EECS 336

Litong Wang

Homework 6

May 16, 2016

Problem 1

Solution:

Assume D_i is the heap after the ith operation, and consists of n_i elements. k is a constant such that $c_i \leq k \ln n$. sWe can define the potential function $\Phi(D_i)$ as follow:

$$\Phi D_i = \begin{cases} 0 & \text{if } i = 0 \\ k n_i \ln n_i & \text{if } i > 0 \end{cases} \tag{1}$$

If we start with an empty heap, then we have $\Phi(D_0) = 0$, and we always maintain that $\Phi(D_i) \geq 0$.

If the ith operation is Insert, then $n_i = n_{i-1} + 1$. If we start with an empty heap, $n_{i-1} = 0$ and $n_i = 1$. And the amortized cost is:

$$\begin{aligned} \widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k \ln 1 + k \cdot 1 \ln 1 - 0 \\ &= 0 \end{aligned}$$

If we insert the element into an nonempty, then $n_i = n_{i-1} + 1$, then the amortized cost is:

$$\begin{aligned} \widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k \ln n_i + k \cdot n_i \ln n_i - k \cdot n_{i-1} \ln n_{i-1} \\ &= k \ln n_i + k \cdot n_i \ln n_i - k \cdot (n_i - 1) \ln(n_i - 1) \\ &= k \ln n_i + k \cdot n_i \ln n_i - k \cdot n_i \ln(n_i - 1) + k \cdot \ln(n_i - 1) \\ &\leq 2k \ln n_i + k \cdot n_i \ln \frac{n_i}{n_i - 1} \\ &\leq 2k \ln n_i + 2k \\ &= \mathcal{O}(\lg n_i) \end{aligned}$$

If the ith operation is Extract-Min, then $n_i = n_{i-1} - 1$. If the ith operation extracts from a heap with only one element, then $n_i = 0$ and $n_{i-1} = 1$. And the amortized cost is:

$$\begin{aligned} \widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k \ln 1 + 0 - k \cdot 1 \ln 1 \\ &= 0 \end{aligned}$$

If the ith operation is Extract-Min and extracts from a heap with more than one element, then $n_i = n_{i-1} - 1$ and $n_{i-1} \geq 2$, and the amortized cost is:

$$\begin{aligned}
\widehat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
&\leq k \ln n_{i-1} + k n_i \ln n_i - k n_{i-1} \ln n_{i-1} \\
&= k \ln n_{i-1} + k(n_{i-1} - 1) \ln(n_{i-1} - 1) - k n_{i-1} \ln n_{i-1} \\
&= k \ln n_{i-1} + k n_{i-1} \ln(n_{i-1} - 1) - k \ln(n_{i-1} - 1) - k n_{i-1} \ln n_{i-1} \\
&= k \ln \frac{n_{i-1}}{n_{i-1} - 1} + k n_{i-1} \ln \frac{n_{i-1} - 1}{n_{i-1}} \\
&< k \ln \frac{n_{i-1}}{n_{i-1} - 1} \\
&\leq k ln 2 \\
&= \mathcal{O}(1)
\end{aligned}$$

Problem 2

Solution:

a) Let's say we have two stacks A and B. We use stack A to do enqueue operation and use stack B to do dequeue operation. In case of a dequeue operation, if stack B is empty, we need to pop all the elements from stack A and push it into stack B, then we can do the dequeue operation.

The actual costs of operations were:

$$\begin{aligned}
Enqueue &\quad 1, \\
Dequeue &\quad s, \\
Multi - Dequeue &\quad 2\min(k, s),
\end{aligned}$$

where k is the arguement supplied by Multi-Dequeue and s is the size of stack A. We can assgin the the following amortized costs:

$$\begin{aligned}
Enqueue &\quad 4, \\
Dequeue &\quad 0, \\
Multi - Dequeue &\quad 0,
\end{aligned}$$

We can pay for any sequence of queue operations by charging the amortized costs. Suppose we use a dollar bill to represent each unit of cost. We start with two empty stacks A and B. In case of Enqueue operation, we push one element into stack A, and we use 1 dollar to

pay the actual cost of the push and are left with a credit of 2 dollar. At any point in time, every element on the stack has 2 dollar of credit on it.
The dollar stored as credits serves as prepayment for the future cost. One dollar of popping it from the stack A, one dollar of pushing it into stack B and one dollar of popping it from stack B. When we execute a Dequeue operation, we charge the operation nothing and pay its actual cost using the credit stored in the stack. To dequeue an element from our queue, if stack B is not empty, we take one dollar of credit from the bank and use it to pay the actual cost of the operation. If stack B is empty, we need to pop all elements from stack A, we take one dollar of credit from the bank at each poping. And we push all these elements into stack B, we take one dollar of credit from the bank to pay for the actual cost. And we use one dollar of credit from the bank to pay for the actual cost of Dequeue which actually is poping from stack B. Thus, by charging the Enqueue operation a little bit more, we can charge the Dequeue operation nothing.
Moreover, we can also charge Multi-Dequeue operations nothing. If stack B is not empty, to Dequeue the first element, we take the dollar of credit from the bank and use it to pay the actual cost of a Pop operation. To Dequeue a second element, we again have a dollar of credit from the bank to pay for the Pop operation, and so on. If stack B is empty, we Pop all elements from stack A and Push it into stack B, we can use credits to pay for the actual costs. And then we Pop elements from stack B to finish the Multi-Dequeue operation. Thus, we have always charged enough up front to pay for Multi-Dequeue operations. In other words, since each element on the stack A has 3 dollar of credit on it, and the stack always has a nonnegative number of elements, we have ensured that the amount of credit is always nonnegative. Thus, for any sequence of n Enqueue, Dequeue, and Multi-Dequeue operations, the total amortized cost is an upper bound on the total actual cost. Since the total amortized cost is $\mathcal{O}(n)$, so is the total actual cost. And the amortized cost of each Enqueue, Dequeue and Multi-Dequeue is $\mathcal{O}(1)$.

Problem 3

Solution:

Problem 4

Solution:

Problem 5

Solution: