CS120: Intro. to Algorithms and their Limitations Prof. Salil Vadhan

Problem Set 3

Harvard SEAS - Fall 2021 *Due: Wed Sep. 29, 2021 (5pm)*

Your name:
Collaborators:
No. of late days used on previous psets:
No. of late days used after including this pset:

The purpose of this problem set is to give you experience reasoning about correctness and efficiency of dynamic data-structure operations, on variants of binary-search trees.

Specifically, we will work with *selection data structures*. We have seen how binary search trees can support min queries in time $O(h)$, where h is the height of the tree. A generalization is *selection* queries, where given a natural number q, we want to return the q'th smallest element of the set. So `DS.select(0)` should return the item-key pair with the minimum key among those stored by the data structure `DS`, `DS.select(1)` should return the one with the second-smallest key, `DS.select(n-1)` should return the one with the maximum key if the set is of size n, and `DS.select((n-1)/2)` should return the median element if n is odd.

In the Roughgarden text (§11.3.9), it is shown that if we *augment* binary search trees by adding to each node v the size of the subtree rooted at v, then Selection queries can be answered in time $O(h)$[1]

1. In the Github repository, we have given you a Python implementation of size-augmented BSTs supporting search, insertion, and selection, and with stubs for deletion and rotation. One of the implemented functions (`search`, `insert`, or `select`) has a correctness error, and another one is too slow (running in time linear in the number of nodes of the tree rather than in the height of tree). Identify and correct these errors. You should provide a text explanation of the errors and your corrections, as well as implement the corrections in Python.

2. In class on 9/21, we saw how to perform deletion in time $O(h)$ (in the active learning exercise) and rotation in time $O(1)$ (during lecture).

   (a) Describe (in pseudocode or pictures) how to extend these operations to size-augmented BSTs, and argue that your extensions maintain the runtimes of $O(h)$ and $O(1)$, respectively.

   (b) Prove that your new deletion and rotation operations preserve the invariant of correct size-augmentations. (That is, if every node's size attribute had the correct subtree size before the operation, then the same is true after the operation.)

   (c) Implement your new operations in Python (in the stubs we have given you, which for simplicity are intended to only do rotation at non-root nodes).

3. Applying what you have done above to AVL Trees, show that there exists a dynamic data structure for storing item-key pairs supporting insertion, search, and selection in time $O(\log n)$,

[1] Note that the Roughgarden text uses a different indexing than us for the inputs to Select. For Roughgarden, the minimum key is selected by Select(1), whereas for us it is selected by Select(0).

where is n is the number of item-key pairs stored at the time of the query. Your solution should precisely describe the form of the data structure — namely, what attributes are stored at each node and what invariants they should satisfy[2] — and then explain why all of the aforementioned operations can be done in time $O(\log n)$. You do *not* need to precisely describe how to implement the operations here, but rather explain how an $O(\log n)$-time implementation of the operations follows from what you have seen (above, in lecture, and in the Roughgarden text). In particular, you can use the high-level description of how AVL Inserts work from lecture.

[2]By *invariants*, we mean constraints or properties that the data structure should always maintain. For example, in a sorted array data structure, our invariants are that the data structure is an array containing all of the item-key pairs that have been inserted but not deleted yet, and that the keys are in ascending order.