Amortized Analysis Part-I (DAA, M.Tech + Ph.D.)

By:

Sunil Kumar Singh, PhD
Assistant Professor,
Department of Computer Science and Information Technology

School of Computational Sciences, Information and Communication Technology,
Mahatma Gandhi Central University, Motihari
Bihar, India-845401

Outline

- Amortized Analysis
- An Example
- Types of Amortized Analysis
- Aggregate Analysis
- The Accounting Method
- The Potential Method
- Conclusion
- References

Amortized Analysis

- In an amortized analysis, the time required to perform a sequence of data-structure operations is averaged over all the operations performed. Amortized analysis can be used to show that the average cost of an operation is small, if one average over a sequence of operations, even though a single operation withing the sequence might be expensive.

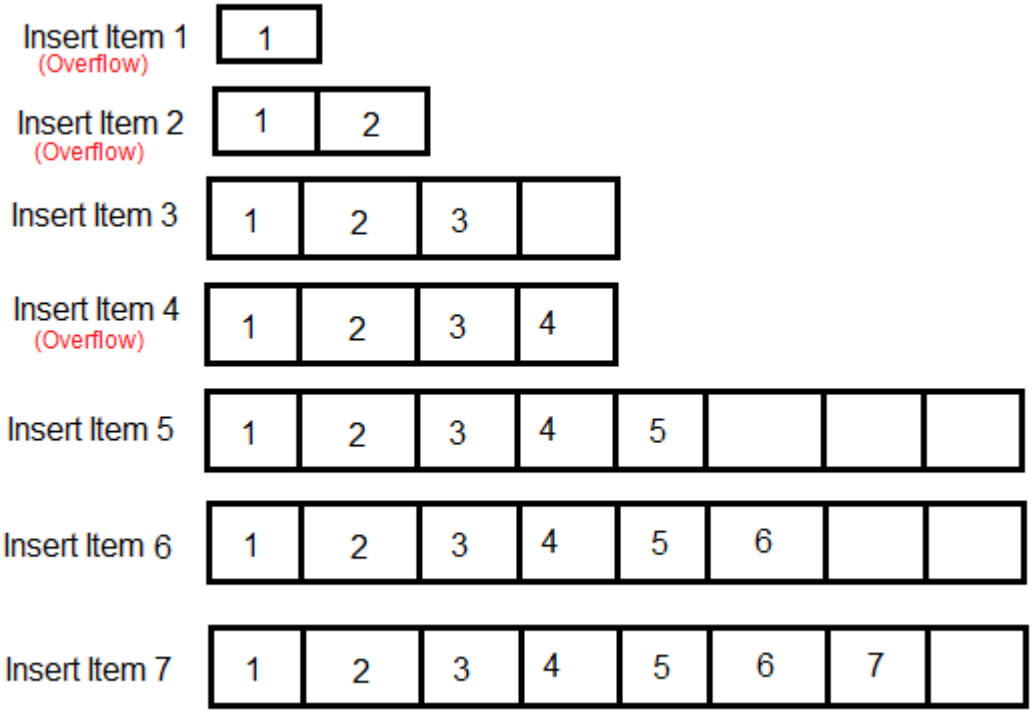
- Amortized analysis differs from average-case analysis in that probability is not involved; an amortized analysis guarantees the average performance of each operation in the worst case.
- **Example:** simple hash table insertion, the idea is to increase size of table whenever it becomes full.

1. Allocate memory for a larger table of size, typically twice the old table.
2. Copy the contents of old table to new table.
3. Free the old table

Amortized Analysis cont..

If the table has available space then simply insert the new item.

Initially table is empty and size is 0



Next overflow would happen when we insert 9, table size would become 16

Item No.	1	2	3	4	5	6	7	8	9	10	.....
Table Size	1	2	4	4	8	8	8	8	16	16	.....
Cost	1	2	3	1	5	1	1	1	9	1	.....

$$\text{Amortized Cost} = \frac{(1 + 2 + 3 + 5 + 1 + 1 + 9 + 1...)}{n}$$

We can simplify the above series by breaking terms 2, 3, 5, 9.. into two as (1+1), (1+2), (1+4), (1+8)

$$\text{Amortized Cost} = \frac{[\overbrace{(1 + 1 + 1 + 1...)}^{n \text{ terms}} + \overbrace{(1 + 2 + 4 + ...)}^{\lfloor \text{Log}_2(n-1) \rfloor + 1 \text{ terms}}]}{n}$$

$$<= \frac{[n + 2n]}{n}$$

$$<= 3$$

$$\text{Amortized Cost} = O(1)$$

Amortized Analysis

- Basically there are three types of Amortized Analysis:
  1. Aggregate Analysis
  2. The Accounting Method
  3. The Potential Method
- **Aggregate Analysis**

In aggregate analysis, we show that for all n, a sequence of n operations takes worst-case time $T(n)$ in total.

In the worst case, the average cost, or amortized cost, per operation is therefore T(n)/n.

Note that this amortized cost applies to each operation, even when there are several types of operations in the sequence.

Aggregate Analysis cont..

- **Ex-1 Stack Operations**

A. PUSH(S,x)

B. POP(S)

C. MULTIPOP(S,k)

```
1.      while not STACK-EMPTY(S) and k≠0
2.              do POP(S)
3.                      k=k-1
```

When we analyze the running time of a sequence on n PUSH, POP, and MULTIPOP operations on an initially empty stack.

Using aggregate analysis, any sequence of n operations takes a total of $O(n)$ time. The average cost of an operation is $O(n)/n=O(1)$

Amortized cost of each operation is to be the Average cost.

Aggregate Analysis cont..

- **Ex-2 Incrementing a binary counter**

As another example of aggregate analysis, consider the problem of implementing a k-bit binary counter that counts upward from 0. We use an array A[0..k-1] of bits, where length[A], as the counter. A binary number x that is stored in the counter has its lowest-order bin in A[0] and its highest-order bit in A[k-1], so that

$$x = \sum_{i=0}^{k-1} A[i].2^i$$

Initially, x=0, and thus A[i]=0

for i=0,1,.....k-1. to add 1

(modulo 2^k) to the value in

the counter, we use the following procedure.

```
INCREMENT(A)
1  i ← 0
2  while i < length[A] and A[i] = 1
3       do A[i] ← 0
4          i ← i + 1
5  if i < length[A]
6     then A[i] ← 1
```

Aggregate Analysis cont..

- The cost of each INCREMENT operation is linear in the number of bits flipped
- A single execution of INCREMENT takes time $O(k)$ in the worst case, in which array A contains all 1's
- Thus a sequence of n INCREMENT operations on a initially zero counter takes time $O(nk)$ in the worst case.

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

Aggregate Analysis cont..

- We can tighten our analysis to yield a worst-case cost of $O(n)$ for a sequence of n INCREMENT's by observing that not all bits flip each time INCREMENT is called. As figure shows A[0] does flip each time INCREMENT is called.
- The next-highest-order bit, A[1], flips only every other time: a sequence of n INCREMENT operations on an initially zero counter causes A[1] to flip $\left\lfloor \frac{n}{2} \right\rfloor$ times.
- Similarly, bit A[2] flips only every fourth time, or $\left\lfloor \frac{n}{4} \right\rfloor$ times in a sequence of n INCREMENTS's. In general, for $i = 0,1,2 \ldots . \left\lfloor \frac{\log n}{2} \right\rfloor$, bit A[i] flips $\left\lfloor \frac{n}{2^i} \right\rfloor$ times in a sequence of n INCREMENT operations on an initially zero counter.

Aggregate Analysis cont..

- For $i > \lfloor \log n \rfloor$, bit $A[i]$ never flips at all. The total number of flips in the sequence is thus

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

- 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 $O(n)/n = O(1)$.

Accounting Method

- In the accounting method of amortized analysis, we assign differing charges to different operations, with some operations charged more or less than they actually cost.
- Amount charged on an operations is called its amortized cost.
- When an operation's amortized cost exceeds its actual cost, the difference is assigned to specific objects in the data structure as credit.
- Credit can be used later on to help pay for operations whose amortized cost is less than their actual cost. Thus, one can view the amortized cost of an operations as being split between its actual cost and credit that is either deposited or used up.

This method is very different from aggregate analysis, in which all operations have the same amortized cost.

Accounting Method

- One must choose the amortized costs of operations carefully. If we want analysis with amortized costs to show that in the worst case the average cost per operation is small, the total amortized cost of a sequence of operations must be an upper bound on the total actual cost of the sequence.
- Moreover, as in aggregate analysis, this relationship must hold for all sequences of operations. If we denote the actual cost of the ith operation by c_i and the amortized cost of the ith operation, by $\widehat{c_i}$ we require.

$$\sum_{i=1}^{n} \widehat{c_i} \geq \sum_{i=1}^{n} c_i$$

For all sequences of n operations. The total credit stored in the data structure is the difference between the total amortized cost and the actual cost.

Accounting Method cont..

$$\sum_{i=1}^{n} \hat{c_i} - \sum_{i=1}^{n} c_i$$

- By inequality the total credit associated with the data structure must be non-negative at all times. If the total credit were ever allowed to become negative, then the amortized costs incurred at that time would be below the total actual costs incurred; for the sequence of operations up to that time, the total amortized cost would not be an upper bound on the total actual cost.

- Thus we must take care that the total credit in the data structure never becomes negative.

Accounting Method cont..

Example: Incrementing a binary counter

- As an illustration of the accounting method, we analyze the INCREMENT operation on a binary counter that starts at zero. As we observed earlier, the running time of this operation is proportional to the number of bits flipped, which we shall use as our cost for this example.
- For the amortized analysis, let us charge an amortized cost of 2 dollars to set a bit to 1.
- When a bit is set, we use 1 dollar to pay to flip the bit back to 0.
- The amortized cost of INCREMENT can now be determined. The cost of resetting the bits within the **while** loop is paid by the dollars on the bits that are reset.

Accounting Method cont..

Example: Incrementing a binary counter

- The number of 1's in the counter is never negative, and thus the amount of credit is always non-negative. Thus, for n INCREMENT operations, the total amortized cost is O(n), which bounds the total actual cost.

References

1. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.
2. Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. "Introduction to algorithms second edition." *The Knuth-Morris-Pratt Algorithm, year* (2001).
3. Seaver, Nick. "Knowing algorithms." (2014): 1441587647177.

Thank You