# CS3230 Chapter 8 - Amortized Analysis

Based on lectures by Chang Ee-Chien
Notes taken by Andrew Tan
AY18/19 Semester 1



## 1 Amortized Analysis

In an **amortized analysis**, we average the time required to perform a sequence of data-structure operations over all the operations performed. With amortized analysis, we can show that the average cost of an operation is small, if we average over a sequence of operations, even though a single operation within the sequence might be expensive. Amortized analysis differes from average-case analysis in that probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst case.

There are three common techniques used in amortized analysis: aggregate analysis, accounting method, and potential method.

### 1.1 Aggregate analysis

With **aggregate analysis**, we determine an upper bound $T(n)$ on the total cost of a sequence of $n$ operations. The average cost per operation is thus $T(n)/n$. Note that this amortized cost applies to each operation, even when there are several types of operations in the sequence.

### 1.2 Accounting method

In the **accounting method**, we assign differing charges to different operations, with some operations charged more or less than they actually cost. We call the amount we charge an operation its **amortized cost**. When an operation's amortized cost exceeds its actual cost, we assign the difference to specific objects in the data structure as *credit*. Thus, we can view the amortized cost of an operation as being split between its actual cost and credit that is either deposited or used up. Different operations may have different amortized costs.

If we denote the actual cost of the $i^{th}$ operation by $c_i$ and the amortized cost of the $i^{th}$ operation by $\hat{c}_i$, we require

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

for all sequences of $n$ operations. The total credit stored in the data structure is the difference between the total amortized cost and the total actual cost, or $\sum_{i=1}^{n} \hat{c}_i - \sum_{i=1}^{n} c_i$. By the inequality above, the total credit associated with the data structure must be nonnegative at all times.

### 1.3 Potential method

Instead of representing prepaid work as credit stored with specific objects in the data structure, the **potential method** represents the prepaid work as "potential energy" or simply "potential", which can be released to pay for future operations. We associate the potential with the data structure as a whole rather than with specific objects within the data structure.

The potential method works as follows: we will perform $n$ operations, starting with an initial

data structure $D_0$. For each $i = 1, 2, \ldots, n$, we let $c_i$ be the actual cost of the $i^{th}$ operation and $D_i$ be the data structure that results after applying the $i^{th}$ operation to data structure $D_{i-1}$. A **potential function** $\Phi$ maps each data structure $D_i$ to a real number $\Phi(D_i)$, which is the potential associated with data structure $D_i$. The amortized cost $\hat{c}_i$ of the $i^{th}$ operation with respect to potential function $\Phi$ is defined by

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

The amortized cost of each operation is therefore its actual cost plus the change in potential due to the operation. Thus the total amortized cost of the $n$ operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)$$

Hence, if we can define a potential function $\Phi$ so that $\Phi(D_n) \geq \Phi(D_0)$, then the total amortized cost $\sum_{i=1}^{n} \hat{c}_i$ gives an upper bound on the total actual cost $\sum_{i=1}^{n} c_i$.