Amortized Analysis

October 2020

The problem solved by amortized analysis

Some algorithms on data structures have operations with varying time complexities

long-run computation C = sequence of operations $o_1, o_2, \ldots, o_n$

- the operations are often fast, but from time to time they are slow

$\Rightarrow$ The worst-case runtime estimate $T(C) = \sum_{i=1}^{n} T(o_i)$ where $T(o_i)$ where $T(o_i)$ are the worst-case runtime estimates of o_i is too inaccurate.

The problem solved by amortized analysis

Some algorithms on data structures have operations with varying time complexities

long-run computation C = sequence of operations $o_1, o_2, \ldots, o_n$

- the operations are often fast, but from time to time they are slow

$\Rightarrow$ The worst-case runtime estimate $T(C) = \sum_{i=1}^{n} T(o_i)$ where $T(o_i)$ where $T(o_i)$ are the worst-case runtime estimates of o_i is too inaccurate.

Amortized analysis is a better method to estimate the time complexity of many operations at once.

Case study: Dynamic arrays

An array A with 2^n elements

`insert` (A, x) : inserts item x at the next free position in the array

- we double the size of A when it becomes full, and copy all elements from the old array to the newly created array $\rightarrow$ time consuming operation

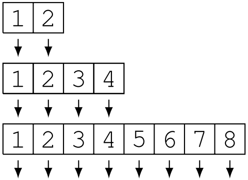
Case study: Dynamic arrays

An array A with 2^n elements

`insert`(A, x) : inserts item x at the next free position in the array

- we double the size of A when it becomes full, and copy all elements from the old array to the newly created array
  $\rightarrow$ time consuming operation

Analyzing the time complexity:



$\Rightarrow$ time complexity of i-th operation `insert`(A, x) :

- $O(2^n)$ if $i = 2^n$
- $O(1)$, otherwise.

Number of elementary operations (assignment or copy) required by the i-th operation `insert(`A, x`)`:

item number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
elem. ops.	1	2	1	4	1	1	1	8	1	1	1	1	1	1	1	16

Number of elementary operations (assignment or copy) required by the i-th operation `insert(`A, x`)`:

item number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
elem. ops.	1	2	1	4	1	1	1	8	1	1	1	1	1	1	1	16

- **Amortized analysis** with the aggregate method
  - computes an upper bound of the total cost $T(n)$ of a sequence of n operations. In our example

$$\begin{aligned} T(n) &\leq \sum_{i=1}^{n} 1 + \sum_{i=1}^{\lfloor \log_2 n \rfloor} 2^i = n + (2^{\lfloor \log_2 n \rfloor + 1} - 1) \\ &\leq 2 + ((n+1) - 1) = 2 \cdot n \end{aligned}$$

$\Rightarrow$ $T(n)/n$ is called the amortized cost per operation:

$$\frac{T(n)}{n} = \frac{2 \cdot n}{n} = 2 = O(1)$$

$\Rightarrow$ amortized cost is constant.
$\Rightarrow$ this cost applies to each operation, even if there are several types of operations in the sequence.

Other methods for amortized analysis

There are three frequently-used methods for amortized analysis:

1. aggregate method
2. accounting method
3. potential method

Accounting method

Assigns different amortized costs to different operations.

- The amortized cost depends on i (the i-th iteration) and may differ from the actual cost.
  - With aggregate method, all operations are assumed to have same amortized cost. With accounting method, they can have different amortized costs.
- When an operation's amortized cost exceeds its actual cost, the surplus goes into a *"bank"*.
- Idea: Need to **overcharge** for simpler operations, to build up enough **savings** to afford a more expensive operation later
  - Bank balance must always be ≥ 0

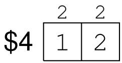
Accounting method

Ilustrated example: Dynamic array

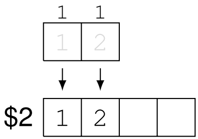
Charge 3 units per operation

$0 

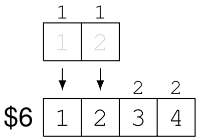
Charge 3 units per operation



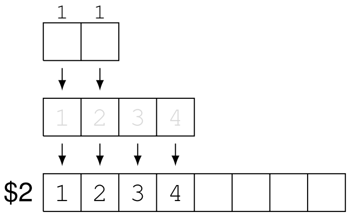
Charge 3 units per operation



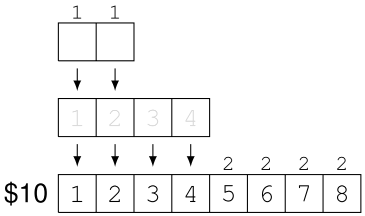
Charge 3 units per operation



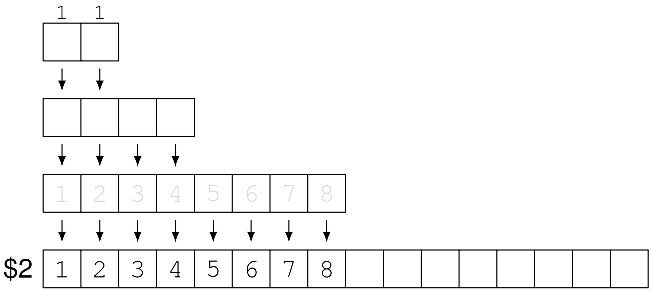
Charge 3 units per operation



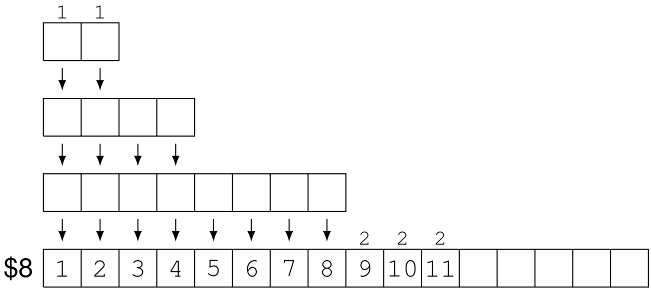
Charge 3 units per operation



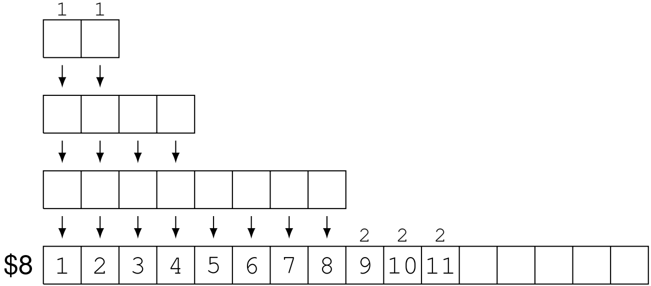
Charge 3 units per operation



Charge 3 units per operation



Charge 3 units per operation



- Bank balance is always ≥ 2

$\Rightarrow$ each operation has constant amortized cost 3.

Potential method

Conceptually similar to accounting method

- Same idea of using stored surplus to pay for more expensive operations
- The potential (energy) is analogous to the bank
- Potential must always ≥ 0

Key differences:

- Accounting method: bank balance of a particular state is **dependent** on previous state
- Potential method involves a potential function $\Phi(A)$
  - can be used to derive the potential at any state
  - can also be used to compute a potential difference, which shows the change in cost between two operations

Potential method

Conceptually similar to accounting method

- Same idea of using stored surplus to pay for more expensive operations
- The potential (energy) is analogous to the bank
- Potential must always ≥ 0

Key differences:

- Accounting method: bank balance of a particular state is **dependent** on previous state
- Potential method involves a potential function $\Phi(A)$
  - can be used to derive the potential at any state
  - can also be used to compute a potential difference, which shows the change in cost between two operations

Finding a potential function is the challenging part!

$\Phi(A) = 2\,(i+1) - \mathit{size}(A)$ where i is the number of elements in A

- Easy to check that $\Phi(A)$ is always ≥ 0.

The amortized cost $\hat{c}_i$ of the i-th operation is defined to be
$\hat{c}_i = c_i + \Phi(A_i) - \Phi(A_{i-1})$ where

- c_i is the actual cost of the i-th operation
- A_i is the dynamic array after the i-th operation

We distinguish 2 cases:

1. i-th step is normal step. Then $c_i = 1$, $\mathit{size}(A_i) = \mathit{size}(A_{i+1})$ and
   $\hat{c}_i = 1 + (2\,(i+1) - \mathit{size}(A_i)) - (2\,i - \mathit{size}(A_i)) = 3$
2. i-th step is an expansion step. This happens when $\mathit{size}(A_i) = i$ and $\mathit{size}(A_{i+1}) = 2\,i$. Then $c_i = i$ and
   $\hat{c}_i = i + (2\,(i+1) - 2\,i) - (2\,i - i) = 3$

$\Rightarrow$ every operation has constant amortized cost 3.

Summary

- Aggregate analysis defines amortized cost of any operation as an average:

$$\frac{\text{total cost of a sequence of operations}}{\text{number of operations}}$$

- Potential and Accounting methods involve assigning amortized costs per operation
  - Cheap operations are **overcharged** $\Rightarrow$ we acquire a surplus
  - Expensive operations are paid for by the **surplus**