



Indian Institute of Information Technology
Design and Manufacturing, Kancheepuram
Chennai 600 127, India
An Autonomous Institute under MHRD, Govt of India
<http://www.iiitdm.ac.in>
COM 501 Advanced Data Structures and Algorithms

Instructor
N.Sadagopan
Scribe:
Pranjal Choubey
Renjith.P

Amortized Analysis

Objective: In this lecture, we shall present the need for amortized analysis, and case studies involving amortized analysis.

Motivation: Consider data structures Stack, Binomial Heap, Min-Max Heap; stack supports operations such as push, pop, multipush and multipop, and heaps support operations such as insert, delete, extract-min, merge and decrease key. For data structures with many supporting operations, can we look for an analysis which is better than classical asymptotic analysis. Can we look for a micro-level analysis to get a precise estimate of cost rather than worst case analysis.

1 Amortized Analysis

Amortized analysis is applied on data structures that support many operations. The sequence of operations and the multiplicity of each operation is application specific or the associated algorithm specific. Classical asymptotic analysis gives worst case analysis of each operation without taking the effect of one operation on the other, whereas amortized analysis focuses on a sequence of operations, an interplay between operations, and thus yielding an analysis which is precise and depicts a micro-level analysis.

Since many operations are involved as part of the analysis, the objective is to perform efficiently as many operations as possible, leaving very few costly operations (the time complexity is relatively more for these operations). To calculate the cost of an operation or the amortized cost of an operation, we take the average over all operations. In particular, worst case time of each operation is taken into account to calculate the average cost in the worst case. Some of the highlights of amortized analysis include;

- Amortized Analysis is applied to algorithms where an occasional operation is very slow, but most of the other operations are faster.
- In Amortized Analysis, we analyze a sequence of operations and guarantee a worst case average time which is lower than the worst case time of a particular expensive operation.
- Amortized analysis is an upper bound: it is the average performance of each operation in the worst case. Amortized analysis is concerned with the over all cost of a sequence of operations. It does not say anything about the cost of a specific operation in that sequence.
- Amortized analysis can be understood to take advantage of the fact that some expensive operations may pay for future operations by somehow limiting the number or cost of expensive operations that can happen in the near future.
- Amortized analysis may consist of a collection of cheap, less expensive and expensive operations, however, amortized analysis (due to its averaging argument) will show that average cost of an operation is cheap.
- This is different from average case analysis, wherein averaging argument is given over all inputs for a specific operation. Inputs are modeled using a suitable probability distribution. In amortized analysis, no probability is involved.

2 The Basics: the three approaches to amortized analysis

It is important to note that these approaches are for analysis purpose only. The underlying algorithm design is unaltered and the purpose of these micro-level analysis is to get a good insight into the operations being performed.

- **Aggregate Analysis:** Aggregate analysis is a simple method that involves computing the total cost $T(n)$ for a sequence of n operations, then dividing $T(n)$ by the number n of operations to obtain the amortized cost or the average cost in the worst case. For all operations the same amortized cost $T(n)/n$ is assigned, even if they are of different types. The other two methods may allow for assigning different amortized costs to different types of operations in the same sequence.
- **Accounting Method:** As part of accounting method we maintain an account with the underlying data structure. Initially, the account contains '0' credits (charges). When we perform an operation, we charge the operation, and if we over charge an operation, the excess charge will be deposited to the account as credit. For some operation, we may charge nothing, in such a case, we make use of charges available at credit. Such operations are referred to as *free* operations. Analysis ensures that the account is never at debit (negative balance). This technique is good, if for example, there are two operations O_1 and O_2 which are tightly coupled, then O_1 can be over charged and O_2 is free. Typically, the charge denotes the actual cost of that operation. The excess charge will be stored at objects (elements) of a data structure.

In the accounting method, the amount charged for each operation type is the amortized cost for that type. As long as the charges are set so that it is impossible to go into debt, the amortized cost will be an upper bound on the actual cost for any sequence of operations. Therefore, the trick to successful amortized analysis with the accounting method is to pick appropriate charges and show that these charges are sufficient to allow payment for any sequence of operations.

- **Potential Function Method:** Here, the analysis is done by focusing on structural parameters such as the number of elements, the height, the number of property violations of a data structure. For an operation, after performing the operation, the change in a structural parameter is captured as a function which is stored at a data structure. The function that captures the change is known as a *potential function*. As part of the analysis, we work with non-negative potential functions. If the change in potential is positive, then that operation is over charged and similar to accounting method, the excess potential will be stored at the data structure. If the change in potential is negative, then that operation is under charged which would be compensated by excess potential available at the data structure.

Formally, let c_i denote the actual cost of the i^{th} operation and \hat{c}_i denote the amortized cost of the i^{th} operation. If $\hat{c}_i > c_i$, then the i^{th} operation leaves some positive amount of credit, the credits $\hat{c}_i - c_i$ can be used up by future operations. And as long as

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

the total available credit will always be non negative, and the sum of amortized costs will be an upper bound on the actual cost. The potential function method defines a function that maps a data structure onto a real valued non-negative number. In the potential method, the amortized cost of operation i is equal to the actual cost plus the increase in potential due to that operation:

$$\hat{c}_i = c_i + \phi_i - \phi_{i-1} \quad (2)$$

From equation 1 and 2:

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \phi_i - \phi_{i-1}) \quad (3)$$

$$\sum_{i=1}^n \hat{c}_i = \langle \sum_{i=1}^n c_i \rangle + \phi_n - \phi_0 \quad (4)$$

3 Case Studies

We shall present three case studies to explain each analysis technique in detail.

Stack

Our stack implementation consists of three operations, namely $\text{push}()$, $\text{pop}()$ and $\text{multi-pop}()$. The operation $\text{multi-pop}(k)$ fetches top k elements of the stack if stack contains at least k elements. If stack contains less than k elements, then output the entire stack.

Let us analyze stack from the perspective of amortized analysis by assuming there are n of the above operations which are performed in some order. In classical world, the actual cost in worst case for push and pop is $O(1)$ and multi-pop is $O(n)$. We now show that the amortized cost of all three operations is $O(1)$.

Aggregate Analysis:

Let us assume that in a sequence of n operations, there are l multi-pop operations and the rest are push and pop operations. Clearly, l multi-pop operations perform at most n pops and therefore, the cost is $O(n)$. The cost for the other $n - l$ operations is $O(n)$. The amortized cost is $O(n) + O(n)$ divided by n which is $O(1)$. Since aggregate analysis assigns the same cost to all operations, push, pop and multi-pop incur $O(1)$ amortized cost.

Accounting Method:

As part of accounting method, we need to assign credits to objects of stack S . Whenever we push an element x into S , we charge '2' credits. One credit will be used for pushing the element which is the actual cost for pushing x into S and the other credit is stored at x which would be used later. For pop operation, we charge nothing, i.e., pop operation is free. Although, the actual cost is '1', in our analysis we charge '0'. The excess credit '1' available at the element will be used when we perform a pop operation. Since pop is performed on a non-empty stack, the account will always be at credit. Similarly, we charge nothing for multi-pop as sufficient credits are available for all k elements at elements themselves. Thus, the amortised cost of push is $2 = O(1)$, pop is $0 = O(1)$ and multi-pop is $0 = O(1)$.

Potential Function Method: The first thing in this method is to define a potential function capturing some structural parameter. A natural structural parameter of interest for stack is the number of elements. Keeping the number of elements as the potential function, we now analyze all three operations.

1. Push

$$\begin{aligned}\hat{c}_{push} &= c_{push} + \phi_i - \phi_{i-1} \\ &= 1 + x + 1 - x, \text{ where } x \text{ denotes the number of elements in } S \text{ before push operation.} \\ &= 2\end{aligned}$$

2. Pop

$$\begin{aligned}\hat{c}_{push} &= c_{push} + \phi_i - \phi_{i-1} \\ &= 1 + x - 1 - x \\ &= 0\end{aligned}$$

3. Multi-pop(k)

$$\begin{aligned}\hat{c}_{push} &= c_{push} + \phi_i - \phi_{i-1} \\ &= k + n - k - n, \text{ where } n = |S|. \\ &= 0\end{aligned}$$

Thus, the amortized cost of all three operations is $O(1)$. Note the similarity between this method and the accounting method. The potential function, in particular, the change in potential helps us to fix the excess value to be stored at each element of S . Also, note that potential functions such as $2 \cdot |S|$ or $c \cdot |S|$, where

c is a constant will also work fine and still yields $O(1)$ amortized cost. The excess potential will be simply stored at the data structure as credits. Similarly, in accounting method, if we charge, say 4 credits for push, then excess credits will be stored at elements.

Suppose, we introduce another operation, namely, multi-push(k) which pushes k elements into S . Let us analyze the cost of multi-push in all three techniques.

- Aggregate Analysis: In any sequence of n operations consisting of push, pop, multi-pop and multi-push, we may find l_1 multi-pop, l_2 multi-push and the rest are push and pop operations. In the worst case scenario, $l_2 = O(1)$ and each multi-push pushes $O(n)$ elements. For example, three multi-push with each push inserts $n/3$ elements. Therefore, the total cost is $O(n)$ and the amortized cost is $O(n)/O(1)$ which is $O(n)$. Thus, for all operations, the amortized cost is $O(n)$.
- Accounting Method: As before, we use '2' credits for push and '0' for pop. With this credit scheme, the cost of push, pop and multi-pop is $O(1)$ amortized, whereas, multi-push is $2 \cdot O(n)/O(1)$ which is $O(n)$. Therefore, the amortized cost of multi-push is $O(n)$.
- Potential Function Method: For multi-push(k);

$$\hat{c}_{push} = c_{push} + \phi_i - \phi_{i-1}$$

$$= k + n + k - n, \text{ where } n = |S|.$$

$$= 2 \cdot k = O(n).$$

Binary Counter

Consider a binary counter on k bits which is initially set to all 0's. The primitive operations are Increment, Decrement and Reset. Increment on a counter adds a bit '1' to the current value of the counter. Similarly, decrement on a counter subtracts a bit '1' from the current value of the counter. For example, if counter contains '001101', on increment, the content of the counter is '001110' and on decrement, the result is '001100'. The reset operation makes the counter bits all 0's. The objectives here are to analyze amortized costs of (i) a sequence of n increment operations, (ii) a sequence of n increment and decrement operations, (iii) a sequence of n increment, decrement and reset operations.

Amortized analysis: a sequence of n increment operations

Given a binary counter on k -bits, we shall now analyze the amortized cost of a sequence of n increment operations. An illustration is given in Figure 1. A trivial analysis shows $O(k)$ for each increment and for n increment operation, the total cost is $O(nk)$ and the average cost is $O(k)$. We now present a micro-level analysis using which we show that the amortized cost is $O(1)$. Note that not all bits in the counter flip in each increment operation.

The 0^{th} bit flips in each increment and there are $\lfloor n \rfloor$ flips. The 1^{st} bit is flipped alternately and thus $\lfloor \frac{n}{2} \rfloor$ flips in total. The i^{th} bit is flipped $\lfloor \frac{n}{2^i} \rfloor$ times in total. It is important to note that, for $i > \lfloor \log n \rfloor$, bit $A[i]$ never flips at all. The total number of flips in a sequence of n increments is thus

$$\sum_{i=0}^{\lfloor \log n \rfloor} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

The worst-case time for a sequence of n increment operations on an initially zero counter is therefore $O(n)$. The average cost of each operation, and therefore the amortized cost per operation, is $O(n)/n = O(1)$. This completes the argument for aggregate analysis.

As part of accounting method, we assign '2' credits with each bit when it is flipped from 0 to 1. '1' credit will be used for the actual flip and the other '1' credit will be stored at the bit itself. When a bit is flipped from 1 to 0 in subsequent increments, it is done for free. The credit '1' stored at the bit (1) will actually pay for

| Counter value | A[7] | A[6] | A[5] | A[4] | A[3] | A[2] | A[1] | A[0] | Total cost |
|---------------|------|------|------|------|------|------|------|------|------------|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 3 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 4 |
| 4 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 7 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 8 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 10 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 11 |
| 8 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 15 |
| 9 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 16 |
| 10 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 18 |
| 11 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 19 |
| 12 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 22 |
| 13 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 23 |
| 14 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 25 |
| 15 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 26 |
| 16 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 31 |

Figure 1: Source: CLRS, An 8-bit binary counter where the value goes from 0 to 16 by a sequence of 16 increment operations. Bits that flip to achieve the next value are shaded.

this operation. At the end of each increment, the number of 1's in the counter is the credit accumulated at the counter. The primitive operations are flipping a bit from 1 to 0 and 0 to 1. The former charges 2 credits which $O(1)$ and the latter charges 0 credits which is also $O(1)$. Thus, the amortized cost of increment is $O(1)$.

For potential function method, the structural parameter of interest is the number of 1's in the counter. During i^{th} iteration all ones after the last zero is set to zero and the last zero is set to 1. For example, when increment is called on a counter with its contents being '11001111', the result is '11010000'. Let x denotes the total number of 1's before the i^{th} operation and t denote the number of ones after the last zero. At the end of i^{th} operation there will be $x - t + 1$ ones, t ones are changed to 0 and the last zero is changed to 1. Thus, the actual cost for the increment is $1 + t$. Therefore, the amortized cost is

$$\begin{aligned}
\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\
&= 1 + t + (x - t + 1) - x \\
&= 2 \\
&= O(1)
\end{aligned}$$

Remark: Amortized analysis for a sequence of n decrement operations is similar to the analysis presented above and hence, a sequence of n decrements incur $O(1)$ amortized.

Amortized analysis: a sequence of n increment and decrement operations

Consider a sequence having $\frac{n}{2}$ increments followed by $\frac{n}{2}$ increments and decrements happen alternately. As far as aggregate analysis is concerned, the actual cost for the first half of the operations is $O(n)$, whereas the second half incurs $\frac{n}{2} \cdot O(k)$. Thus, the amortized cost is $O(k)$. In general, there are l increments in a row followed by alternating increment and decrement operations. This is an example of worst case sequence for which amortized cost is $O(k)$. Therefore, both increment and decrement incur $O(k)$ amortized cost.

As part of accounting method, we assign k credits for both increment and decrement operations. Therefore, the amortized cost for both operations is $O(k)$ amortized.

For potential function method, the function refers to the number of 1's in the counter. For increment operation, similar to the above section, let x and t denote the number of 1's in the counter and the number of 1's after the last zero. Thus, the amortised cost of increment is

$$\begin{aligned}\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= 1 + t + (x - t + 1) - x \\ &= 2 \\ &= O(1)\end{aligned}$$

For decrement, let x and t denote the number of 1's in the counter and the number of 0's after the last one. Thus, the amortised cost of $\hat{c}_i = c_i + \phi_i - \phi_{i-1}$

$$\begin{aligned}&= 1 + t + (x + t - 1) - x \\ &= 2t \\ &= O(k)\end{aligned}$$

Thus, the amortized cost of increment is $O(1)$ and that of decrement is $O(k)$.

Amortized analysis: a sequence of n increment, decrement and reset operations

Consider a sequence having $\frac{n}{2}$ increments followed by one reset operation. The total cost is $O(n) + O(k)$. Thus, aggregate analysis for the $\frac{n}{2}$ increments is $O(1)$ amortized and that of reset is $O(k)$ amortized. Therefore, the amortized cost for this sequence is $O(k)$ amortized.

We shall now analyze amortized analysis of reset operation using potential function method. The actual cost of reset is $O(k)$ and assume that there are x 1's in the current configuration of the counter.

$$\begin{aligned}\hat{c}_i &= c_i + \phi_i - \phi_{i-1} \\ &= k + 0 - x \\ &= O(k) \text{ if } x = O(1)\end{aligned}$$

Thus, the amortized cost of reset is $O(k)$. The analysis of increment and decrement are similar to the previous section and thus, $O(1)$ amortized and $O(k)$ amortized, respectively.