Lab07-Amortized Analysis

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

∗ If there is any problem, please contact TA Yihao Xie.
∗ Name: Zilong Li Student ID: 518070910095 Email: logcreative-lzl@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.

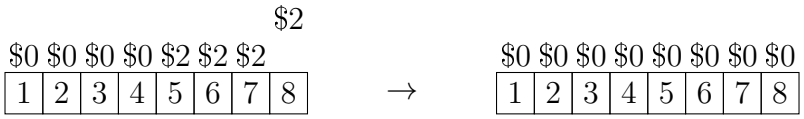
   Solution. Define C_i as the cost of the ith operation,

   $$C_i = \begin{cases} i, & \text{if } i \text{ is an exact power of 2,} \\ 1, & \text{otherwise} \end{cases}$$

   Case 1: i is not the power of 2. an amortized cost $\hat{C}_i = \$3$ is charged:

   - \$1 pays for itself;
   - \$2 is stored for later cost expanding, \$1 \$1 for paying one of the recent $\frac{i}{2}$ items, \$1 for paying one of the old $\frac{i}{2}$ items.

   Case 2: i is the power of 2. an amortized cost $\hat{C}_i = \$2$ is charged, in order not to have leftouts.

   Thus,

   $$T(n) = \sum_{i=1}^{n} C_i \leq \sum_{i=1}^{n} \hat{C}_i = 3n - \lfloor \log_2 n \rfloor$$

   □

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. Let Φ be the sum of the cost on every node

   $$\Phi(S_i) = \sum_{k=1}^{i} \log_2 k$$

   which meets the requirement of a potential function:

   $$\Phi(S_n) \geq \Phi(S_0) = 0$$

   because it has to be the nonnegative integer.

   Describe Insert and Extract-Min for binary min-heap data as follows:

INSERT

(a) Insert the element to the end of the heap.

(b) PERLOCATE-UP from the insertion position.

EXTRACT-MIN

(a) Extract the minimum element.

(b) Put the last element to the root.

(c) PERLOCATE-DOWN from root.

Because both PERLOCATE-UP and PERLOCATE-DOWN take $\log_2 n = O(\log n)$, so both the algorithms cost $C_i = O(\log n)$.

As for the amortized analysis, the potential function is considered: INSERT The number of element will be increased by 1.

$$\begin{aligned}\hat{C}_i &= C_i + \Phi_i - \Phi_{i-1} \\ &= 1 + \log_2 i + \log_2 i = 2\log_2 i + 1\end{aligned}$$

EXTRACT-MIN The number of element will be decreased by 1.

$$\hat{C}_i = C_i + \Phi_i - \Phi_{i+1} = 2 + \log_2 i - \log_2(i+1) = 2 + \log_2 \frac{i}{i+1} < 2$$

So the amortized cost for INSERT is $O(\log n)$ and for EXTRACT-MIN is $O(1)$.

□

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.

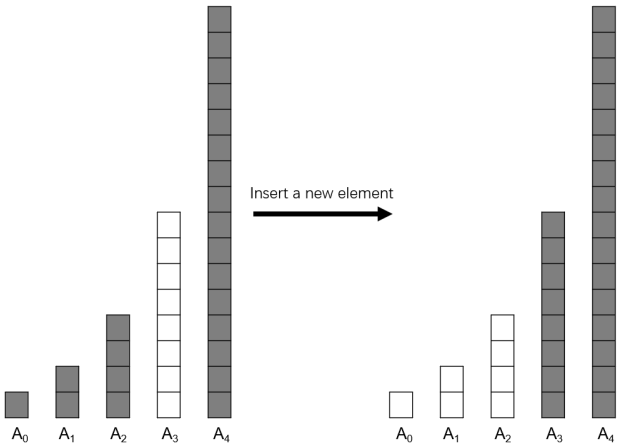


Figure 1: An example of making room for one new element in the set of arrays.

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

Solution. The worst case comes to that n is the power of 2, assumming it to be $n = 2^i$. All elements from $A_0, \cdots, A_{i-1}$ has to be popped and pushed into A_i, which is the last column. The inserted element will be inserted into A_i as well.

$$T(n) = 2(1 + \cdots + 2^{i-1}) + 1 = 2^{i+1} - 1 = 2n - 1 = \Omega(n)$$

□

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

Solution. Enumerate the cost of different n:

i	1	2	3	4	5	6	7	8
C_i(new push)	1	1	1	1	1	1	1	1
C_i(pop+push)		2		6		2		14
i	9	10	11	12	13	14	15	16
C_i(new push)	1	1	1	1	1	1	1	1
C_i(pop+push)		2		6		2		30

Denote $S(n)$ as the sum of operations for inserting n elements. The recurrence of $S(n)$ is:

$$\begin{aligned} S(1) &= 1 \\ S(2^i - 1) &= 2S(2^{i-1} - 1) + T(2^{i-1}) \\ S(2^i) &= S(2^i - 1) + T(2^i) \end{aligned}$$

Because the worst case comes to $n = 2^i$, the aggregation analysis will provide the upper bound reflected by the case of $\frac{S(2^i)}{2^i}$.

$$\begin{aligned} S(2^i) &= 2^{i+1} - 1 + S(2^i - 1) \\ &= 2^{i+1} - 1 + 2S(2^{i-1} - 1) + T(2^{i-1}) \\ &= 2^{i+1} - 1 + 2^{i-1}S(1) + 2^{i-2}T(2^1) + \cdots + 2^1 T(2^{i-2}) + T(2^{i-1}) \\ &= 2^{i+1} - 1 + 2^{i-1} + 2^i - 2^{i-2} + 2^i - 2^{i-3} + \cdots + 2^i - 1 \\ &= 2^{i+1} + (i-1)2^i \end{aligned}$$

By *Aggregation Analysis*, the amortized cost of adding an element is

$$\hat{C}_i = \frac{S(n)}{n} = \frac{S(2^i)}{2^i} = i + 1 = \log_2 n + 1 = O(\log n)$$

□

(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?

Solution. Binary Search in each A_i.

$$T(n) = 1 + 2 + \cdots + \log(n+1) - 1 = \frac{1}{2}\log(n+1)(\log(n+1) - 1) = O(\log^2 n)$$

If the array is sorted, then the search algorithm is as follows.

Algorithm 1: The searching algorithm for sorted arrays

Input: set of arrays $A_0, A_1, \cdots, A_i$, the element x to be searched
Output: the searching result

```
for j ← 0 to i do
    if A_j is not empty and (head of A_j) ≤ x ≤ (tail of A_j) then
        foreach y in A_j do
            if x = y then return (j, position of y in A_j) ;
return No such an element is found;
```

In the worst case, the element is in the last array to be searched and near the tail of the array. And the element is in the range of every array. In other word, the number of elements satisfies

$$n = 2^0 + 2^1 + \cdots + 2^i = 2^{i+1} - 1$$

the number of arrays satisfies

$$i = \log_2(n+1) - 1$$

The time complexity comes to

$$T'(n) = 2i + n = 2(\log_2(n+1) - 1) + n = \Omega(n)$$

□

(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?

i	1	2	3	4	5	6	7	8
C_i(new push)	1	1	1	1	1	1	1	1
C_i(pop+push)		2		6		2		14
C_i(sort)		$Q(3)$		$Q(7)$		$Q(3)$		$Q(15)$
i	9	10	11	12	13	14	15	16
C_i(new push)	1	1	1	1	1	1	1	1
C_i(pop+push)		2		6		2		30
C_i(sort)		$Q(3)$		$Q(7)$		$Q(3)$		$Q(31)$

Solution. Like Merge Sort. $O(\log n)$. Bit-Flip.

Enumerate the cost of different n with sorting, where sorting requires $Q(n) = O(n \log n)$ time complexity:
Because it starts from A_0, every pop will lead to a sort operation over the influnced elements. Let S' be the sum of new cost.

$$\begin{aligned} S'(1) &= 1 \\ S'(2^i - 1) &= 2S'(2^{i-1} - 1) + T(2^{i-1}) + Q(T(2^{i-1})) \\ S'(2^i) &= S'(2^i - 1) + T(2^i) + Q(T(2^i)) \end{aligned}$$

By the same *aggregation analysis* process, denote

$$R(n) = T(n) + Q(T(n)) = 2n - 1 + O((2n-1)\log(2n-1))$$

Then, in the worst case sum,

$$\begin{aligned} S'(2^i) &= 2^{i+1} - 1 + 2^{i-1}S'(1) + 2^{i-2}R(2^1) + \cdots + 2^1 R(2^{i-2}) + R(2^{i-1}) \\ &= S(2^i) + \sum_{j=0}^{i-2} 2^j R(2^{i-1-j}) \\ &= S(2^i) + (2^i - 1)\sum_{j=0}^{i-2} O(\log(2^{i-j} - 1)) \\ &= S(2^i) + O\left((2^i - 1)\frac{(i+2)(i-1)}{2}\right) \end{aligned}$$

So the new amortized cost is

$$\begin{aligned} \hat{C_i}' = \frac{S'(n)}{n} = \frac{S'(2^i)}{2^i} &= \hat{C}_i + \frac{(2^i - 1)O(i^2)}{2^i} \\ &= O(\log n) + O((\log n)^2) \\ &= O((\log n)^2) \end{aligned}$$

□

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