Math 176 - Fall 2000 - Advanced Data Structures

Amortized Costs Analyses

S. Buss

This document discusses a simple use of amortized costs analysis for dynamically resized arrays. The textbook, in Chapter 11, has several proofs of amortized cost bounds. The example and analysis we give below is a very simple case of amortized cost analysis that helps to introduce the ideas, the terminology and the notation for amortized cost analysis.

In a business/financial setting, the phrase "amortized" costs refers to the practice of spreading upfront expenses, say for a piece of equipment, over the expected period of time during which the equipment will be in use. For example, if a grocery store purchases a refrigerator for $10,000, it does not try to recoup its investment in the very day it is used. Instead, it spreads the costs over the period of years in which the refrigerator will be used, and raises prices so as to recover the costs gradually.

The previous paragraph, the costs were incurred "upfront" and then recovered over a period of time. For amortized costs bounds in algorithms, a more stringent bound is generally used. Namely, the costs cannot be incurred upfront: instead, one gradually builds up a "savings account" of time, which can be drawn against as needed. (In other words, we do not allow algorithms to go into debt, time-wise.) This is more like the now old-fashioned idea of a "Christmas savings club", where one saves money throughout the year to be used for Christmas purchases at the end of the year.

Mathematically, we assume that there a function $T(i)$ which tells the time spent by operation i. We let $A(i)$ be the amortized time spent by operation i. (Actually, $A(i)$ does not depend on the time, but on the operation performed in time step i). Then we say that the operations have amortized time given by $O(A(i))$, provided we have

$$\text{Total time for } m \text{ operations } \leq \sum_{i=1}^{m} A(i)$$

holding for all sequences of m operations. Rephrasing this slightly, we need to have, for all m:

$$\sum_{i=1}^{m} T(i) \leq \sum_{i=1}^{m} A(i). \tag{1}$$

What will usually happen is that $A(i)$ will be slightly larger than $T(i)$ for most values of i. But occasionally, $T(i)$ will be large (for an occasional operation that has slow run time), and on these occasions, it is possible for $A(i)$ to be less than $T(i)$.

It will helpful to work with notions of "cumulative time" too. For this, we define,

$$CT(m) \;=\; \sum_{i=1}^{m} T(i) \qquad \text{and} \qquad CA(m) \;=\; \sum_{i=1}^{m} A(i).$$

Equation (1) is then equivalent to

$$CT(m) \;\leq\; CA(m) \tag{2}$$

In order to prove these inequalities, we shall introduce a potential function $Potential(i)$. The "potential" will be a measure of how much time has been saved or "banked" and is available for time overruns in the future. We shall require that the potential is zero at time $i = 0$ and that

$$Potential(i) \;\geq\; 0$$

for all i. We write $\Delta Potential$ for the change in potential; i.e.,

$$\Delta Potential \;=\; Potential(i) - Potential(i-1).$$

The general strategy for amortized costs proofs will be as follows: we first get explicit run times for the various operations. These give the values of $T(i)$. We then choose functions $A()$ for the amortized costs of the various operations, and choose a potential function $Potential$. We then prove that

$$T(i) \;+\; \Delta Potential \;\leq\; A(i)$$

always holds. By summing over $i = 1, 2, 3, \ldots, m$, we get that

$$CT(i) \;+\; \Delta Potential \;\leq\; CA(i),$$

and from the non-negativity of the potential, this implies that equation 2 holds. In this case, we say the functions $A()$ are amortized run time bounds for the various operations.

Note that we have a good deal of freedom on how to choose $A()$ and the potential. One wants to make $A()$ take on small values in order to prove better amortized bounds. Frequently the hard part of an amoritized run time bound proof is figuring out what to use for the potential in order to make the proof work.

0.1 Dynamically resized arrays

We consider the data structure of dynamically resized arrays. For simplicity, we let this abstract data type support the operations `get( int index )`, and `append ( Object o )`, and `int size()`. Obviously, `get` and `size` are constant

time operations, taking $O(1)$ time. The `append` operation will usually take time $O(1)$, but occasionally it has to resize the array. Resizing the array involves allocating space for an array of double the size of the current array, copying the elements from the old array to the new, larger array, and then deallocating the old array.

We define the dynamically resizable arrays to have the following data members:

```
int sizeUsed;          // Number of elements stored
int sizeAllocated;     // Number of entries allocated
Object A[];            // Array of objects
```

Theorem 1 *The amortized time for an* `append` *operation is* $O(1)$.

The actual time for an `append` operation is

- If the array does not need to be resized, we take the run time of `append` to be 1 unit. In this case, $T(i) = 1$.
- If the array has n elements and needs to be resized, we take the run time for `append` to be $1 + n$ units; i.e., $T(i) = 1 + n$ in this case.

For the potential function, we use the following function:

$$\mathit{Potential}(i) \;=\; 2 \cdot \max\{0, \texttt{sizeUsed} - (\texttt{sizeAllocated}/2)\}$$

For the amortized run time of `append`, we use $A(i) = 3$.

In order to prove that the amortized run time of `append` is really $O(1)$, it is enough to prove that we always have:

$$T(i) \;+\; \Delta \mathit{Potential} \;\leq\; 3.$$

Proof. There are two cases to consider:

Case 1: The array does not need to be resized. In this case, $T(i) = 1$ and $\Delta \mathit{Potential} \leq 2$. And since $1 + 2 \leq 3$, we are done.

Case 2: The array does need to be resized. In this case, we have $n =$ `sizedUsed` $=$ `sizeAllocated`. The potential function value then changes from n to 2. So the change in potential is

$$\Delta \mathit{Potential} \;=\; 2 - n.$$

The actual runtime $T(i)$ is equal to $1 + n$. We immediately get that

$$T(i) + \Delta \mathit{Potential} \;=\; 3$$

and the proof is completed. □