.

(a) (10 points) You are given two constants $\gamma, \delta > 1$. When an overflow occurs, we allocate a new array of size γm, copy the elements from the old array over to the new array. The total cost is 1 (for the push) plus δm (for copying). Derive a tight bound on the amortized cost, which holds in the limit as $m \to \infty$. Express your answer as a function of γ and δ. Explain your answer.

(You can do the special case $\gamma = 2$ for half-credit.)

(b) (10 points) Your computer has a *hardware accelerator* that copies a block of memory of size k in time $k/(\lg k)$. When the stack overflows, we allocate a new array of size $2m$, copy the elements from the old array over to the new array. The total cost is 1 (for the push) plus $m/(\lg m)$ (for copying). Derive a tight bound on the amortized cost, which holds in the limit as $m \to \infty$. Explain your answer.