CENG 218
Design and Analysis of Algorithms

Izmir Institute of Technology

Lecture 14: Amortized Analysis

Amortized Analysis

- In ***amortized analysis***, we average the time required to perform a sequence of operations in an algorithm.
- We can show that the average cost of an operation is small, even though the operation for some element in the sequence might be expensive.
- It is different from average-case asymptotic analysis. In amortized analysis, probability is not involved and it guarantees the *average performance of each operation in the worst case*.

Aggregate Analysis

Aggregate analysis is one way of amortized analysis:

- We show that for all n, a sequence of n operations takes worst-case time $T(n)$ in total.
- Then the average cost, i.e. *amortized cost*, per operation becomes $T(n)/n$.

E.g.1: MULTIPOP(S,k) popping k elements from stack S. Apply a sequence of n operations, k is random in [1..n].

```
MULTIPOP(S, k)
1  while not STACK-EMPTY(S) and k > 0
2      POP(S)
3      k = k − 1
```

Aggregate Analysis

```
MULTIPOP(S, k)
1  while not STACK-EMPTY(S) and k > 0
2      POP(S)
3      k = k − 1
```

- In the worst-case, one MULTIPOP operation is $O(n)$ since there are n elements and they all can be popped.
- Hence a sequence of n operations cost $O(n^2)$, making average cost per operation $O(n)$.
- But, this bound is not tight!

Aggregate Analysis

```
MULTIPOP(S, k)
1  while not STACK-EMPTY(S) and k > 0
2      POP(S)
3      k = k − 1
```

In amortized analysis, we see that POP operation inside MULTIPOP can be called n times at most. Therefore, $O(n)/n=O(1)$ average cost. Much less than $O(n)$!

Note: You can think like MULTIPOP(S,n) called once:
$= O(n)/n=O(1)$,
or MULTIPOP(S,1) called n times:
$= n \cdot O(1)/n=O(1)$

Aggregate Analysis

E.g.2: Incrementing binary counter. A k-digit number, increment n times.
Cost: Number of changes in digits.

With asymptotic analysis, in worst case it takes $O(n \cdot k)$

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

However, with amortized analysis we see that $A[0]$ is flipped each time, $A[1]$ is flipped only every other time and goes on..

Total number of digit increments up to n is $2n$. That makes amortized average cost per operation $O(2n)/n = O(n)/n = O(1)$.

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

Accounting method

Accounting method is the second method of amortized analysis, where we assign different charges to different operations.

We charge an operation by *amortized cost.*

amortized cost - actual cost = credit

We spend the credit to pay for operations whose cost is higher than the amortized.

For the stack example lets assign amortized costs:

PUSH=2, POP=0, MULTIPOP=0

When we PUSH, we charge 2 units. But we gain a credit for a POP. So just by charging PUSH we have average upper bound.

Potential method

There is a also a third method, called Potential method.
Credit is represented in another way called ‘potential’.
We will not go into details in our course.

Section 17.1 in the book gives the aggregate method.

Section 17.2 and 17.3 gives the accounting method and the potential method respectively, but you are not responsible for them.