Design and Analysis of Algorithms, Fall 2014
Exercise IV: Solutions

IV-1 (**CLRS 17.1-1**) 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?

No. The time complexity of such a series of operations depends on the number of pushes (pops vise versa) could be made. Since one MULTIPUSH needs $\Theta(k)$ time, performing n MULTIPUSH operations, each with k elements, would take $\Theta(kn)$ time, leading to amortized cost of $\Theta(k)$.

IV-2 (**CLRS 17.1-3**) Suppose we perform a sequence of n operations on a data structure in which the ith operation costs i if i is an exact power of 2, and 1 otherwise. Use aggregate analysis to determine the amortized cost per operation.

In a sequence of n operations there are $\lfloor \log_2 n \rfloor + 1$ exact powers of 2, namely $1, 2, 4, \ldots, 2^{\lfloor \log_2 n \rfloor}$. The total cost of these is a geometric sum $\sum_{i=0}^{\lfloor \log_2 n \rfloor} 2^i = 2^{\lfloor \log_2 n \rfloor + 1} - 1 \leq 2^{\log_2 n + 1} = 2n$. The rest of the operations are cheap, each having a cost of 1, and there are less than n such operations. The total cost of all operations is thus $T(n) \leq 2n + n = 3n = O(n)$, which means $O(1)$ amortized cost per operation.

IV-3 (**CLRS 17.2-2** and **CLRS 17.3-2**) Redo the previous exercise using (a) an accounting method of analysis and (b) a potential method of analysis.

(a) The actual cost of the ith operation is

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

We assign the amortized costs as follows:

$$\hat{c}_i = \begin{cases} 2 & \text{if } i \text{ is an exact power of 2} \\ 3 & \text{otherwise.} \end{cases}$$

Now an operation, which is an exact power of 2 uses all previously accumulated credit plus one unit of its own amortized cost to pay its true cost. It then assigns the remaining unit as credit. On the other hand, if i is not an exact power of 2, then the operation uses one unit to pay its actual cost and assigns the remaining two units as credit. This covers the actual cost of the operation. We still have to show that there will always be enough credit to pay the cost of any power-of-2 operation. Clearly this is the case for the first operation ($i = 1$), which happens to be an exact power of two. Let now $j > 0$. After 2^{j-1}:th operation there is one unit of credit. Between operations 2^{j-1} and 2^j there are $2^{j-1} - 1$ operations none of which is an exact power of 2. Each assigns two units as credit resulting to total of $1 + 2 \cdot (2^{j-1} - 1) = 2^j - 1$ accumulated credit before 2^jth operation. This, together with one unit by its own, is just enough to cover its true cost. Therefore the amount of credit stays nonnegative all the time, and the total amorized cost is an upper bound for the total actual cost.

(b) We define the potential function as follows:

$$\Phi(D_0) = 0 \quad \text{and} \quad \Phi(D_i) = 2i - 2^{\lfloor \log_2 i \rfloor + 1} + 1 \text{ for } i > 0.$$

(Note that $\Phi(D_i)$ actually equals to the amount of credit after ith operation in previous exercise.) This potential is always nonnegative, so we have $\Phi(D_i) \geq 0 = \Phi(D_0)$ for all i. Now

- For i, which is an exact power of 2, the potential difference is

$$\Phi(D_i) - \Phi(D_{i-1}) = (2i - 2i + 1) - (2(i-1) - i + 1) = 2 - i$$

  Note that this also holds for $i = 1$. Thus, the amortized cost of ith operation is

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = i + 2 - i = 2.$$

- For i, which is not an exact power of 2, the potential difference is

  $$\Phi(D_i) - \Phi(D_{i-1}) = (2i - i + 1) - (2(i-1) - i + 1) = 2.$$

  Thus, the amortized cost of ith operation is

  $$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2 = 3.$$

As seen above, the amortized cost of each operation is $O(1)$.

IV-4 (**CLRS 17.3-4**) What is the total cost of executing n of the stack operations PUSH, POP, and MULTIPOP, assuming that the stack begins with s_0 objects and finishes with s_n objects?

Let Φ be the potential function that returns the number of elements in the stack. We know that for this potential function, we have amortized cost 2 for PUSH operation and amortized cost 0 for POP and MULTIPOP operations.

The total amortized cost is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0).$$

Using the potential function and the known amortized costs, we can rewrite the equation as

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{i=1}^{n} \hat{c}_i + \Phi(D_0) - \Phi(D_n) \\ &= \sum_{i=1}^{n} \hat{c}_i + s_0 - s_n \\ &\le 2n + s_0 - s_n,\end{aligned}$$

which gives us the total cost of $O(n + (s_0 - s_n))$. If $s_n \ge s_0$, then this equals to $O(n)$, that is, if the stack grows, then the work done is limited by the number of operations.

(Note that it does not matter here that the potential may go below the starting potential. The condition $\Phi(D_n) \ge \Phi(D_0)$ for all n is only required to have $\sum_{i=1}^{n} \hat{c}_i \ge \sum_{i=1}^{n} c_i$, but we do not need for that to hold in this application.)

IV-5 (**CLRS 17.4-3**) 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 $\Phi(T) = |2 \cdot N(T) - S(T)|$, show that the amortized cost of a TABLE-DELETE that uses this strategy is bounded above by a constant. Here $N(T)$ and $S(T)$ denote the number of items stored in table T and the size of T, respectively.

Let c_i denote the actual cost of the ith operation, $\hat{c}_i$ its amortized cost and n_i, s_i and Φ_i the number of items stored in the table, the size of the table and the potential after the ith operation, respectively. The potential Φ_i cannot get negative values and we have $\Phi_0 = 0$. Therefore $\Phi_i \ge \Phi_0$ for all i and the total amortized cost provides an upper bound on the actual cost.

Now if the ith operation is TABLE-DELETE, then $n_i = n_{i-1} - 1$. Consider first the case when the load factor does not drop below $1/3$, i.e. $\frac{n_i}{s_{i-1}} \ge \frac{1}{3}$. Then the table isn't contracted and $s_i = s_{i-1}$. We pay $c_i = 1$ for deleting one item. Thus

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + |2n_i - s_i| - |2n_{i-1} - s_{i-1}| \\ &= 1 + |2n_{i-1} - s_{i-1} - 2| - |2n_{i-1} - s_{i-1}| && n_i = n_{i-1} - 1,\ s_i = s_{i-1} \\ &\le 1 + |(2n_{i-1} - s_{i-1} - 2) - (2n_{i-1} - s_{i-1})| && \text{(reverse) triangle inequality} \\ &= 1 + 2 = 3\end{aligned}$$

Now consider the case when the load factor drops below $1/3$, i.e.

$$\frac{n_i}{s_{i-1}} < \frac{1}{3} \leq \frac{n_{i-1}}{s_{i-1}} \Rightarrow 2n_i < \frac{2}{3}s_{i-1} \leq 2n_i + 2. \tag{1}$$

Now we contract the table and have

$$s_i = \lfloor 2/3 \cdot s_{i-1} \rfloor \Rightarrow \frac{2}{3}s_{i-1} - 1 \leq s_i \leq \frac{2}{3}s_{i-1}. \tag{2}$$

By combining (1) and (2) we get $2n_i - 1 \leq s_i \leq 2n_i + 2$ and thus $|2n_i - s_i| \leq 2$. Furthermore, from (1) we get $s_{i-1} > 3n_i$ and thus $|2n_{i-1} - s_{i-1}| = -2n_{i-1} + s_{i-1} \geq n_{i-1}$. We pay $c_i = n_i + 1$ for deleting one item and moving the remaining n_i items into the contracted table. Then,

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (n_i + 1) + |2n_i - s_i| - |2n_{i-1} - s_{i-1}| \\
&\leq (n_i + 1) + 2 - n_i = 3
\end{aligned}$$

In both cases the amortized cost is at most 3 and thus bounded above by a constant.