Design and Analysis of Algorithms: Lecture 5

Ben Chaplin

Contents

1 Amortization

How can we measure the "cost" of an algorithm when an operation is slow in only some cases?

For example, recall **table-doubling**, the process of scaling up a hash table when its capacity is full. Most insertions are $O(1)$, but when the table needs to be doubled, the rehashing takes linear time. However, this table-doubling does not happen often, and crucially—does not happen until a certain number of insertions have already been made.

There are many methods of defining this **amortized cost**. Let's look at a few.

1.1 Aggregate method

Definition. The **amortized cost** of an operation is equal to:

$$\frac{\text{the total cost of } k \text{ operations}}{k}$$

Example. Take table-doubling, starting with an empty structure on which we must perform n insertions.

Definition. The **load factor** is defined as $\frac{m}{n}$, where n is the number of elements in the table and m is the size of the table.

We double the table when the load factor grows to m. Then, the runtime of n insertions is:

$$\Theta(1 + 2^0 + 1 + 1 + 2^1 + \ldots + 2^{\log n}) = \Theta(n)$$

Where the 1's are plain insertions, and the factors of 2 are the respective doublings. Therefore the **amortized cost** of n insertions is:

$$\frac{\Theta(n)}{n} = \Theta(1)$$

1.2 Accounting method

For each operation, store "credit" in a "bank account"—the main rule is that this bank account must maintain a positive balance. Then, we can allow future operations to pay their cost using credit from the bank.

Definition. The **amortized cost** of an operation is equal to:

$$\text{the real cost of the operation} - \text{the credit used}$$

Example. Back to table-doubling—consider insertions: when inserting, add a "coin" worth $\Theta(1)$. Then, whenever we need to double the table:

- we need n coins,
- and we have $\frac{n}{2}$ coins.

Thus, the **amortized cost** of an insert is:

$$\Theta(n) - \Theta(1) \cdot \frac{n}{2} = \Theta(1)$$

1.3 Potential method

Definition. A **potential function** is a function $\Phi : \{\text{states}\} \to \mathbb{Z}^+$ giving a positive integer value to each state (or configuration) of a data structure.

Definition. The **amortized cost** of an operaiton is equal to:

$$\text{the actual cost of the operation} + \Phi(\text{after state}) - \Phi(\text{before state})$$

As a result of the above definition:

$$\sum \text{amortized costs} = \sum \text{actual costs} + \Phi(\text{end state}) - \Phi(\text{beginning state})$$

Example. Take a binary counter with an "increment" operation (e.g. $0011010111 \to 0011011000$). We define the potential function:

$$\Phi(x) = \text{the number of 1's in } x$$

Let t be the number of trailing 1's in x. Then, an increment operation destroys t 1's and creates one 1. Thus, the **amortized cost** of an increment:

$$\begin{aligned}
&= \text{the actual cost of the operation} + \Phi(\text{after state}) - \Phi(\text{before state}) \\
&= (1 + t) + (-t + 1) \\
&= \Theta(1)
\end{aligned}$$