Lab07-Amortized Analysis

CS214-Algorithm and Complexity, Xiaofeng Gao & Lei Wang, Spring 2021.

* Name: Haoyi You Student ID: 519030910193 Email: yuri-you@sjtu.edu.cn

1. Suppose we perform a sequence of n operations on a data structure in which the i th operation costs i if i is an exact power of 2, and 1 otherwise. Use an accounting method to determine the amortized cost per operation.

   Solution. Let $f(n)$ be the total cost of n operations.

   $$f(n) = \sum_{i=1}^{\lfloor logn \rfloor} 2^i + (n - \lfloor logn \rfloor) \tag{1}$$

   So amortized cost per operation $g(n)$ is

   $$g(n) = \frac{f(n)}{n} = \frac{2^{\lfloor logn \rfloor + 1} - 1 + n - \lfloor logn \rfloor}{n} \leq 3 \tag{2}$$

   □

2. Consider an ordinary **binary min-heap** data structure with n elements supporting the instructions INSERT and EXTRACT-MIN in $O(\log n)$ worst-case time. Give a potential function Φ such that the amortized cost of INSERT is $O(\log n)$ and the amortized cost of EXTRACT-MIN is $O(1)$, and show that it works.

   Solution. Assume C_i is the cost of the i^{th} operation, $\hat{C}_i = \Phi(S_i) - \Phi(S_{i-1})$.
   Let $\Phi(S_i) = n * log(n+1)$ with n is the number of elements in binary min-heap.

   (a) $\forall n > 0, n * log(n+1) - 0 * log0 > 0$.

   (b) For INSERT operation.

   $$\begin{aligned} \hat{C}_i &= log(n) + n * log(n+1) - (n-1)log(n) \\ &= n * log(n+1) - (n-2) * log(n) \\ &= 2log(n) + n * log(1 + \frac{1}{n}) \\ &\leq 2log(n) + n * (1 + \frac{1}{n} - 1) \\ &= 2log(n) + 1 \end{aligned} \tag{3}$$

   So the amortized cost of INSERT is O(log(n)).

   (c) For EXTRACT-MIN operation.

   $$\begin{aligned} \hat{C}_i &= log(n) - (n-1)log(n) + (n-2)log(n-1) \\ &\leq (n-1)log(1 + \frac{1}{n-1}) \\ &= 1 \end{aligned} \tag{4}$$

   So the amortized cost of EXTRACT-MIN is O(1).

(d) The amortized cost analysis is based on the fact that if we take EXTRACT-MIN at n elements, we must have taken INSERT at $n-1$ elements before.

□

3. Assume we have a set of arrays $A_0, A_1, A_2, \cdots$, where the i^{th} array A_i has a length of 2^i. Whenever an element is inserted into the arrays, we always intend to insert it into A_0. If A_0 is full then we pop the element in A_0 off and insert it with the new element into A_1. (Thus, if A_i is already full, we recursively pop all its members off and insert them with the elements popped from $A_0, ..., A_{i-1}$ and the new element into A_{i+1} until we find an empty array to store the elements.) An illustrative example is shown in Figure **??**. Inserting or popping an element take $O(1)$ time.

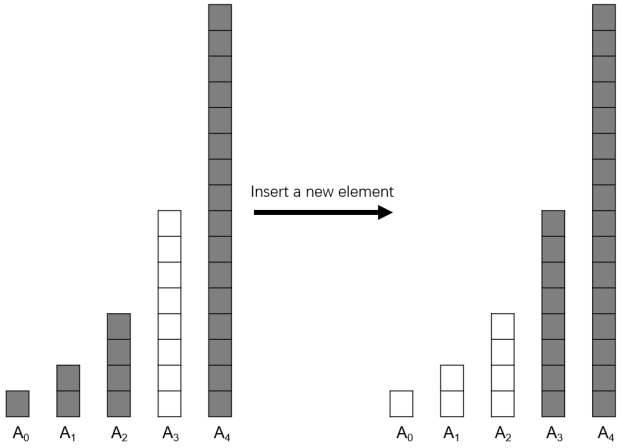

图 1: An example of making room for one new element in the set of arrays.

(a) In the worst case, how long does it take to add a new element into the set of arrays containing n elements?

(b) Prove that the amortized cost of adding an element is $O(\log n)$ by *Aggregation Analysis*.

(c) If each array A_i is required to be sorted but elements in different arrays have no relationship with each other, how long does it take in the worst case to search an element in the arrays containing n elements?

(d) What is the amortized cost of adding an element in the case of (c) if the comparison between two elements also takes $O(1)$ time?

Solution. (a) If $n = 2^i - 1$,this time A_0 A_{i-1} is full, it costs $\sum_{j=0}^{i-1} 2^j + 2^i = 2n - 1$ times.

(b) First we prove at any state, $\forall i, A_i$ is either empty or full.

i. We assume there are k elements in total.

ii. When $k = 1, A_0$ is full,others is empty.

iii. Assume when $k = n - 1$ the hypothesis is correct.

iv. This time we want to add a new element.Assume j to be the minimum number that A_i is empty at $k = n - 1$. When we add the element,we find A_0~A_{j-1} is full, we should pop them. And A_0~A_{j-1} become empty. Here there are 2^j elements need inserting, so A_j become full,which means when $k = n$,the hypothesis is still correct.

v. From the mathematical induction, $\forall n \in \mathbb{R}$, the hypothesis is correct.

Then we know $\forall n \in \mathbb{R}$, we can write n as $n = \sum_{i=0}^{n} b_i * 2^i$. So A_i is full iff $b_i = 1$.
If we add from 0 to n, the k^{th} bit of n changes $\lfloor \frac{n}{2^k} \rfloor$ times. Every time changing means making A_k from full to empty or from empty to full, which costs 2^k times.
So the amortized cost

$$C_n = (\sum_{i=0}^{\lfloor log(n) \rfloor} 2^i * \lfloor \frac{n}{2^i} \rfloor)/n = log(n) \tag{5}$$

So the amortized cost is $O(log(n))$.

(c) If the elements is in the last array or do not contained in the set of arrays.We must search from A_0~$A_{\lfloor log(n) \rfloor}$ and in each array we cost $log|A_i| = i$ times by binary search. So the total cost is

$$\sum_{i=0}^{\lfloor log(n) \rfloor} i = \frac{(\lfloor log(n) \rfloor)(\lfloor log(n) \rfloor + 1)}{2} \approx \frac{log^2 n}{2} \tag{6}$$

So the cost in worst case is $O(log^2 n)$.

(d) Based on the proof in (b), add an elements means change A_j from empty to full and change A_0~A_{j-1} from full to empty. So we need to pop A_0~A_{j-1} and insert them in A_j. The pop operation costs $\sum_{i=0}^{j-1} 2^i = 2^j - 1$. The insert operation we use merge sort and costs $O(2^j)$. So the total cost is $O(2^j)$ with j is changed from empty to full.
So the total cost add from 0 to n is

$$\sum_{i=0}^{\lfloor log(n) \rfloor} O(2^i) * \lfloor \frac{n}{2^i} \rfloor) = O(nlog(n)). \tag{7}$$

So the amortized cost is $O(nlog(n))/n = O(log(n))$.

□

Remark: Please include your .pdf, .tex files for uploading with standard file names.