① Exercise 17-3-3 on pg 462.

Consider an ordinary binary min-heap data structure with n elements supporting the instructions INSERT & EXTRACT-MIN in $O(\lg n)$ worst-case time. Give a potential funct^n ϕ such that the amortized cost of INSERT is $O(\lg n)$ & the amortized cost of EXTRACT-MIN is $O(1)$, & show that it works.

Solution:

* Let D_i be the heap after the ith operation, let D_i consist of n_i elements. Also, let k be a constant such that each INSERT or EXTRACT-MIN operation takes at most $k \ln n$ time, where $n = \max(n_{i-1}, n_i)$.

* Define

$$\phi(D_i) = \begin{cases} 0 & \text{if } n_i = 0, \\ k n_i \ln n_i & \text{if } n_i > 0. \end{cases}$$

This function exhibits the characteristics we like in a potential function: if we start with an empty heap, then $\phi(D_0) = 0$, & we always maintain that $\phi(D_i) \geq 0$.

* Before proving that we achieve the desired amortized times, we show that if $n \geq 2$, then $n \ln \frac{n}{n-1} \leq 2$.

We have, $n \ln \frac{n}{n-1} = n \ln \left(1 + \frac{1}{n-1}\right)$

$$= \ln \left(1 + \frac{1}{n-1}\right)^n$$

$$\leq \ln \left(e^{\frac{1}{n-1}}\right)^n \quad (\text{since } 1 + x \leq e^x \text{ for all real } x)$$

$$= \ln e^{\frac{n}{n-1}}$$

$$= \frac{n}{n-1}$$

$$\leq 2.$$

assuming that $n \geq 2$.

* If the ith operation is an INSERT, then $n_i = n_{i-1} + 1$
If the ith operation inserts into an empty heap, then $n_i = 1$, $n_{i-1} = 0$, & the amortized cost is.

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &\le k \ln 1 + k \cdot 1 \ln 1 - 0 \\ &= 0.\end{aligned}$$

* If the ith operation inserts into a nonempty heap, then $n_i = n_{i-1} + 1$, & the amortized cost is

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &\le k \ln n_i + k n_i \ln n_i - k n_{i-1} \ln n_{i-1} \\ &= k \ln n_i + k n_i \ln n_i - k(n_i - 1) \ln (n_i - 1) \\ &= k \ln n_i + k n_i \ln n_i - k n_i \ln (n_i - 1) + k \ln (n_i - 1) \\ &< 2k \ln n_i + k n_i \ln \frac{n_i}{n_i - 1} \\ &\le 2k \ln n_i + k n_i \cdot \frac{1}{n_i - 1} \cdot 2k \\ &= O(\lg n_i).\end{aligned}$$

* If the ith operation is an EXTRACT-MIN, then $n_i = n_{i-1} - 1$.
If the ith operation extracts the one & only heap item, then $n_i = 0$, $n_{i-1} = 1$, & the amortized cost is

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &\le k \ln 1 + 0 - k \cdot 1 \ln 1 \\ &= 0.\end{aligned}$$

* If the ith operation extracts from a heap with more than 1 item, then $n_i = n_{i-1} - 1$ & $n_{i-1} \ge 2$, & the amortized cost is

$$\begin{aligned}\hat{c}_i &= c_i + \phi(D_i) - \phi(D_{i-1}) \\ &\le k \ln n_{i-1} + k n_i \ln n_i - k n_{i-1} \ln n_{i-1}\end{aligned}$$

$$\hat{c}_i = k \ln n_{i-1} + k(n_{i-1} - 1) \ln(n_{i-1} - 1) - k n_{i-1} \ln n_{i-1}$$

$$= k \ln n_{i-1} + k n_{i-1} \ln(n_{i-1} - 1) - k \ln(n_{i-1} - 1) - k n_{i-1} \ln n_{i-1}$$

$$= k \ln \frac{n_{i-1}}{n_{i-1} - 1} + k n_{i-1} \ln \frac{n_{i-1} - 1}{n_{i-1}}$$

$$< k \cdot \ln \frac{n_{i-1}}{n_{i-1} - 1} + k n_{i-1} \ln 1$$

$$= k \ln \frac{n_{i-1}}{n_{i-1} - 1}$$

$$\leq k \ln 2 \quad (\text{since } n_{i-1} \geq 2)$$

$$= O(1).$$

* A slightly different potential function - which may be easier to work with - is as follows. For each node x in the heap, let $d_i(x)$ be the depth of x in D_i.

Define $\phi(D_i) = \sum_{x \in D_i} k(d_i(x) + 1)$

$$= k\left(n_i + \sum_{x \in D_i} d_i(x)\right)$$ where k is defined as before.

* Initially, the heap has no items, which means that the sum is over an empty set, & so $\phi(D_0) = 0$. We always have $\phi(D_i) \geq 0$

* Observe that after an INSERT, the sum changes only by an amount equal to the depth of the new last node of the heap, which is $\lfloor \lg n_i \rfloor$. Thus, the change in potential due to an INSERT is $k(1 + \lfloor \lg n_i \rfloor)$, & so the amortized cost is

$$O(\lg n_i) + O(\lg n_i) = O(\lg n_i) = O(\lg n)$$

* After an EXTRACT-MIN, the sum changes by the negative of the depth of the old last node in the heap, & so the potential decreases by $k(1 + \lfloor \lg n_{i-1} \rfloor)$. The amortized cost is at most

$$k \lg n_{i-1} - k(1 + \lfloor \lg n_{i-1} \rfloor) = O(1).$$

(2) Exercise 17.4-3 on pg 471

Suppose that instead of contracting a table by ~~defacst~~ 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

$$\phi(T) = |2 \cdot T.num - T.size|,$$

show that the amortized cost of a TABLE-DELETE that uses this strategy is bounded above by a constant.

Solution:

* Suppose that ith operation is TABLE_DELETE, consider the value of load factor α:

α_i = (no of entries in table after iteration i)/(size of table after iteration i)

$= num_i / size_i$

case 1: If $\alpha_{i-1} = 1/2$, $\alpha_i < 1/2$

$$\hat{c}_i = c_i + \phi_i - \phi_{i-1}$$
$$= 1 + (size_i - 2num_i) - (2num_{i-1} - size_{i-1})$$
$$= 1 + (size_{i-1} - 2(num_{i-1} - 1)) - (2num_{i-1} - size_{i-1})$$
$$= 3 + 2size_{i-1} - 4num_{i-1}$$
$$= 3 + 2size_{i-1} - 4\alpha_{i-1} size_{i-1}$$
$$\leq -1 + 2size_{i-1} - 4/2\, size_{i-1}$$
$$= 3$$

case 2: If $1/3 \leq \alpha_{i-1} < 1/2$, $\alpha_i \leq 1/2$, i-th operation would not cause a contraction

$$\hat{c}_i = c_i + \phi_i - \phi_{i-1}$$
$$= 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1})$$
$$= 1 + (size_{i-1} - 2(num_{i-1} - 1)) - (size_{i-1} - 2num_{i-1})$$
$$= 3$$

Case 3: If $\alpha_{i-1} = 1/3$, $\alpha_i \leq 1/2$ thus i-th operation would cause a contraction.

$$\hat{c}_i = c_i + \phi_i - \phi_{i-1}$$

$$= num_i + 1 + (size_i - 2num_i) - (size_{i-1} - 2num_{i-1})$$

$$= num_{i-1} - 1 + 1 + (2/3\, size_{i-1} - 2(num_{i-1} - 1)) - (size_{i-1} - 2num_{i-1})$$

$$= num_{i-1} - 1/3\, size_{i-1} + 2$$

$$= 2.$$

Thus the amortized cost of TABLE-DELETE is bounded by 3.

(3) A deque is like a queue. but one can insert & delete at either end- ie, one can insert/delete at both the head & the tail. The object of this problem is to implement a deque using 3 stacks, called Head, Tail & Temp, in such a way that all insert/delete operations take amortized O(1) time.

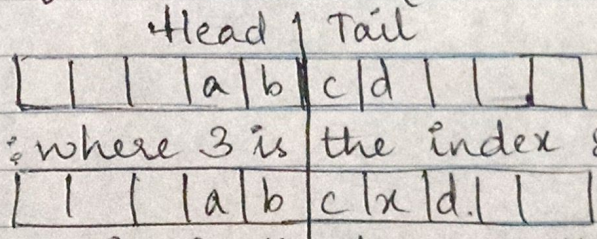
(a) The 2 stacks Head & Tail contain, respectively, the front & rear ele of deque. The tops of the stacks are the ends of the deque so they are accesible, while the bottom stack elements are the innermost of the deque. The 4 deque operatns are insert/delete from the front ~~end~~ & insert/delete from the rear. Describe the insert operations.

Solution:

* - The deque has 2 stacks Head & Tail. It places these stacks back-to-back, so that operations are fast at either end.
- The total number of elements, n = Head.size() + Tail.size()

∴ int size() {
    return Head.size() + Tail.size();
}

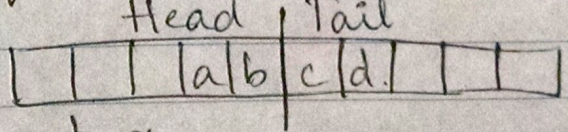
÷ Insertion of an element. (Insert rear)



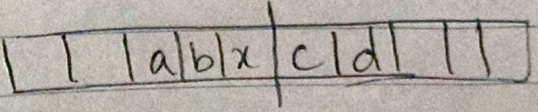
Add (3, x); where 3 is the index & 'x' is the element to be inserted

this is because if i < Head.size(), then it corresponds to element of Head at position Head.size() - i - 1.

- Insertion of an element (Insert front).



Add element x:



(3)(b) The delete operations are simple if the corresponding stack is not empty - just pop the top element off the appropriate stack. But if the stack you need to pop from is empty, you need to get to the bottom element on the other stack. Explain how to do this using the Temp stack to split the contents of the non-empty stack into 2 halves so that Head & Tail contain, respectively, the front & rear elements of the deque.

Solution:

* If both stacks, Head & Tail are not empty, we can extract the topmost element (pop() operation) from the Head stack or tail stack. This will be used as Front Delete & Rear delete

* If either of the 2 stacks, Head & tail is empty, we need to split the non-empty stack using the Temp stack & later push the elements onto the queue.

(3)(c) What is the worst-case of each of the 4 operations?

- Insert Front : Need to push element in Head stact.

$$T(\text{Insert Front}) = O(1)$$

- Insert Rear : Need to push element in Tail stack

$$T(\text{Insert Rear}) = O(1).$$

- Delete Front : Need to pop element out from Head stack

$$T(\text{Delete Front}) = O(1).$$

- Delete Rear : Need to pop element out from Tail stack

$$T(\text{Delete Rear}) = O(1).$$