Lecture 1: Amortization

CS-GY 6043: Design & Analysis of Algorithms II
Sept. 7th, 2023
Scribed by Kevin Tan

1. Amortization

- Amortization is used to bound the cost of a **sequence** of operations.
- The worst-case total cost is no more than the number of operations multiplied by the amortized cost of each operation.
  - Let c_i be the real cost of each operation, $\hat{c}_i$ be the amortized cost of each operation, then we want $\sum c_i \leq \sum \hat{c}_i$.
- There are three ways to find out an amortized cost:
  - Accounting Method
  - Aggregate Analysis
  - Potential Method

2. `Multipop` Example

Let's consider a stack supporting the following operations:

Operation	Constraint	Cost
`push(x)`	None	1 unit of work
`pop()`	The stack is not empty	1 unit of work
`multipop(k)`	$k <$ the number of elements in the stack	k units of work

Question: What is the amortized cost of a sequence of n `push`, `pop`, and `multipop` operations in total?

It is easy to come up with the answer that equals the number of operations multiplied by the max cost of operations, i.e., $n \times n = n^2$. However, it is too broad because `pop` and `multipop` cannot be used without enough elements that have been `push`ed. The more precise answer is $O(n)$.

Let's reconsider it using the accounting method.

- When we `push(x)` into the stack, we can regard it as spending $1 for the real cost of `push` and will put $1 away for the future; therefore, the amortized cost of `push` is $2.
- When we `pop`, although we spend a real $1 for `pop`, we pay for it from a previous `push` that has already been considered; therefore, the amortized cost of `pop` is $1 - $1 = 0.
- Likewise, when we `multipop(k)`, although we are spending in real cost $\$k$, we are using $k \times \$1$ in credit; therefore, the amortized cost of `multipop` is $k - k = 0$.

- Thus, the total cost is $O(n)$.

The aggregate analysis can be more straightforward.

- Reordering all the operations, we realize that the number of elements `push`ed must be greater than or equal to the number of elements popped or multi-popped.
- Therefore, there are almost n items pushed and almost n items popped. Since pushing an item and popping an item costs $1, the total cost is no more than $2n$, which is $O(n)$.

3. Binary Counter Example

Let's consider an array A with k bits. We want to increment the array with cost being the amount of bit flips, as shown in the figure below.

```
A[k-1]      ...      A[3]  A[2]  A[1]  A[0]  cost
  0         ...       0     0     0     0
                                              $1
  0         ...       0     0     0     1
                                              $2
  0         ...       0     0     1     0
                                              $1
  0         ...       0     0     1     1
                                              $3
  0         ...       0     1     0     0
                                              $1
```

It is easy to realize that the worst case of increment is to have all k bits flipped, so the total cost of n increments is $O(nk)$. However, this estimate is too conservative because this worst case is very unlikely to happen.

Let's reconsider it using the aggregate analysis.

- Consider all operations as a whole. Bit 0 changes n times. Bit 1 changes $\lfloor \frac{n}{2} \rfloor$ times. Bit 2 changes $\lfloor \frac{n}{4} \rfloor$ times. Bit 3 changes $\lfloor \frac{n}{8} \rfloor$ times. Bit i changes $\lfloor \frac{n}{2^i} \rfloor$ times. Evaluating the sum as a geometric series with the formula, we have the total amount of operations

$$\sum_i \lfloor \frac{n}{2^i} \rfloor \leq n \sum_{i=0}^{\infty} \frac{1}{2^i} = n \frac{1}{1-\frac{1}{2}} = 2n.$$

  Thus, the total cost of n increment operations is $2n$, which is $O(n)$ in asymptotic form.

4. Potential Method

- It associates with the data structure a quantity we call φ, the potential function.
- It has two requirements:
  - $\varphi_0 = 0$, which is the value of φ before operation 1.
  - $\varphi \geq 0$, among which φ_i is the value of φ after operation i.
- For operation i, the amortized cost of each operation $\hat{c_i} = c_i + \Delta\varphi_i = c_i + (\varphi_i - \varphi_{i-1})$, where c_i is the real cost of each operation. Then, we have

$$
\begin{aligned}
\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} [c_i + (\varphi_i - \varphi_{i-1})] \\
&= \sum_{i=1}^{n} (c_i + \varphi_n - \varphi_0) \\
&= \sum_{i=1}^{n} (c_i + \varphi_n) \\
&\geq \sum_{i=1}^{n} c_i,
\end{aligned}
$$

so the sum of amortized costs bounds the sum of real costs, as desired.

- If we can find such a function for a problem, then we can use it to calculate the amortized cost. However, if we cannot find one, we cannot conclude anything.

5. Doubling Dynamic Array Example

We want to store things in an array A of size s but do not want to use a lot more space than the actual number of items stored (n).

An implementation of `insert(x)` can be:

```
if n < s:
    append x to A
else:
    alloc new array of size s := 2s
    copy old data into it, call it A, free old array
    append x to A
    n := n + 1
```

The cost of `insert` is 1 for inserting the new element n if copying the n old elements is needed, i.e., sometimes $1 + n$.

Let's find out its amortized cost using the potential method.

- We want the expensive operation to have a cheaper cost and the cheap operation becomes expensive. $\varphi = 2n - s$ is a valid potential function since we always have $n \geq \frac{s}{2}$, $\varphi = 2n - s \geq 0$. We assume $\varphi_0 = 0$ as $n = s = 0$ at that time. This is true because the actual code is more complicated than the above pseudo code if we handle the special case of an empty structure where $n = s = 0$.
- To compute the amortized cost, we consider the good case first. The real cost is 1, $\varphi = [2(n + 1) - s] - [2n - s] = 2$, amortized cost $= 1 + 2 = 3$.
- Then, we consider the bad case that needs doubling. The real cost is $1 + n$, $\varphi = [2(n + 1) - 2s] - [2n - s] = 2 - s$, amortized cost $= 3 + n - s = 3$ because $n = s$ in bad cases.
- Thus, the amortized cost is $O(1)$ in asymptotic form.
- Note that if φ is different, comparing the numerical values of the amortized cost calculated from different φs is meaningless. For example, if $\varphi = 4n - s$, the amortized cost is 5, but it does not affect the conclusion that the amortized cost is $O(1)$ for this problem.

We can also solve this problem using the aggregate method.

- Consider all operations as a whole. The total cost is n insertions and all copying amount on doubling, i.e.,

$$n + 1 + 2 + 4 + \cdots + 2^k = n + 2^{k+1} - 1,$$

  where 2^k is the last time doubling,

$$2^k < n.$$

  Multiplying it by 2, we have

$$2^{k+1} < 2n.$$

- Thus, the total cost is

$$n + 2^{k+1} - 1 < 3n - 1,$$

  and the amortized cost is $\frac{O(n)}{n} = O(1)$.

> Question: How to find a potential function in general?
>
> Answer: It needs experience, inspiration, and trial-and-error loops.