COMPSCI 330: Design and Analysis of Algorithms 10/21/2014 and 10/23/2014

Lecture #15 and #16

Lecturer: Debmalya Panigrahi *Scribe: Nat Kell and Ang Li*

1 Introduction

In this lecture, we will cover *amortized analysis*, which is a method where we consider the average work done over an entire sequence of operations instead of just considering the worst-case cost/time for a single operation; by doing such an analysis, we can obtain tighter bounds in scenarios when expensive operations occur rarely enough that we can charge their cost to more frequent inexpensive operations. We cover such charging arguments as well as the *potential method*, which is another powerful tool for doing amortized analysis.

2 Binary Counting

2.1 Amortized Analysis

Suppose we wanted to increment a variable x starting at 0 until we reach some value n, where we represent x as a binary number with $\lfloor \log n \rfloor + 1$ bits (we'll denote this binary representation as $\textsc{bin}(x)$). Using the basic carry addition algorithm (that we all know so well from primary school), we implement each x++ as follows. Starting at the least significant bit (LSB), if we observe a 1 we flip it to 0 and then move to the next bit to the left. Otherwise if we observe a 0, we flip it to a 1 and stop (for instance, if we add 1 to 101001101*0111111*, we obtain 101001101*1000000*, where the digits in italics are the bits that get flipped).

Over all n increments, we can ask the question: *how many bit flips do we do in total?* Clearly, the number of flips for a single increment is $O(\log n)$ since there are only $\lfloor \log n \rfloor + 1$ bits, and since we do n increments in total, a rough upper bound on the total flips is $O(n \log n)$. However, it seems like we should be able to tighten this aggregate bound since on many increments we are only flipping a few bits; ideally, we would like to show a bound of $O(n)$, which would establish that, even if the worst-case bound of a single increment is $\Theta(\log n)$, on average we are only doing a constant amount of work for each operation.

We can indeed obtain an $O(n)$ bound with the following "direct analysis." Observe that the kth LSB (i.e., 0th least significant bit is the right most bit) can be flipped from 1 to 0 and back to 1 at most $n/2^k$ times. When this event happen, the counter has increased by 2^k, and since our final total is n, it follows that these two flips can occur at most $n/2^k$ times. Summing over all bit positions k and using the closed form for an infinity geometric series we obtain:

$$\text{Total bit flips} = \sum_{k=0}^{\lfloor \log n \rfloor + 1} \frac{n}{2^k} < n \cdot \sum_{k=0}^{\infty} \frac{1}{2^k} = 2n = O(n).$$

2.2 Charging Arguments

This analysis is similar to arguments we have seen previously this semester: For each possible index for a bit flip, we argued an upper bound on the number of times this type of flip/event can happen and summed these upper bounds over all possible positions to obtain the desired upper bound for the total number of flips. Now, we are going to perturb this analysis somewhat and instead frame it as a *charging argument*. The idea is that instead of analyzing each operation directly, we distribute or *charge* the work done on expensive operations onto inexpensive operation and then sum over all the work done over all operations with this new work distribution.

To make this more concrete, consider the following charging scheme for binary counting. First observe that:

1. On a given increment, there is exactly one 0-1 flip (the last flip we perform); therefore, there are exactly n 0-1 flips in total.

2. Each 1-0 is proceeded by a unique 0-1 flip.

Therefore, we *charge every 1-0 flip the preceding 0-1 flip*. By Observation 2, a given 0-1 flip is charged exactly once; thus now, we have distributed the work so that 1-0 flips cost nothing and 0-1 flips cost 2—one unit of work for doing the actual flip and then another unit of work from the charge the proceeding 1-0 flip is now placing on this operation. Even though we have made 0-1 flips more expensive, we have only done so by a constant amount. By Observation 1, there are only n 0-1 flips in total, and therefore using this new work distribution we conclude that the number of flips is at most $2n = O(n)$.

2.3 Potential Functions

Another method for doing amortized analysis is using what are called *potential functions*. Broadly defined, a potential function $\Phi(i)$ maps the state of an algorithm or data structure after the ith operation to a carefully defined non-negative value. The naming convention of calling $\Phi(i)$ a "potential" comes from the fact that the change in $\Phi(i)$ is somewhat analogous to the state system accumulating potential energy and then releasing it later as kinetic energy. On an inexpensive operation i, $\Delta\Phi = \Phi(i) - \Phi(i-1)$ will likely be positive and thus we build potential energy in the system. Then on expensive operations, we release potential energy as kinetic energy by making $\Delta\Phi$ negative, which will counterbalance the higher cost. Another analogy that is often used is that the potential function is a bank account where we save money on inexpensive operations and spend the money we have saved on expensive operations.

More specifically, let c_i be the cost of operation i (so in the case of the binary counter, this is the number of bits flipped on the ith increment). Given a potential function Φ, we define the *amortized cost* $\hat{c}_i$ of operation i to be $c_i + \Phi(i) - \Phi(i-1)$. Observe that we have the following bound if we perform n operations:

$$\text{Total amortized cost} = \sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(i) - \Phi(i-1)) = \sum_{i=1}^{n} (c_i) + \Phi(n) - \Phi(0). \tag{1}$$

The last equality follows since all the $\Phi(i)$ terms in the summation *telescope* except for $\Phi(0)$ and $\Phi(n)$ (i.e., for each $j = 1, \ldots, n-1$, the positive term $\Phi(i)$ when $i = j$ will cancel with the negative term $-\Phi(i-1)$ when $i = j+1$). Therefore, as long as $\Phi(0)$ and $\Phi(n)$ are non-negative, then amortized cost is an upper bound on actual cost (since actual cost is just $\sum_{i=1}^{n} c_i$).

As we mentioned for our binary counter example, c_i is just the number of bit flips on the ith increment. To define our amortized cost, we will use the following potential function:

$$\Phi(i) = \text{The number of 1s in } \textsc{bin}(x) \text{ after } i \text{ increments.} \tag{2}$$

To do our analysis, consider the difference between the number of 1s in x before and after the ith increment; namely, we will define $\Phi(i)$ in terms of $\Phi(i-1)$. Observe that based on the algorithm, there are $c_i - 1$ 1s that become 0s on the ith increment: the first $c_i - 1$ are 1s being flipped to 0, and then the last unit of cost is added from flipping the last 0 to a 1. Since this last flip also adds an extra 1 to x, the total number of 1s after the ith increment is $\Phi(i) = \Phi(i-1) - (c_i - 1) + 1$. Therefore, the total amortized cost is as follows:

$$\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} (c_i + \Phi(i) - \Phi(i-1)) \\
&\leq \sum_{i=1}^{n} (c_i + (\Phi(i-1) - c_i - 1) + 1 - \Phi(i-1)) \qquad (\text{since } \Phi(i) = \Phi(i-1) - (c_i - 1) + 1) \\
&= \sum_{i=1}^{n} 2 = 2n.
\end{aligned}$$

Clearly, both $\Phi(0)$ and $\Phi(n)$ are non-negative (we cannot have a negative number 1s in $\textsc{bin}(x)$); therefore by (1), $2n = O(n)$ is an upper bound on the total actual cost, as desired.

To get some intuition as to how the potential function is working, observe that whenever we only flip the first bit (i.e., we are incrementing an even number to an odd number), the amortized cost is 2 instead of just 1—these are the operations where we save potential. For any increment where we flip three or more bits, there is a drop in potential where we spend the money we saved from each of the odd increments. Note that such operations can charge the potential much more than just a cost of 1 (it can be as high as $\log(n)$); however, since we are saving potential on half of the increments, we always have enough money in the potential's bank account to lower the cost of every operation down to just 2.