Amortized Analysis and Binomial Heaps

Amortized analysis

Suppose we have a data structure which starts off in some initial state, S_0. It is then subjected to a series of n operations, resulting in the data structure going through states $S_1, S_2, \ldots, S_n$. We're trying to analyze the amount of time it takes to perform all n operations.

A potential function, denoted Φ, is a map from the state of the data structure, S_i, to a nonnegative number, $\Phi(S_i)$. Let's write Φ_i for $\Phi(S_i)$. We assume $\Phi_0 = 0$.

The potential function is used only to *analyze* our data structure's performance over the sequence of n operations. It's not implemented in code or any such thing. It's simply an accounting trick. We've used a similar accounting trick before (in discussing dynamic rehashing).

The change in potential on performing the i'th operation, for $i \in [1..n]$, is defined as

$$\Delta\Phi_i = \Phi_i - \Phi_{i-1}.$$

The actual cost of performing the operation i will be denoted cost_i. The actual cost for performing the sequence of n operations operations will be denoted cost.

The amortized cost of the i-th operation, amcost_i is defined as

$$\mathsf{amcost}_i = \mathsf{cost}_i + \Delta\Phi_i.$$

Proposition 1. *Over the sequence of n operations, the (actual cost) is at most n times the maximal amortized cost for each operation.*

The proof is simple:

$$\begin{aligned}
\mathsf{cost} &= \mathsf{cost}_1 + \mathsf{cost}_2 + \cdots \mathsf{cost}_n \\
&= (\mathsf{cost}_1 + \Delta\Phi_1) + (\mathsf{cost}_2 + \Delta\Phi_2) + \ldots + (\mathsf{cost}_1 + \Delta\Phi_n) - (\Phi_1 + \Phi_2 + \cdots + \Phi_n) \\
&= \mathsf{amcost}_1 + \mathsf{amcost}_2 + \cdots + \mathsf{amcost}_n - (\Phi_1 - \Phi_0 \ + \ \Phi_2 - \Phi_1 \ + \ \cdots \ + \ \Phi_n - \Phi_{n-1}) \\
&= \mathsf{amcost}_1 + \mathsf{amcost}_2 + \cdots + \mathsf{amcost}_n - (\Phi_n - \Phi_0) \\
&\leq \mathsf{amcost}_1 + \mathsf{amcost}_2 + \cdots + \mathsf{amcost}_n
\end{aligned}$$

In other words, the total cost is bounded by the sum of amortized costs, so the average cost per operation is bounded by the amortized cost of that operation. $\diamondsuit$

Binomial Heaps

In class we used amortized analysis to show that binomial heaps support `insert` and `combine` in $O(1)$ amortized time, and `delete-min` in $O(\lg n)$ amortized time. In the 10:00 section I stumbled in the proof of the last fact, and so let me repeat it now.

Recall that we were using a potential function of

$$\Phi(S) = \text{The number of trees in binomial heap } S$$

When we do a `delete-min` operation we start off by deleting the element pointed to by the `min` pointer, linking its children into the root list. This has actual cost $\leq \lg n$ (in fact, the cost is 1 just by circularly linking the children of each node together, so that we can splice the children of the node into the root list by a constant number of pointer adjustments). After that, we consolidate the trees and identify the new minimum in the process. To see how long this takes, suppose there were t_0 trees before we did the consolidation and t_f trees after. The number of times that two trees got combined must have been $t_0 - t_f$. If we charge $1/2$ to inspect a tree and update `min`, and we charge another $1/2$ to combine two different trees, then the total cost for doing the consolidation of trees will be $t_0/2 + (t_0 - t_f)/2 \leq t_0$. Adding this to the earlier cost of $\leq \lg n$, we conclude that the actual cost of `delete-min` is $\leq t_0 + \lg n$.

Now what is $\Delta\Phi$ for doing the `delete-min`? The potential Φ got increased by $\leq \lg n$ because linking the deleted node's children into the root list increased the number of trees by at most that many. But then Φ got decreased by the $t_0 - t_f$ tree consolidations. So the change in potential is $\leq \lg n - (t_0 - t_f) \leq 2\lg n - t_0$.

The amortized cost is the actual cost plus the change in potential, so the amortized cost is $\leq (t_0 + \lg n) + (2\lg n - t_0) \leq 3\lg n$. Thus the amortized cost is $O(\lg n)$, as claimed.