Data Structures and Algorithms - Homework 3

HK Rho

February 14, 2020

Problem 1

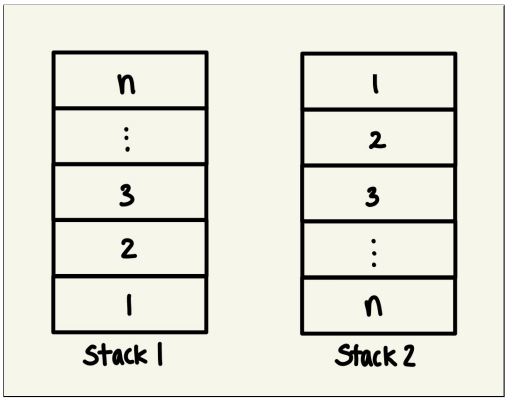

Figure 1: Implementation of a queue with two stacks. Stack 1 is used for enqueueing and Stack 2 is used for dequeueing.

A queue can be implemented with two stacks, where one stack is used to deal with the enqueue operation and the other stack is used to deal with the dequeue operation. The visualization for the explanation right after this is shown in Figure 1. As an example, if we wanted to enqueue elements $1, 2, 3, ..., n$ in the named order, these elements would be pushed to Stack 1. If we wanted to dequeue elements, then we would first copy all the elements from Stack 1 to 2 in the reverse order–elements are copied in the order of $n, ..., 3, 2, 1$ since last added is first deleted in stacks. If we wanted to enqueue after dequeueing, Stack 1 would need to be updated with Stack 2, so all the elements from Stack 2 needs to be copied over to Stack 1 in the reverse order.

Before getting into the run-time analysis of this implementation, the Bankers Method/Accounting Method was used for the amortized analysis. When using the Bankers Method, the goal is for the least costly operations to pay more and for the more costly operations to pay less.

The two major operations in this problem are enqueueing and dequeueing. For dequeueing to be possible, there has to be elements added to the stack through enqueueing; therefore, I analyzed enqueueing to have a run-time of $O(a)$, with a being the number of times the element is moved between the two stacks. With enqueueing having a run-time of $O(a)$, I analyzed dequeueing to have a run-time of $O(0)$. Over n operations, the total run-time would then be of $O(an)$.

$$TotalRuntime = n * (O(a) + O(0)) \tag{1}$$

$$n * (O(a) + O(0)) = O(an) \tag{2}$$

$O(an)$ is simplified into $O(n)$. The total run-time is then divided by the total number of operations to find the amortized run-time.

$$\frac{O(n)}{n} = O(1)\, amortized \tag{3}$$

Problem 2

Given that n is the number of operations taken on the deque:

$$n \geq number\ of\ push\_back\ operation \geq number\ of\ pop\_back\ operation \tag{4}$$

$$n \geq number\ of\ push\_front\ operation \geq number\ of\ pop\_front\ operation \tag{5}$$

The four operations: push_back, push_front, pop_back, and pop_front that are operated on the deque has the run-time of $O(1)$. However, when we consider moving the elements from one stack to another, the run-time for transferring the elements would be $O(\frac{a}{2})$, with a being the number of times elements are moved from one stack to another and with $\frac{1}{2}$ being multiplied since half of the elements from a stack are moved each time.

$$O(1) * O(a) * O(\frac{1}{2}) = O(\frac{a}{2}) \tag{6}$$

With n operations,

$$O(\frac{a}{2}) * O(n) = O(\frac{an}{2}) \tag{7}$$

which $O(\frac{an}{2})$ can be simplified into $O(n)$. The total run-time is then divided by the total number of operations to find the amortized run-time.

$$\frac{O(n)}{n} = O(1)\ amortized \tag{8}$$

Problem 3

The two data structures used in this problem are: a queue and a deque. The queue is used to perform the operations enqueue and dequeue; the deque is used to perform the operation find_min. The potential method was used for the amortized analysis for this problem.

$$Cost = actual\ cost + data\ structure\ after\ k^{th}\ operation - data\ structure\ before\ k^{th}\ operation \tag{9}$$

or

$$Cost = actual\ cost + \Delta\ in\ elements\ in\ the\ data\ structure \tag{10}$$

Find_min

$$[5, 2, 1, 0, 2, 7, 8, 9, 1]$$

To better explain the algorithm for finding the minimum value of the queue, a queue of values is given above. Whenever a new element is enqueued to the queue, the element is also enqueued to the deque as well. However, once the deque starts to have more than one element it starts keeping track of what the minimum value is. To have the minimum value as the first element in the deque, values prior to the minimum value and are greater than the minimum value are pop_fronted from the deque. Therefore values being pop_fronted are to get rid of elements when a new minimum is detected. So the deque for keeping track of the minimum value for the queue above can reach the following status below:

$$[0, 2, 7, 8, 9, 1]$$

When values greater than the current minimum value are added to the deque, they are kept since they are potential candidates for the next minimum value in case the current minimum value (which is 0) gets dequeued from the queue. If a new minimum is detected afterwards (1 in this case) then elements are deleted from the deque in the leftward way so that 1 can be the next minimum if 0 is dequeued from the queue.

$$[0, 1]$$

Correctness

Since the deque keeps record of the minimum values in order, this method of finding the minimum of the queue has to be correct. From the left side of the queue, we always have the smallest value by deleting the ones that are greater than it. In case the minimum value is dequeued from the original queue, the minimum value can also be dequeued from the deque and the next minimum value is already at the next index, by deleting elements in the rightward direction as explained above.

Queue

The amortized analysis using potential method on enqueueing on queues is given in equation (20) to (22).The total cost of enqueueing itself only takes 1, there would be $n+1$ elements in the queue after performing the enqueue, and there would be n elements in the queue before performing the enqueue. The difference in the number of elements would be therefore 1.

$$Enqueue = 1 + (n+1) - n \tag{11}$$

$$Enqueue = 1 + 1 \tag{12}$$

$$Enqueue = 2 \rightarrow O(1)\ amortized \tag{13}$$

The amortized analysis using the potential method on dequeueing on queues is given in equation (14) to (16). The total cost of enqueueing itself also only takes 1, there would be $n-1$ elements in the queue after performing the enqueue, and there would be n elements in the queue before performing the enqueue. The difference in the number of elements would be therefore -1.

$$Dequeue = 1 + (n-1) - n \tag{14}$$

$$Dequeue = 1 + (-1) \tag{15}$$

$$Dequeue = 0 \rightarrow O(1)\ amortized \tag{16}$$

Deque

The amortized analysis on pop_front *and* $push\_front$ is similar to the amortized analysis done on queueing on queues.

$$pop\_front,\ push\_front = 1 + (n+1) - n \tag{17}$$

$$Enqueue = 1 + 1 \tag{18}$$

$$Enqueue = 2 \rightarrow O(1)\ amortized \tag{19}$$

The amortized analysis on pop_back *and* $push\_back$ is similar to the amortized analysis done on dequeueing on queues.

$$pop\_back,\ push\_back = 1 + (n-1) - n \tag{20}$$

$$Enqueue = 1 + (-1) \tag{21}$$

$$Enqueue = 0 \rightarrow O(1)\ amortized \tag{22}$$

The run-time analysis for $find\_min$ is simply $O(1)$ as we only need to take the first value of the deque.