10/11/2016

Algorithms

HW - 4

Kiran Shettar
ID - 01605800

Problem 1 17.1-1 (pg ~~433~~ 456)

If the set of stack operations included a MULTIPUSH operation, which pushes 'k' items onto the stack, would the $O(1)$ bound on the amortized cost of stack operations continue to hold?

Solution: In the case of MULTIPUSH operation the $O(1)$ bound on the amortized cost of stack operations does not hold.

Let us consider the following scenario:

* Push : $O(1)$
* Pop : $O(1)$
* MULTIPUSH : $O(k)$
* MULTIPOP : $O(k)$

In case of MULTIPOP of 'n' operations, the amortized cost was $O(n)$.

$$\therefore \text{Average} = \frac{O(n)}{n} = O(1)$$

For multipush, the cost will be $O(n^2)$

We could also consider $O(kn)$. So that the average cost of n-operations is $O(n)$ or $O(k)$ which is not bound on $O(1)$ as in the case of only multipop operation.

Problem 2 17.4-3 (pg 471)

Suppose that instead of contracting a table by halving its size when its load factor drops below 1/4, we contract it by multiplying its size by 2/3 when its load factor drops below 1/3. Using the potential function

$$\Phi(T) = |2 \cdot T.num - T.size|,$$

show that the amortized cost of a ~~tadate~~ TABLE-DELETE that uses this strategy is bounded above by a constant.

$$\alpha(T) = \frac{num_i}{size_{i-1}}$$

Case 1: Table contradiction occurs,

i.e when $\alpha < 1/3$

Case 2: $c_i = T.num(1) + 1$

$$\hat{c}_i = c_i + \Delta\phi$$

$$= c_i + \phi_i - \phi_{i-1}$$

i.e when table contradiction does not occurs. $\alpha \geq 1/3$

$$T.size(i) = \frac{2}{3} \cdot T.size(i-1)$$

Considering the "table-delete" operation, the number of elements reduce by 1.

$$T.num(i) = T.num(i-1) - 1$$

So, we reduce the table if $\alpha < 1/3$

when $\alpha = 1/3 \rightarrow$ ~~E~~ $T.num(i-1)$

$$= T.size(i-1)/3$$

i.e $\frac{\cancel{3}}{2} \times \frac{T.size(i)}{\cancel{3}}$

$$\therefore T.num(i) + 1 = \frac{T.size(i)}{2} = T.num(i-1)$$

Hence, $\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$

$$\Rightarrow (T.num(i) + 1) + [2 . T.num(i) - T.size(i)]$$
$$- [2 . T.num(i-1) - T.size(i-1)]$$

$$= T.num(i) + 1 + [2 . (T.num(i) + 1) - T.size(i) - \frac{3}{2} . T.size(i)]$$

$\Rightarrow T.num(i) + 1 + \left| 2 \cdot T.num(i) - 2 \cdot Tnum(i) - 2 \cdot Tnum(i) - 2 \right| - \left| T.size(i) - \frac{3}{2} \cdot T.size(i) \right|$

$\Rightarrow T.num(i) + 1 + |-2| - \left| -\frac{T.size(i)}{2} \right|$

$\Rightarrow T.num(i) + 1 + 2 - T.num(i) - 1$

$\Rightarrow 2$ // $\boxed{\therefore \hat{C}_i = 2}$ //

Problem 3 17-2 (Pg 473)

(a) Using binary search to search each sorted array one by one, until find the element that we search

The worst case running time will be as follows:

$$T(n) = \Theta(\log 1 + \log 2 + \log 2^2 + \ldots + \log 2^{k-1})$$

$$= \Theta(0 + 1 + \ldots + (k-1))$$

$$= \Theta\left(\frac{1}{2} k(k-1)\right)$$

$$= \Theta(\log^2 n) //$$

which is a worst case run time.

(b) Performing an <u>INSERT</u> operation:

Firstly we have to create a new sorted array of size 1. Let there be one more array that has a new element that is to be inserted.

If the first array is empty then we replace it with a new array. Else we will merge the array 1 & array 2 to create a new sorted array 3.

We repeat this step till we do it for all the elements.

The amortized running time of the worst case:

<u>Accounting method</u>: To insert an element it'll take 'k'. And we put (k-1) on the inserted ~~from to pay for it~~ element which is involved & it merges later on.

And this can move at the maximum of (k-1) times. ∴ Run time amortized : O(log n)

Aggregation method:

While sorting in the worst case we move all the $n+1$ elements.

For a sequence of 'n' insertions in the array, it will be changed $n/2^i$ times.

$\therefore$ The total cost of 'n' operations will be

$$\sum_{i=0}^{k-1} \frac{2^i n}{2^i} = \sum_{i=0}^{k-1} n = nk \in O(n \lg n)$$

$\therefore$ The amortized cost will be $O(\log n)$ per insertion.

(c) There's no better way to implement ~~delete~~ DELETE which is better than linear time. Let us consider that we delete an element from the middle in the largest array. Since each deletion take $O(n)$ time, the amortized cost is also $O(n)$.

Let us consider array A_i & find smallest element in the array & the item to be deleted which is in ~~arr~~ A_j. Remove item from A_j & move an element from A_i to A_j & leave it if $i=j$. And rearrange the whole array into sorted array by breaking it into parts. Hence, INSERT & DELETE operation result in amortize cost no better than worst case cost.