4231 Homework 3

Eumin Hong (eh2890)

February 28, 2022

Problem 1

Prove the following useful property of a binary search tree (with distinct keys):

Property 1. *Let x be a node in a BST T. Let* max *and* min *denote the largest and smallest keys in the subtree rooted at x, respectively. For any node y outside the subtree rooted at x, show that either $y.key >$* max *or $y.key <$* min*. This implies that if there is a key k in the tree that satisfies* min $\leq k \leq$ max *then it must lie inside the subtree rooted at x. (Here the subtree rooted at x includes x itself.)*

Use it to solve Exercise 12.2-5, 12.2-6, and 12.2-9 on page 293. In all three exercises, we assume the BST has distinct keys.

12.2-5

Show that if a node in a binary tree has two children, then its successor has no left child and its predecessor has no right child.

12.2-6

Consider a binary search tree T whose keys are distinct. Show that if the right subtree of a node x in T is empty and x has a successor y, then y is the lowest ancestor of x whose left child is also an ancestor of x. (Recall that every node is its own ancestor.)

12.2-9

Let T be a binary search tree whose keys are distinct, let x be a leaf node, and let y be its parent. Show that $y.key$ is either the smallest key in T larger than $x.key$ or the largest key in T smaller than $x.key$.

Problem 2

Problem 13-2 on page 332: Join operation. For a): you only need to answer the following two questions:

1. Let T be a red-black tree in which the root has black height $T.bh$. Then after an insertion, $T.bh$ either stays the same or increases by 1. Describe the scenario when it increases by 1.
2. If a node z has black height h, use $O(1)$ time to compute the black height of z's children.

Replace d) with the following: If $T_1.bh = T_2.bh$, what color should we make x to get a red-black tree? If $T_1.bh > T_2.bh$, what color should we make x so that properties 1, 2, 3, and 5 are maintained? "Briefly" describe how to enforce property 4 in $O(\lg n)$ time.
Skip e) and f).

13-2

The ***join*** operation takes two dynamic sets S_1 and S_2 and an element x such that for any $x_1 \in S_1$ and $x_2 \in S_2$, we have that $x_1.key \leq x.key \leq x_2.key$. It returns a set $S = S_1 \cup \{x\} \cup S_2$. In this problem, we investigate how to implement the join operation on red-black trees.

a. From the original problem description:

1. Let T be a red-black tree in which the root has black height $T.bh$. Then after an insertion, $T.bh$ either stays the same or increases by 1. Describe the scenario when it increases by 1.
2. If a node z has black height h, use $O(1)$ time to compute the black height of z's children.

We wish to implement the operation RB-Join(T_1, x, T_2), which destroys T_1 and T_2 and returns a red-black tree $T = T_1 \cup \{x\} \cup T_2$. Let n be the total number of nodes in T_1 and T_2.

b. Assume that $T_1.bh \geq T_2.bh$. Describe an $O(\lg n)$-time algorithm that finds a black node y in T_1 with the largest key from among those nodes whose black-height is $T_2.bh$.

c. Let T_y be the subtree rooted at y. Describe how $T_y \cup \{x\} \cup T_2$ can replace T_y in $O(1)$ time without destroying the binary-search-tree property.

d. From the original problem description: If $T_1.bh = T_2.bh$, what color should we make x so that properties 1, 2, 3, and 5 are maintained? "Briefly" describe how to enforce property 4 in $O(\lg n)$ time.

Problem 3

Exercise 14.3-6 on page 354. You only need to describe the following key points:

1. What extra information to store in each node?
2. With this additional information in each node, how to answer MIN-GAP efficiently?
3. Use Theorem 14.1 to prove that insertion and deletion of a node can still be done in $O(\lg n)$ time: Show that all the extra information for a node x can be derived from the information stored in its two children in $O(1)$ time.

14.3-6

Show how to maintain a dynamic set Q of numbers that supports the operation MIN-GAP, which gives the magnitude of the difference of the two closest numbers in Q. For example, if $Q = \{1, 5, 9, 15, 18, 22\}$, then MIN-GAP($Q$) returns $18 - 15 = 3$, since 15 and 18 are the two closest numers in Q. Make the operations INSERT, DELETE, SEARCH, and MIN-GAP as efficient as possible, and analyze their running times.

Problem 4

Problem 16-1 (b) and (c) only on page 447.

16-1

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

c. Given a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

Problem 5

Problem 16-2 on page 447.

Suppose you are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of tasks, where task a_i requires p_i units of processing time to complete, once it has started. You have one computer on which to these tasks, and the computer can run only one task at a time. Let c_i be the ***completion time*** of task a_i, that is, the time at which task a_i completes processing. Your goal is to minimize the average completion time, that is, to minimize $(1/n)\sum_{i=1}^{n} c_i$. For example, suppose there are two tasks, a_1 and a_2, with $p_1 = 3$ and $p_2 = 5$, and consider the schedule in which a_2 runs first, followed by a_1. Then $c_2 = 5, c_1 = 8$, and the average completion time is $(5 + 8)/2 = 6.5$. If task a_1 runs first, however, then $c_1 = 3, c_2 = 8$, and the average completion time is $3 + 8)/2 = 5.5$.

a Give an algorithm that schedules the tasks so as to minimize the average completion time. Each task must run non-preemptively, that is, once task a_i starts, it must run continuously for p_i units of time. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.

b Suppose now that the tasks are not all available at once. That is, each task cannot start until its ***release time*** r_i. Suppose also that we allow ***preemption***, so that a task can be suspended and restarted at a later time. For example, a task a_i with processing time $p_i = 6$ and release time $r_i = 1$ might start running at time 1 and be preempted at time 4. It might then resume at time 10 but be preempted at time 11, and it might finally resume at time 13 and complete at time 15. Task a_i has run for a total of 6 time units, but its running time has been divided into three pieces. In this scenario, a_i's completion time is 15. Give an algorithm that schedules the tasks so as to minimize the average completion time in this new scenario. Prove that your algorithm minimizes the average completion time, and state the running time of your algorithm.