Homework 5

Joni Vrapi

11/6/2022

Statement of Integrity: I, Joni Vrapi, attempted to answer each question honestly and to the best of my abilities. I cited any, and all, help that I received in completing this assignment.

Problem 1. In order to solve this question, we will use information gleaned from [1], [2], and [3].

To store our elements, we will use a dynamic array. A dynamic array is an implementation of a normal array whereby when the array is full, its size is geometrically increased, generally by a factor of 2. The worst case cost of an insertion into a dynamic array is $O(n)$ [4], in the case where the array is full, and all elements must be copied over to a newly allocated array which is double (or more) the size, followed by the insert of the original element. Its amortized cost of insertion, however, is $O(1)$ as it just appends a value on to the end of the array like normal. Our INSERT(S, x) therefore will run in $O(1)$.

In order to delete the larger half, we will first look through the array using the SELECT algorithm presented in the Order-Statistic section of our book (Section 9.3) [1] which per the book takes $O(n)$ time, in order to find the element e with the order-statistic $\lceil |S|/2 \rceil$. We will then go through the array and copy any elements that are $\leq e$ out into a new array that is half the size of the original. This operation will take $O(|S|)$ time while reducing the number of elements by $\lfloor |S|/2 \rfloor \in \Omega(|S|)$. We can make these operations take constant time by choosing a potential function to be linear in $|S|$. Since the INSERT operation only increases the size of S by one, there is only a constant amount of work added towards the potential, so the total amortized cost is still constant.

Finally, to output all the elements in S in $O(|S|)$ time, you would just iterate through the elements and print out each.

Problem 2. We begin by performing [5] n MAKE-SET operations on $\{x_1\}...\{x_n\}$. Then we create a binomial tree of height $log(n)$ using $n-1$ UNION's by doing a UNION(x_1, x_2) then UNION(x_3, x_4)...UNION(x_{n-1}, x_n) sets of size 2, then we create sets of size 4 by a pairwise UNION of these. This is a binomial tree, so at least half of its n nodes are at a depth $\geq log(n)/2$. FIND-SET therefore takes $\Omega(log(n))$ on those nodes. If we set $m \geq 3n$ we have $> \frac{m}{3}$ FIND-SET's, so the total cost is $\Omega(mlog(n))$.

Problem 3a. By induction, if $x = 1$ then [2] it has rank $0 = \lfloor log(1) \rfloor$. Assume it holds for $1...x$ nodes.

Given $x + 1$ nodes, we perform a UNION on two disjoint sets with y and z nodes each, where $y, z \leq x$. The root of the first set then has rank at most $\lfloor log(y) \rfloor$ and the second has rank $\lfloor log(z) \rfloor$.

Finally, we assume the ranks are not equal, because if they were the UNION would preserve rank and we would be done. Assuming unequal ranks, the rank of the UNION increases by 1 and the resulting set has rank $\lfloor log(y) \rfloor + 1 \leq \lfloor log(x+1)/2 \rfloor = \lfloor log(x+1) \rfloor$

Colloquially we can say that the rank increases by 1 only when the ranks of x and y are equal. Therefore, we would need twice the number of elements to increase the rank by 1, which is $\lfloor log(x) \rfloor$.

Problem 3b. Since each value is at most $\lfloor log(n) \rfloor$, then for $n \to \infty$ it would take $\Theta(log(log(n)))$ bits.

Problem 4a. The worst case running time of MultiPopA and MultiPopB occur when you have to pop off the entire stack of each. So, worst case, MultiPopA runs in $O(n)$ while MultiPopB runs in $O(m)$. The worst case running time for Transfer occurs when you have to pop all the elements off of stack A and push them on to stack B. Here you have $O(n) + O(n) = O(n)$.

Problem 4b. From [6] we know the conditions for an appropriate potential function are such that if $\Phi(s_0) \leq \Phi(s_n)$, we get $\sum_i c_i \leq \sum_i ac_i$. It must also be in the form $an + bm$. We also know that $\Phi(s_0)$ must equal 0 for the initial state of the data structure, and that $\Phi(s_i)$ must be ≥ 0 for all subsequent states. Thus, the potential function must always grow, though not by too much. To accomplish this, I have chosen the potential function to be $\Phi(n, m) = 3n + m$.

Problem 4c. To prove these we must solve for $\hat{c_i} = c_i + \Delta(\Phi(S))$ or $\hat{c_i} = c_i + \Phi(S_{i+1}) - \Phi(S_i)$.

- MultiPopA

  $\hat{c_i} = k + 3(n - k) + m - (3n + m) = -2k$

- MultiPopB

  $\hat{c_i} = k + 3n + (m - k) - (3n + m) = 0$

- Transfer

  $\hat{c_i} = 2k + 3(n - k) + (m + k) - (3n + m) = 0$

Here we can see that the amortized cost is bounded by a constant, so the overall run time for all operations is $O(1)$.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* The MIT Press, 4 ed., 2022.

[2] "Walkccc." `https://walkccc.me/CLRS`. Accessed on 2022-10-22.

[3] "Wikipedia dynamic arrays." `https://en.wikipedia.org/wiki/Dynamic_array`. Accessed on 2022-10-22.

[4] "Dynamic array amortized analysis." `https://www.interviewcake.com/concept/java/dynamic-array-amortized-analysis`. Accessed on 2022-10-22.

[5] "Tutorial 4." `https://fileadmin.cs.lth.se/cs/Personal/Rolf_Karlsson/tut4.pdf`. Accessed on 2022-10-22.

[6] "Notes on amortization." `https://www.cs.cmu.edu/~15451-s15/LectureNotes/lecture06/sleator-notes.pdf`. Accessed on 2022-10-22.