

## Amortization

Reading: CLRS Chapter 17/CLR Chapter 18

---

### Motivating Amortized Analysis

In many situations, the worst-case running time of a *sequence* of data structure operations is asymptotically smaller than the sum of the worst-case running times for the individual operations.

Why? Intuitively, a single costly operation is often balanced by a large number of cheap operations.

**Amortized analysis** is a collection of techniques for analyzing the worst-case running times of sequences of operations. Amortized analysis involves the average running time of operations in a sequence, but it is *not* probabilistic in nature.

---

### Example: Binary Counters

Given an array  $A$  whose slots contain either 0 or 1, the  $\text{Inc}(A)$  procedure below increments the binary number represented by  $A$ :

```
▷ Increment the binary number denoted by  $A$ .
▷ Assume indexing of  $A$  is 0-based, right-to-left.
Inc( $A$ )
 $i \leftarrow 0$ 
while ( $i < \text{length}[A]$ ) and  $A[i] = 1$  do
  flip( $A[i]$ ) ▷ Flip the bit value from 1 to 0
   $i \leftarrow i + 1$ 
if  $i < \text{length}[A]$  then
  flip( $A[i]$ ) ▷ Flip the bit value from 0 to 1
```

We can add  $n$  to the number denoted by  $A$  by incrementing  $n$  times:

```
▷ Add  $n$  to the binary number denoted by  $A$ .
▷ Assume indexing of  $A$  is 0-based, right-to-left.
Add( $n, A$ )
for  $j \leftarrow 1$  to  $n$  do
  Inc( $A$ )

▷ Count to  $n$  in binary using binary numbers with  $k$  digits
Count( $n, k$ )
 $A \leftarrow \text{new array}(k)$ 
Add( $n, A$ )
```

*Question:* What is the cost (measured in number of flips) of  $\text{Count}(n, k)$ ?

*Loose Analysis:* A call to  $\text{Inc}(A)$  could have worst-case cost  $O(k)$ , so  $\text{Count}(n, k)$  has cost  $O(n \cdot k)$ .

*Tighter Analysis (justified below):*  $\text{Count}(n, k)$  has cost  $O(n)$ , so amortized cost of  $\text{Inc}(A)$  is  $O(1)$ .

---

## Aggregate Method

In the **aggregate method**, we determine a worst-case cost  $T(n)$  for a sequence of  $n$  operations. The amortized cost of each operation is then  $T(n)/n$ .

*Binary counter example:*

The following shows the bits flipped in  $\text{Count}(17, 8)$ :

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 0 1 1
0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1
0 0 0 0 1 0 0 0
0 0 0 0 1 0 0 1
0 0 0 0 1 0 1 0
0 0 0 0 1 0 1 1
0 0 0 0 1 1 0 0
0 0 0 0 1 1 0 1
0 0 0 0 1 1 1 0
0 0 0 0 1 1 1 1
0 0 0 1 0 0 0 0
0 0 0 1 0 0 0 1
```

Observe that bit position  $i$  (0-indexed, right-to-left) is flipped every  $2^i$  calls to  $\text{Inc}$ . So the total number of flips  $F(n, k)$  performed in  $\text{Count}(n, k)$  is:

$$F(n, k) = \sum_{i=0}^k \lfloor n/2^i \rfloor \leq n \cdot \sum_{i=0}^{\infty} \lfloor 1/2^i \rfloor =$$

So the amortized cost  $I(k)$  of a call to  $\text{Inc}(A)$  on length- $k$  array  $A$  is:

---

## Banker's Method (Accounting Method)

In the **banker's method (accounting method)**, some operations can “deposit” credits in particular positions in a data structure that are used to pay for other operations. The credits are used only for analyzing algorithms; the algorithm is *not* changed to include fields for the credits!

*Binary counter example:* Suppose that flipping a bit costs \$1. In  $\text{Inc}$ , suppose that we charge \$2 to flip a bit from 0 to 1: \$1 is used to pay for the bit flip, and a \$1 credit remains on the 1 bit. We pay for every flip from 1 to 0 using the \$1 credit already “attached” to the 1 bit. Clearly,  $\text{Inc}(A)$  costs \$2, and  $\text{Count}(n, k)$  costs \$2n.

Unlike the aggregate method, the banker's method allows us to assign different amortized costs to different operations.

---

## Physicist's Method (Potential Method)

In the **physicists's method (potential method)**, operations can increase and decrease a non-zero potential associated with a data structure. This is similar to the banker's method, except that the potential serves as a "credit" for the whole data structure, rather than a particular position in it.

*Binary counter example:* Define the potential  $\Phi(A)$  of an array  $A$  containing 0s and 1s as:

$$\Phi(A) = \text{the number of 1s in } A.$$

Then the amortized cost  $\hat{c}_i$  of the  $i$ th call to `Inc` is the actual cost  $c_i$  plus the change in potential:

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

Suppose the  $i$ th call to `Inc` flips  $t_i$  bits from 1 to 0. Then the real cost of this call is:

$$c_i = 1 + t_i$$

and the change in potential is

$$\Phi(A_i) - \Phi(A_{i-1}) = 1 - t_i$$

so the amortized cost of the  $i$ th call to `Inc` is

$$\hat{c}_i = 2$$

The sum of the total amortized costs is

$$\sum_{i=0}^n \hat{c}_i = \sum_{i=0}^n (c_i + \Phi(A_i) - \Phi(A_{i-1})) = \left(\sum_{i=0}^n c_i\right) + \Phi(A_n) - \Phi(A_0) = \left(\sum_{i=0}^n c_i\right) + b_n$$

where  $b_n$  is the number of 1 bits in  $A_n$ . So

$$\sum_{i=0}^n c_i = \left(\sum_{i=0}^n \hat{c}_i\right) - b_n = 2n - b_n$$