Lab07-Amortized Analysis

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

∗ If there is any problem, please contact TA Yihao Xie.

∗ Name: Jianxing Qin   Student ID: 519030910270   Email: qdelta@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. We assigned the amortized cost $\widehat{C_{op}} = 3$. **Proof**:

$\forall n \in \mathbb{N}^+$, suppose that $2^k \leq n < 2^{k+1}$ for $k \in \mathbb{N}$. Then the total cost of n operations will be:

$$n - (k+1) + \sum_{i=0}^{k} 2^i = n + 2^{k+1} - k \leq 3n - k < 3n$$

□

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. Define

$$\Phi(S) = \begin{cases} 0, & n = 0 \\ \sum_{i=1}^{n} \log i, & n \geq 1 \end{cases}$$

Where n is the number of elements in the heap, then $\Phi(S_0) = 0 \leq \Phi(S)$.

If before operation, the number of elements in the heap is n, then:

- For Insert, the amortized cost is $\log n + \log(n+1) = O(\log n)$.
- For Extract-Min, the amortized cost is $\log n - \log n = O(1)$.

□

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 1. Inserting or popping an element take $O(1)$ time.

   (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?

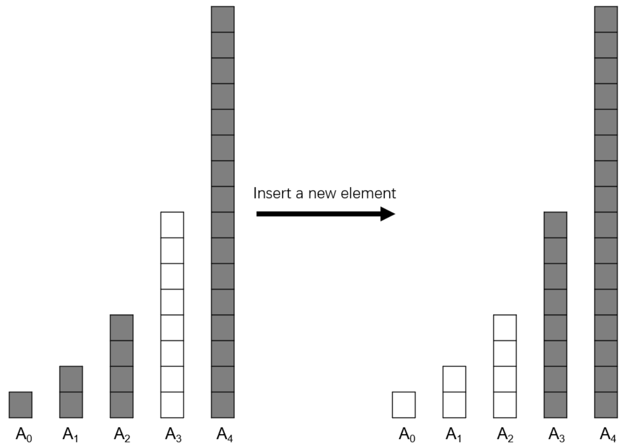

图 1: An example of making room for one new element in the set of arrays.

(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) The worst case happens if $n+1 = 2^{k+1}$, which means $A_0, A_1, \cdots, A_k$ are full and $A_k, A_{k+1}, \cdots$ are empty. Then $O(n)$ time is needed to move all elements to A_k if we insert a element.

(b) Suppose we perform n insertions where $2^k \le n < 2^{k+1}$. The cost of each insertion will be 2^i if i is the minimum index that A_i is empty before, so consider the total cost $T(n)$, we will get:

$$\begin{aligned} T(n) &< T(2^{k+1}) \\ &= \sum_{i=0}^{k} 2^i \cdot 2^{k-i} \\ &= (k+1)2^k \end{aligned}$$

So the amortized cost $\frac{T(n)}{n} < \frac{(k+1)2^k}{2^k} = k+1 \le 1 + \log_2 n = O(\log n)$

(c) The worst case happens if we have to search in all full arrays.

Let $n = 2^{k_1} + 2^{k_2} + \cdots + 2^{k_m}$ where $0 \le k_1 < k_2 < \cdots < k_m$, then the total cost will be (use binary search in each array):

$$\sum_{i=1}^{m} k_i \le \frac{k_m(k_m+1)}{2} \le \frac{\log_2 n(1+\log_2 n)}{2} = O(\log^2 n)$$

When $k_i = i-1$, then the total cost will be $\frac{k_m(k_m+1)}{2} = O(\log^2 n)$. From all above, the worst-case complexity is $O(\log^2 n)$.

(d) Suppose i is the minimum index that A_i is empty before insertion, then the best complexity of insertion will be $\Theta(2^i)$. **Proof:**

- Since all elements will be moved to A_i, the complexity will be $\Omega(2^i)$.
- Use the procedure MERGE from algorithm **Merge-Sort**. First we merge the element to insert with A_0 to generate B_1, then merge B_1 and A_1 to B_2, $\cdots\cdots$, finally merge B_{i-1} and A_{i-1} to A_i. The complexity will be:

$$\sum_{p=0}^{i-1}\left(1 + \sum_{q=0}^{p} 2^q\right) = \sum_{p=0}^{i-1} 2^{p+1} = 2^{i+1} - 2 = O(2^i)$$

If $2^k \leq n < 2^{k+1}$, consider the total cost $T(n)$, we will get:

$$
\begin{aligned}
T(n) &< T\left(2^{k+1}\right) \\
&= \sum_{i=0}^{k} 2^i \cdot 2^{k-i} \\
&= (k+1)2^k
\end{aligned}
$$

So the amortized cost $\frac{T(n)}{n} < \frac{(k+1)2^k}{2^k} = k+1 \leq 1+\log_2 n = O(\log n)$

$\square$

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