Amortized Cost Analysis

Example 1. Clearable Table

Consider a data structure "Clearable Table". It has two operations : add and clear.

Actual cost of operations

C(add) = 1

C(clear) = k, where k denotes the number of items present in the "Clearable Table".

Our goal is to determine the average cost of a sequence of n operations.

Sample Instance 1:

add, add, add, clear, add, add, clear, add, add, add, add, add, clear, add, add, clear.

1 1 1 3 1 1 2 1 1 1 1 1 5 1 1 2

Total cost = 24
Number of operations = 16
Average cost per operation = 24/16 ≤ 2.

Sample Instance 2:

add, clear, add, clear, add, clear, add, clear, add, clear, add, clear, add, clear.

1 1 1 1 1 1 1 1 1 1 1 1 1 1

Total cost = 14
Number of operations = 14
Average cost per operation = 14/14 = 1 ≤ 2.

Sample Instance 3:

add, add, add, add, add, add, add, add, add, add, add, clear

1 1 1 1 1 1 1 1 1 1 1 11

Total cost = 22
Number of operations = 12
Average cost per operation = 22/12 ≤ 2.

It seems **the average cost of an operation is constant time!** Can we show mathematically?

First Try

Traditional worst-case analysis

Each add costs **1** in the worst-case.

Each clear costs **n** in the worst-case (where n is the size of the Clearable Table.

Now consider a sequence of n operations.

All those operations can be clear. Therefore, total cost = n*n.

The number of operations = n.

The average cost = Total cost / The number of operations =n*n /n = n.

This is certainly wrong. We cannot use traditional worst-case analysis!

Second Try

Observe that any item you add to the table needs to be removed during the next clear operation. Therefore, **can we distribute the cost of clear among all previous adds?**

Sample Instance 1 (revisited)

add, add, add, clear, add, add, clear, add, add, add, add, add, clear, add, add, clear.

1 1 1 3 1 1 2 1 1 1 1 1 5 1 1 2

2 2 2 0 2 2 0 2 2 2 2 2 0 2 2 0

Total cost = 24
Number of operations = 16
Average cost per operation = 24/16 ≤ 2.

Sample Instance 2:

add, clear, add, clear, add, clear, add, clear, add, clear, add, clear, add, clear.

1 1 1 1 1 1 1 1 1 1 1 1 1 1

2 0 2 0 2 0 2 0 2 0 2 0 2 0

Total cost = 14
Number of operations = 14
Average cost per operation = 14/14 = 1 ≤ 2.

Sample Instance 3:

add, add, add, add, add, add, add, add, add, add, add, clear

2 2 2 2 2 2 2 2 2 2 2 0

Total cost = 22
Number of operations = 12
Average cost per operation = 22/12 ≤ 2.

Amortized_Cost (add) = 2

Amortized_Cost (clear) = 0

Now consider a sequence of n operations.

All those operations can be add (the most costly operation).

Therefore, total amortized cost = 2n.

The number of operations = n.

The average cost = Total amortized cost / The number of operations =2n /n = 2.

The average cost of an operation is 2.

The average cost of an operation is constant time!

EXAMPLE 2. ARRAYLIST WITH SIZE DOUBLING STRATEGY

Consider a data structure "ArrayList with size doubling strategy". It has two operations : add and resize.

Actual cost of operations

C(add) = 1

C(resize) = 3k, where k denotes the present size of the ArrayList.

(It cost 2k to create a new array of size 2k. Then it costs k to copy current array content to the newly created array. Thus total cost is 2k + k = 3k.)

Our goal is to determine the average cost of a sequence of n operations.

Sample Instance 1:

A resize just happened from size 4 to size 8.

That means, you have 4 free spaces in your newly created ArrayList.

add add add add rezise

1 1 1 1 3*size = 3*8 = 24.

Total cost = 4 + 24 = 28.

As in the previous example, we want to distribute the total cost of 28 among previous 4 adds.

Amortized_Cost(add) = 28/4 = 7.

Amortized_Cost(resize) = 0.

Sample Instance 2:

A resize just happened from size 8 to size 16.

That means, you have 8 free spaces in your newly created ArrayList.

add add add add add add add add rezise

1 1 1 1 1 1 1 1 3*size = 3*16 = 48.

Total cost = 8 + 48 = 56.

As in the previous example, we want to distribute the total cost of 56 among previous 8 adds.

Amortized_Cost(add) = 56/8 = 7.

Amortized_Cost(resize) = 0.

Amortized_Cost (add) = 7

Amortized_Cost (resize) = 0

Now consider a sequence of n operations.

All those operations can be add (the most costly operation).

Therefore, total amortized cost = 7n.

The number of operations = n.

The average cost = Total amortized cost / The number of operations =7n /n = 7.

The average cost of an operation is 7.

The average cost of an operation is constant time!

VERY IMPORTANT: NEVER TRY AMORTIZIZING WITH 1 ITEM (Or just considering the very first resize.)

1 add

Resize 3

Amortized cost = (1+3)/1 = 4. That is wrong.

Example 3. Resizing Using Fixed Increments.

A resize just happened and let it is size = K and also assume that the fixed size increment is d.

Now there are d empty slots.

Therefore, we can add d items and then we need to resize again.

Cost of d adds = d.

Cost of resize = K + d to create a new array of size K + d

PLUS

copy K items from the old array to new array.

= K + d + K = 2K + d.

Thus the total cost = 2K + d + d = 2K + 2d

Amortized cost of resize = (2K + 2d)/d = 2(K/d) + 2.

Note that (K/d) is not a constant as in the previous cases. It is unbounded! Therefore, you cannot show the average cost of an operation is constant time.