Algorithms Desgin
Homework 4 Solutions

November 30, 2020

1 Amortized Analysis

Use a singly linked list to store those integers. To implement `INSERT(x, S)`, we append the new integer to the end of the linked list. This takes $\Theta(1)$ time. To implement `REMOVE-BOTTOM HALF(S)`, we use the median finding algorithm taught in class to find the median number, and then go through the list again to delete all the numbers smaller or equal than the median. This takes $\Theta(n)$ time.

Suppose the runtime of `REMOVE-BOTTOM-HALF(S)` is bounded by cn for some constant c. For amortized analysis, use $\Phi = 2cn$ as our potential function. Therefore, the amortized cost of an insertion is $1 + \Delta\Phi = 1 + 2c = \Theta(1)$. The amortized cost of `REMOVE-BOTTOM-HALF(S)` is $cn + \Delta\Phi = cn + (-2c \times \frac{n}{2}) = 0$.

2 Operations Sequence

Answer is 3. Two ways of analyzing are:

- summing up the totals and dividing by n. In this case, the powers of 2 give at most $1 + 2 + 4 + 8 + ... + n < 2n$, and the non-powers-of-2 give $n - \log(n) < n$, so the total is $< 3n$, or amortized 3 per operation.
- the banking method: if every operation since the last power of 2 pays 2$ extra, that produces enough money to pay for the expensive operation. So, in this accounting, the cheap operations pay 3$, and the expensive operations pay 2$, for an *amortized cost* $< 3$$.

3 Stack with Array

Consider the interval between successive array resizes. The important point is that after each resize operation, the array is exactly half full.

- If it's a double, then there must have been at least $\frac{L}{2}$ pushes since the last resize. This can pay for work.
- If it's a halving, then there must have been at least $\frac{L}{4}$ pops, and these can be used to pay for the resize.

First, we define our load factor α as follows: $\alpha = \frac{k}{l}$ The potential function will be:

- $\alpha \geq \frac{1}{2} \Rightarrow \frac{k}{l} \geq \frac{1}{2} \Rightarrow 2k - l \geq 0 \Rightarrow \Phi = 2k - l \geq 0$
- $\alpha < \frac{1}{2} \Rightarrow \frac{k}{l} < \frac{1}{2} \Rightarrow \frac{l}{2} - k > 0 \Rightarrow \Phi = \frac{l}{2} - k > 0$

So we have:

$$\Phi = \begin{cases} \frac{l}{2} - k & \text{if } \alpha < \frac{1}{2} \\ 2k - l & \text{if } \alpha \geq \frac{1}{2} \end{cases}$$

Now we define c_i as cost of the i^th operation, D_i as the state of our data structure after the i^th operation and c_t as the amortized cost of the operation as follows:

$$c_t = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

We have $k = \frac{l}{4}$ and:

$$\alpha_{i-1} < \frac{1}{2} \Rightarrow \Phi(D_{i-1}) = \frac{l_{i-1}}{2} - k_{i-1}$$
$$\alpha_i = \frac{1}{2} = \frac{k_i}{l_i} \Rightarrow \Phi(D_i) = \frac{l_i}{2} - k_i$$
$$k_i = k_{i-1} - 1$$
$$l_i = \frac{l_{i-1}}{2}$$
$$c_i = k_i + 1$$
$$c_t = c_i + \Phi(D_i) - \Phi(D_{i-1})$$
$$\Rightarrow c_t = k_i + 1 + (\frac{l_i}{2} - k_i) - (\frac{l_{i-2}}{2} - k_i)$$
$$\Rightarrow c_t = 2k_i - l_i + 2 = O(1)$$

4 Pearl Data Structure

Recall that for any given operation op_i, we have $cost_{am}(op_i) = cost_{real}(op_i) + \Phi(D_i) - \Phi(D_{i-1})$ where $\Phi(D_i)$ is the potential of the pear after i operations have been performed. We start by computing the amortized cost of each operation.

- insert: The amortized cost of insert is $\log n + \log n = 2 \log n$.
- findSmallestGap: The amortized cost of findSmallestGap is $x + \log^2 n - (x - 2) = 2 + \log^2 n$.
- removeElement: The amortized cost of removeElement is $\log n + 3$.

Hence the worst-case running time of a sequence of n operations is:

$$\begin{aligned}\sum_{i=1}^{n} cost_{real}(operation_i) &\leq \sum_{i=1}^{n} cost_{amortized}(operation_i) \\ &\leq \sum_{i=1}^{n} max\{2\log n, 2+\log^2 n, 3+\log n\} \\ &\leq \sum_{i=1}^{n}(2+\log^2 n) \\ &\leq n(2+\log^2 n)\end{aligned}$$

So it is in $O(n\log^2 n)$.