# Amortized complexity and the use of potential functions

October 9, 2003

Amortized complexity is a measure which captures the "average" complexity of different kinds of operations while performing a sequence of operations on data structure. When applied to a sequence of operations, it sometimes gives tighter bounds than the bounds derived by the "worst case" complexity measure. Let us illustrate it by a simple example known as the *counter*.

## 1 Simple example

Our data structure will be a counter with $\log n$ bits which will be initialized to 0. The counter has only one operation $A_1$ which is to increase the counter by 1 (it can count to $n$). Every flip of a bit in the counter costs 1 unit (that is a change from 1 to 0 or vias versa).

**Worst case complexity:** in the worst case the operation $A_1$ (increase counter) costs $\log n$ units (if the counter is in the state 01111111111). Using this bound we deduce that the cost of a sequence of $k$ operations is $k \log n$.

We will now see that the bound derived using the 'worst case' measure of a single operation is to pessimistic. In other words, from time to time the worst case will happen but on average the cost is small. Another way to think of it is that if some operation was 'heavy' then this implies that the next few operations will be 'lite'. Let us calculate the total cost in a different way. We consider a sequence of $k$ operations ($k \leq n$), and count for every bit of the counter how many times does it flip. Notice that the LSB flips every round, the next bit flips every two rounds and in general the $i$-th bit (starting from the LSB) flips every $2^i$ rounds. It thus follows that the total number of flips is bounded by:

$$\sum_{i=1}^{k} \frac{k}{2^i} \leq 2k$$

(we assume $k$ is power of 2, this is without loss of generality since otherwise we replace $k$ by $k'$ the nearest power of 2 which is greater than $k$, since $k' < 2k$ we loose only a factor of 2).

## 2 Amortized comlexity

**Definition 2.1.** Let $DS$ be a data structure which supports the operations $A_1, A_2, ..., A_k$. We say that the set of operations $\{A_i\}_{i=1}^k$ have an amortized cost of $\{\hat{c}_i\}_{i=1}^k$ if the following holds:
For every sequence of operations $A_{i_1}, A_{i_2}, ..., A_{i_n}$ performed on the data structure (after $DS$ had been initialized) the total cost of this sequence is bounded by $\sum_{j=1}^{n} \hat{c}_{i_j}$ ($i_j$ is an index from $\{1, .., k\}$ for every $1 \leq j \leq n$).

Intuitively this definition says that operation $A_i$ costs $\hat{c}_i$, when considering sequences of operations. Possibly the amortized costs $\{\hat{c}_i\}_{i=1}^k$ will be a function of the number of operations $n$ (or the number of items in $DS$). For example $\hat{c}_1$ can be $O(\log n)$ or some constant independent of $n$.

In the counter data structure, the amortized cost of the (only) operation $A_1$ is 2.

# 3 The potential function method

Let $DS$ be a data structure with operations $A_1, ..., A_k$. There are many ways to choose the numbers $\hat{c}_i$, we would like to choose them "as low as possible". Sometimes we will not be able to choose all of them low, one of them will have to be high (depending on the data structure). Many times there is a trade off between the amortized costs $\hat{c}_i$: if we make one of them, say $\hat{c}_j$ low then we have to pay by making some other $\hat{c}_i$ high.

The potential function is a method which helps in 'guessing' correct amortized costs $\hat{c}_i$. We start by defining a potential function $\phi : \{D_i\}_{i \in I_n} \to \mathbb{R}^+$. The set $\{D_i\}_{i \in I_n}$ is the set of all possible states of $DS$ during any sequence of $n$ operations (this set is finite and increases with $n$). The advantage of this method is that after we derive the amortized costs $\hat{c}_i$ from $\phi$ (we will show how to do it) we are guaranteed that these numbers are correct i.e. for every sequence of $n$ operations $A_{i_1}, ..., A_{i_n}$ (operating on an initialized data structure) the total cost of this sequence is bounded by $\sum_{j=1}^n \hat{c}_{i_j}$. Notice that different potential functions will give different sets of amortized costs $\{\hat{c}_i\}_{i=1}^k$.

We start with an example which demonstrates the idea, but still does not solve the problem of deriving the amortized costs $\hat{c}_i$ from $\phi$. Then we will modify the example to get the full solution. Fix a sequence of operations $A_{i_1}, ..., A_{i_n}$, denote the sequence of internal states of the data structure by $D_0, D_1, ..., D_n$. A natural way to present it is:

$$D_0 \xrightarrow{A_{i_1}} D_1 \xrightarrow{A_{i_2}} D_2 . \; . \; . \; . \; . D_{n-1} \xrightarrow{A_{i_n}} D_n$$

We demand that $\phi(D_0) = 0$ since $D_0$ is the internal state of a the data structure after initialization (Intuitively the potential function says 'how dangerous' is the internal state. Since the data structure is 'empty' it is not dangerous). For each $j = 1, .., n$ we define the amortized cost of operation $A_{i_j}$ by:

$$\hat{c}_j = cost(D_{j-1} \xrightarrow{A_{i_j}} D_j) + \phi(D_j) - \phi(D_{j-1})$$

Notice that $\hat{c}_j$ is influenced by two things: the actual cost of the $j$-th operation in the sequence and the change in the data structure potential before and after the operation. If the operation reduced the potential of the data structure, then the amortized cost $\hat{c}$ benefit from it. On the other hand if the operation increased the potential then the amortized cost suffers from it. If we sum the amortized costs of all operations $j = 1, ..., n$ in the sequence we get:

$$\sum_{j=1}^n \hat{c}_j = \sum_{j=1}^n cost(D_{j-1} \xrightarrow{A_{i_j}} D_j) + \phi(D_n) - \phi(D_0) \tag{1}$$

Since $\phi(D_n) - \phi(D_0) \geq 0$ we conclude that the total actual cost is bounded by the sum of amortized costs.

The problem with the last assignments of amortized costs is that we gave an amortized cost to each operation of a fixed sequence, i.e. the amortized costs depend on the (fixed) sequence

$A_{i_1}, ..., A_{i_n}$ while by the definition of amortized costs we should give a cost for each type of operation $A_i$. To get rid of the dependency on the unique sequence (so that $\hat{c}_i$ will depend only on $A_i$ and not on the sequence) we define:

$$i = 1, .., k \qquad \hat{c}_i = \max_{\substack{D_j, D_l: \\ D_j \xrightarrow{A_i} D_l}} cost(D_j \xrightarrow{A_i} D_l) + \phi(D_l) - \phi(D_j)$$

(the states $D_j, D_l$ are from the set $\{D_i\}_{i \in I_n}$ of all possible internal states of $DS$ during a $n$ long sequence of operations). The last definition says: take $\hat{c}_j$ to be the worst case over all possible transitions of $A_j$ in (all possible) $n$ long sequences of operations.

Computing the exact value of the above maximum is generally hard, so we will usually bound this maximum from above and define $\hat{c}_i$ to be the this upper bound (this does not pose a problem, instead of summing equalities in equation (1) we will sum inequalities) . The upper bound on the above maximum will sometimes be a function $n$, thus also $\hat{c}_i$ may be a function of $n$.