Lab07-Amortized Analysis

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

∗ If there is any problem, please contact TA Shuodian Yu.

∗ Name:Haotian Xue   Student ID:518021910506   Email: xavihart@sjtu.edu.cn

1. For the TABLE-DELETE Operation in Dynamic Tables, suppose we construct a table by multiplying its size by $\frac{2}{3}$ when the load factor drops below $\frac{1}{3}$. Using *Potential Method* to prove that the amortized cost of a TABLE-DELETE that uses this strategy is bounded above by a constant.

   Solution. The solution for this problem is as follows.

   Since we only need to calculate the amortized cost of TABLE-DELETE operation, we do not need to take into account the expansion of the table.

   We can set the potential function as $\Phi_i = s_i - n_i$, where n_i and s_i are the number of elements in the table and the size of table after the ith operation respectively. $s_0 - n_0 = 0$ and for any $i \in \mathbb{N}$, $\Phi_i \geq 0$.

   Then we do the delete operation:

   - **Case1**: The contraction is not triggered.
     Then we have: $n_i = n_{i-1} - 1$, $s_i = s_{i-1}$, and the amortized cost is:

$$\begin{aligned}\hat{C}_i &= C_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (s_i - n_i) - (s_{i-1} - n_{i-1}) \\ &= 1 + (s_i - s_{i-1}) + (n_{i-1} - n_i) = 2\end{aligned}$$

   - **Case2**: The contraction is triggered.
     Then we have: $n_i = n_{i-1} - 1$, $s_i = \frac{2}{3}s_{i-1}$, $n_i = \frac{1}{3}s_{i-1}$, and the amortized cost is:

$$\begin{aligned}\hat{C}_i &= C_i + \Phi_i - \Phi_{i-1} \\ &= n_i + 1 + (s_i - n_i) - (s_{i-1} - n_{i-1}) \\ &= n_i + (s_i - s_{i-1}) + (n_{i-1} - n_i) \\ &= n_i - \frac{1}{3}s_{i-1} + 1 = 1\end{aligned}$$

   Considering the two cases above, we can safely get the amortized cost for DELETE-TABLE operation is $O(1)$.

   P.S. Actually, we can use many other potential functions and their universal form may be $s_i + kn_i$, where $k \in \{x|x \geq -1, x \in \mathbb{Z}\}$. That is because n_i always satisfies $n_i = n_{i-1} - 1$.

   □

2. A **multistack** consists of an infinite series of stacks $S_0, S_1, S_2, \cdots$, where the i^{th} stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first pop all the elements off S_i and push them onto stack S_{i+1} to make room. (Thus, if S_{i+1} is already full, we first recursively move all its members to S_{i+2} .) An illustrative example is shown in Figure **??**. Moving a single element from one stack to the next takes $O(1)$ time. If we push a new element, we always intend to push it in stack S_0.

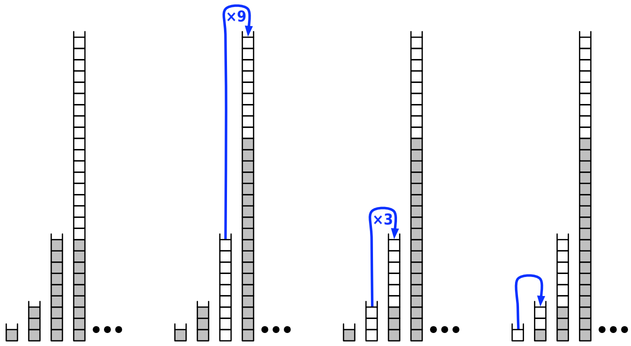

Fig. 1. An example of making room for one new element in a multistack.

(a) In the worst case, how long does it take to push a new element onto a multistack containing n elements?

(b) Prove that the amortized cost of a push operation is $O(\log n)$ by *Aggregation Analysis*.

(c) (Optional Subquestion with Bonus) Prove that the amortized cost of a push operation is $O(\log n)$ by *Potential Method*.

Solution. The solution for this problem is as follows.

We first define the **depth** of an element in a stack : if element a is in Si, then the depth of a is $d(a) = i + 1$. Also we note the cost for inserting the ith element is C_i and the total cost spent on specific element a to get the current situation is $CT(a)$.

(a) In the worst case, the S_0 to S_k are all full so we need to move every stack to make room for the inserted element, so we have $n = 1 + 3^2 + ... + 3^i$, the cost for inserting an element now is : $C_{n+1} = 1 + 1 + 3 + 3^2 + ... + 3^k = n + 1$, $k = O(\log n)$. So the cost is $O(n)$ in worst cases.

(b) To get the amortized cost for each insert operation, we assume now S_0 to S_i is full and we should calculate the total cost $T(n)$ for inserting $n = 1 + 3 + ... + 3^i$ elements.

$$T(n) = \sum_{i=1}^{n} C_i = \sum_{a} CT(a)$$

The equation above transfer calculating the sum of cost for each step into calculating the sum of cost for each element, and we actually have:

$$CT(a) = d(a)$$

So we can get the total cost by tranversing each stacks since elements in the same stack share the same depth:

$$T(n) = \sum_{i=0}^{k} 3^i * (i + 1) = O(k3^k) = O(n \log n)$$

The amortized cost is $C = \frac{T(n)}{n} = O(\log n)$.

(c) We define N_i^n as the number of elements in S_i after n push operation. And we define $D(n)$ to be the maximum depth of elements after n push operations. Also we make $f(n) = \log(2n + 1)$ and it is obvious that: $D(n) < f(n)$.

To get the amortized cost, we define the potential function as:

$$\Phi_n = \sum_{i=1}^{D(n)} N_i^n (f(n) - i)$$

$\Phi_0 = 0$, $\Phi_i (i = 1, 2, ..)$ is always positive since $D(n) < f(n)$.
Then we have $\hat{C_n} = C_n + \Phi_n - \Phi_{n-1}$:

$$\begin{aligned}\hat{C_n} &= C_n + \sum_{i=1}^{D(n)} N_i^n (f(n) - i) - \sum_{i=1}^{D(n-1)} N_i^{n-1} (f(n-1) - i) \\ &= (f(n) \sum_{i=1}^{D(n)} N_i^n - f(n-1) \sum_{i=1}^{D(n-1)} N_i^{n-1}) + (\sum_{i=1}^{D(n-1)} i N_i^{n-1} - \sum_{i=1}^{D(n)} i N_i^n + C_n) \\ &= (nf(n) - (n-1)f(n-1)) + C_n + P_n = O(\log n) + (P_n + C_n)\end{aligned}$$

We can see $C_n + P_n$ from the angle of elements, we want to calculate the **effect of one element** a and then tranverse all of the elements, that is:

$$C_n + P_n = \sum_a c_a - \sum_a \Delta N_a = \sum_a (c_a - \Delta N_a)$$

where ΔN_a is the effect a has caused to the change of P_n. Then there are two cases for states of a:

- **Case 1:** a is not moved from one stack to another.
  Then we have $c_a - \Delta N_a = 0 - 0 = 0$.
- **Case 1:** a is not moved from one stack to another.
  Suppose we move a from S_i to S_{i+1} ,then we have $c_a - \Delta N_a = 1 - (i \times (0 - 1) + (i + 1) \times (1 - 0)) = 0$.

So finally we have: $\hat{C_n} = O(\log n)$. Which in fact prove the amortized cost to be $O(\log n)$.

□

3. Given a graph $G = (V, E)$, and let V' be a strict subset of V. Prove the following propositions.

   (a) Let T be a minimum spanning tree of a G. Let T' be the subgraph of T induced by V', and let G' be the subgraph of G induced by V'. Then T' is a minimum spanning tree of G' if T' is connected.

   (b) Let e be a minimum weight edge which connects V' and $V \setminus V'$. There exists a minimum weight spanning tree which contains e.

Solution. We can prove these two problems by contradction.

(a) If $T^{'}$ is not the MST of $G^{'}$, then there must exists one subtree $T^{''}$ of T which is the MST of $G^{'}$ and the sum of weights of $T^{''}$ is less than that of $T^{'}$. We think about $T_0 = (T \setminus T^{'}) \cup T^{''}$

which is a spanning tree for G. However the total weights of T_0 is less than that of T, which contradicts the fact that T is the MST of G.

(b) If there is no MST that contains e, then we just pick one MST T which satisfies $e \notin T$. Then we add e to T to get $T_0 = \{e\} \cup T$. It is obvious that it will generate a cycle $C \subset T_0$ and $e \in C$. If all vertex of edges in $C - \{e\}$ is in either $V^{'}$ or $V \setminus V^{'}$, then we note $u \in V^{'}, v \in V \setminus V^{'}$ are the two vertexes of e, if we start from u then we can only get to vertexes in V, such that we cannot get to v. So there must exists another edge e_0 in the cycle which connects $V^{'}$ and $V \setminus V^{'}$. If we replace e_0 with e in T to get $T^{'} = T \setminus \{e_0\} \cup \{e\}$, then $T^{'}$ is obviously a spanning tree and its weight is less than that of T, which is contradictory with the fact that T is a MST.

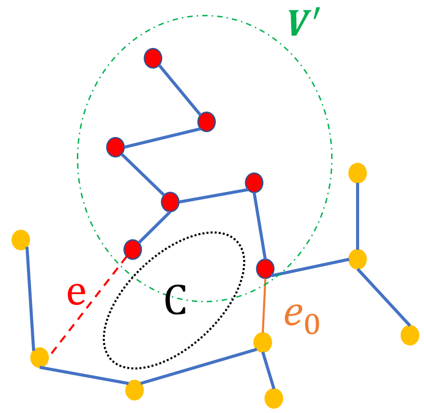

Fig. 2. An illustration for (b) in Problem3.

$\square$

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