University of Waterloo
CS240E, Spring 2025
Assignment 3

Due Date: Tuesday, June 17, 2025 at 5pm, with a grace period until 7:59pm.

Question 1 [20 marks]

We define a variant of AVL trees called *d-AVL trees* as follows. Let d be a fixed constant. If the height of a node z is greater than d, then the usual height-balance property of AVL-trees must be satisfied at z. If the height of a node z is at most d, then there is no balance-property that needs to be satisfied. Show that a d-AVL tree with n nodes has height $O(logn)$.

Question 2 [20 marks]

Recall that the `Selection` problem receives as input a set of n items and an integer k with $0 \leq k \leq n-1$ and it must return the item that would be at $A[k]$ if the items were put into an array A in sorted order.

a) Argue that any comparison-based algorithm for the Selection problem on n items must have $\Omega(n/4)$ worst-case time. For partial credit, argue it must have $\Omega(\log n)$ worst-case time. The 4 here is a hint.

b) Let T be a scapegoat tree that stores n items. Design an algorithm that solves `Selection`(T, k) in $O(\log n)$ time.

Question 3 [20 marks]

Consider the following algorithm to find the minimum in a binary search tree.

Algorithm 1: *findMin*(root r)

```
1 if (r is NIL) then return "empty tree"
2 while r.leftChild != NIL do  r ← r.leftChild
3 return r.key
```

Let $T^{\text{avg}}(n)$ (for $n \geq 0$) be the average-case number of executions of the while-loop in *findMin* for a tree with n nodes. Here the average is taken over all binary search trees that store $\{0, \ldots, n-1\}$, and $T^{\text{avg}}(0) = T^{\text{avg}}(1) = 0$.

a) Show that for $n \geq 2$ we have $T^{\text{avg}}(n) \leq 1 + \frac{1}{C(n)} \sum_{i=0}^{n-1} C(n-i-1)C(i)T^{\text{avg}}(i)$, where $C(n)$ is the number of binary search trees that stores $\{0, \ldots, n-1\}$. Be as precise as we were in class for *avgCaseDemo*.

b) Show that $T^{\text{avg}}(n) \in O(\log n)$. (We recommend that you show $T^{\text{avg}}(n) \leq 2 \log n$, and that you consider a 'good case' where the left subtree has size at most $n/2$.) You may use without proof that $C(n) = \sum_{i=0}^{n-1} C(i) \cdot C(n-i-1)$, and you may assume that n is divisible as needed.)

Question 4 [20 marks]

Motivation: Scapegoat trees as defined in class store at each node z the size of the subtree rooted at z. This is not actually required; this assignment will guide you towards a variation that operates without storing the size of the subtree.

Define a *light scapegoat tree* to be a binary search tree that is *height-balanced*, by which we mean that the height is at most $\lfloor \log_{4/3} n \rfloor$ (or equivalently, every node has depth at most $\lfloor \log_{4/3} n \rfloor$). You may assume that a light scapegoat tree has a field *size* with its number of items. However, a light scapegoat tree does not store its height, and a node knows *nothing* except its key and left and right subtree. (In particular it knows neither its depth, nor its parent, nor the size of its subtree, nor the height of its subtree.)

Algorithm 2 gives (incomplete) pseudo-code to insert in such a tree:

Algorithm 2: *LightScapegoatTree::insert*(k, v)

```
   // Current tree is height-balanced
1  z ← BST::insert(k, v)
2  if height-unbalanced(z) then
3  |  p ← lowest-small-ancestor(z)
4  |  completely rebuild the subtree at p as a perfectly size-balanced tree
```

a) Show that (after inserting the new leaf z) the tree can be height-unbalanced only if the depth of z is too big.

b) Design algorithm *height-unbalanced*(z) to test whether the tree is now no longer height-balanced. The run-time must be $O(\log n)$, where n is the current size of the tree.

c) Show that if the tree is not height-balanced after *BST::insert*, then there exists an ancestor x of leaf z such that

$$size(x) < \left(\tfrac{4}{3}\right)^{d(x,z)}.$$

Here $size(x)$ denotes the number of items in the subtree rooted at x, and $d(x, z)$ denotes the distance from z to x, i.e., the number of levels that x is above z. (So $d(z, z) = 0$, $d(parent(z), z) = 1$, etc.).

d) Sub-routine *lowest-small-ancestor*(z) does the following: Find an ancestor x of z with $size(x) < \left(\tfrac{4}{3}\right)^{d(x,z)}$, and among all those, return the one that minimizes $d(x, z)$. You need **not** say how to implement this (it is not easy to do efficiently).

Argue that the rest of the insertion-routine is correct. Thus, show that after rebuilding the subtree (at the node p returned by *lowest-small-ancestor*) to be perfectly size-balanced, the resulting binary search tree is height-balanced.

For all parts, you may use results of previous parts even if you did not prove them.

Continuing the "motivation": We are *not* asking you to show that *LightScapegoatTree::insert* has $\Theta(\log n)$ amortized time, but with the above results it would be very easy to do so using the potential function from class.

Question 5 [20 marks]

Splay trees are, as AVL and $(2, 4)$ trees, binary search trees whose stucture can change each time an element is inserted or removed. They differ in that their structure is also changing when an element is simply searched in them, in order to adapt the structure of the tree to the frequency of the search queries for each element, a bit like the `Move to front` heuristic on lists.

Splay trees support the search, insertion and deletion operations, but we consider here only the search. The search is performed as in any binary search tree, but is always followed by a splay: if the element is found it is splayed to the root, otherwise the last element accessed is splayed to the root.

The splay operation is performed on a node x through a sequence of splay steps, each of which moves it closer to the root. Each particular step reorganizes the tree differently depending on the position of x relative to its grand-parent, if it has one.

- If x is the left child of the root, a zig step (described in Figure 3) is performed, and if x is the right child of the root, a zag step (symmetric from the zig step) is performed.

- If x is the left-most grand-child of its grand-parent, a zig-zig step is performed (described in Figure 2), and similarly a zag-zag step is performed if x is the right-most grand-child of its grand-parent.
- if x is the right child of the left child of its grand-parent, a zig-zag step is performed (described in Figure 1), and similarly a zag-zig step is performed in the symmetric case where x is the left child of the right child of its grand-parent.

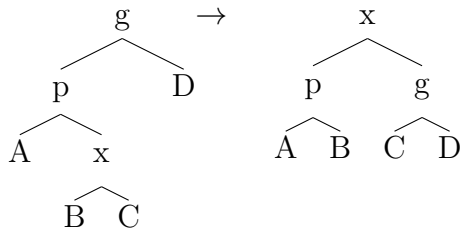

Figure 1: **Zig-zag Step**: when x is the right child of p and p is the left child of g, p becomes the new left child of x, g the new right child of x, and the subtrees A, B, C, and D of x, p, and g are rearranged as necessary.

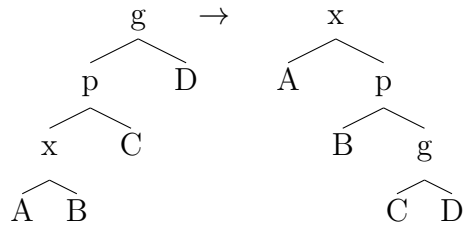

Figure 2: **Zig-zig Step**: when x is the left child of p and p is the left child of g. p becomes the new right child of x, g becomes the new right child of p, and the subtrees A, B, C, and D of x, p, and g are rearranged as necessary.

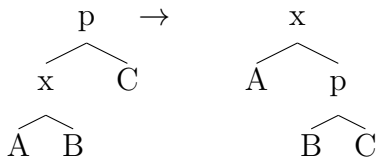

Figure 3: **Zig Step**: when x does not have a grandparent, the tree is updated through a simple rotation on the edge between x and it's parent p. Zig steps exist to deal with the parity issue and will be done only as the last step in a splay operation and only when x has odd depth at the beginning of the operation.

a) Consider the binary search tree given in Figure 4. Describe a sequence of search queries which would transform this tree into the tree in Figure 5.

b) Draw the state of the tree after each search of the following sequence of search queries:

$$1, 2, 3, 4, 5, 6, 7, 8, 1, 2, 3, 4, 5, 6, 7, 8$$

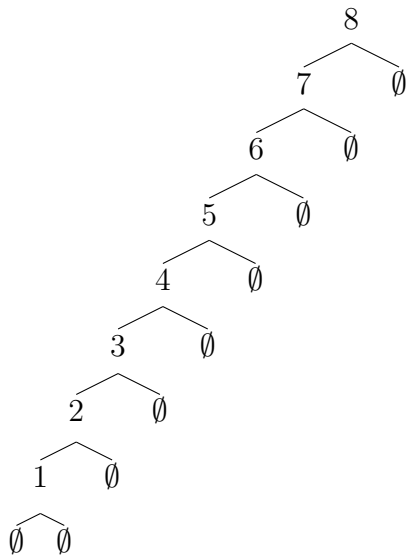

Figure 4: The initial binary search tree used in the first question.

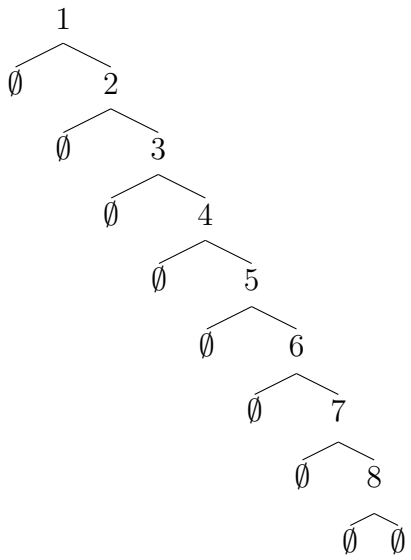

Figure 5: The target binary search tree used in the first question.

c) Give a static binary search tree of optimal height containing the same set of elements.

d) Does the splay tree perform better or worse than the static binary search tree in terms of comparisons, on the sequence of searches given? If better, give a sequence of 5 searches on which this splay tree performs worse. If worse, give a sequence of 5 searches on which this splay tree performs better.