Amortized O(1) Proof

Rishik Saha

August 20, 2023

I am going to use the Potential Method, which is a more general method to analyze trickier data structures. In potential method, We start with an initial data structure S_0 on which n operations are performed. $\forall i \in \mathbf{N} : i \leq n$ let c_i be actual cost of the i-th operation it changes the state from S_{i-1} to S_i. We define a potential function Φ which maps a state S_i to a real number number $\Phi(S_i)$, which is the potential associated with the state S_i.
Given a potential function Φ, the amortized cost $\hat{c_i}$ of the i-th operation is defined as:

$$\hat{c_i} = c_i + \Phi(S_i) - \Phi(S_{i-1})$$

The amortized cost is thus its actual cost plus the increase in potential. The total amortized cost of n operations therefore is:

$$\sum_{i=1}^{n} \hat{c_i} = \sum_{i=1}^{n} (c_i + \Phi(S_i) - \Phi(S_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(S_n) - \Phi(S_0)$$

The $\Phi(S_i)$ terms got telescoped. If we can define a Φ such that $\Phi(S_0) \leq \Phi(S_n)$, we have:

$$\sum_{i=1}^{n} c_i \leq \sum_{i=1}^{n} \hat{c_i}$$

Thus, if we can bound the amortized cost of each of the operations and final potential is at least as large as the initial potential, then total amortized cost is an upper bound on the total actual cost.

1 Stack expansion and contraction

The shrink utility is useful as it is nice if the size the stack occupies is not too much larger than necessary. Our stack, for the time being, has the following methods:

- Stack_B(): creates an empty stack with capacity 1024. (c=1024 and n=0)
- push(): if n = c then increase the capacity of the stack from c to 2c and push the element.
- pop(): if n = c/4 and $c \geq 2048$ then decrease the capacity of the stack from c to c/2 and erase the element.

The capacity of the stack is denoted by c and number of elements by n. And our cost model is:

- writing a value to any location costs 1
- moving a value from one location to another costs 1
- erasing a value costs 1
- all other operations are free

The situation immediately after a increase or decrease in capacity is that n = c/2. The key thing in this design is after one of these expensive operations, we are very far away from having to do another expensive operation. This gives time to build up its potential for the next expensive operation. If we change pop() to shrink when n = c/2, it can cause the amortized cost of an operation to be quite large.
Since we always have n=c/2 immediately follow a increase or decrease in capacity, it would be nice if the potential function is zero when n=c/2. And the potential should build as n/c increases to 1 or decreases to 1/4. Thus, we have the potential function:

$$\Phi(n,c) = \begin{cases} 2n - c, & \text{if } n \geq \frac{c}{2} or,\text{c=1024}, \\ \dfrac{c}{2} - n & \text{if } n < \frac{c}{2} \text{ \& } c > 1024. \end{cases}$$

The first case is the situation where stack is in increasing state, i.e. it is larger than it was after recent resize (or there was no resize at all) and heading towards a increase in capacity. The second case is the situation where stack is in decreasing state, i.e. it is smaller than it was after recent resize and heading towards a decrease in capacity.

2 Amortized Cost

Cost of a push operation: The actual cost is 1 and our potential has two cases: If $n \geq c/2$, n increases by 1 and thus the potential increases by 2. If $n < c/2$, potential decreases by 1. Therefore, the worst case potential increase is 2, thus the amortized cost of push (excluding increase in capacity) is at most 3.
Doubling of capacity(Stack Growth): Before growing, we have n = c and potential is n. After growing, potential is zero, so the potential decreases by n, which is exactly the cost of moving n elements to a new stack, and hence amortized cost is 0.
Cost of a pop operation: Removing an element costs 1, and n decreases by 1. If $n \geq c/2$, the potential decreases by 2. If $n < c/2$, potential increases by 1. So, the worst case potential increase is 1, and the amortized cost of pop (excluding decrease in capacity) is at most 2.
Halving of capacity(Stack Shrink): Before shrinking, we have c = 4n and initial potential n. After shrinking, potential is zero, and the potential decreases by n, which is exactly the cost of moving n elements to a new stack, and hence amortized cost of shrinking is 0.
We consider final and initial potential $\Phi(S_m) \geq 0$ and $\Phi(S_0) = -1024$ (when $c > 1024$). And when c=1024, Φ is increasing, thus $\Phi(S_m) \geq \Phi(S_0)$. So we can conclude the following

1. *Using the doubling-halving stack data structure, the amortized cost of push is at most 3 and the amortized cost of pop is at most 2, hence amortized time of push and pop is O(1).*