

# **CSE 548: Analysis of Algorithms**

## **Lecture 12 ( Amortized Analysis )**

**Rezaul A. Chowdhury**

**Department of Computer Science**

**SUNY Stony Brook**

**Spring 2014**

# A Binary Counter

counter value	counter	#bit flips	#bit resets (1 → 0)	#bit sets (0 → 1)
0	0 0 0 0 0 0 0 0			
1	0 0 0 0 0 0 0 1	1	0	1
2	0 0 0 0 0 0 1 0	2	1	1
3	0 0 0 0 0 0 1 1	1	0	1
4	0 0 0 0 0 1 0 0	3	2	1
5	0 0 0 0 0 1 0 1	1	0	1
6	0 0 0 0 0 1 1 0	2	1	1
7	0 0 0 0 0 1 1 1	1	0	1
8	0 0 0 0 1 0 0 0	4	3	1
9	0 0 0 0 1 0 0 1	1	0	1
10	0 0 0 0 1 0 1 0	2	1	1
11	0 0 0 0 1 0 1 1	1	0	1
12	0 0 0 0 1 1 0 0	3	2	1
13	0 0 0 0 1 1 0 1	1	0	1
14	0 0 0 0 1 1 1 0	2	1	1
15	0 0 0 0 1 1 1 1	1	0	1
16	0 0 0 1 0 0 0 0	5	4	1

# A Binary Counter

Consider a  $k$ -bit counter initialized to 0 ( i.e., all bits are 0's ).

Suppose we increment the counter  $n$  times.

and cost of an increment = #bits flipped

**Question:** What is the worst-case total cost of  $n$  increments?

**Worst-case cost of a single increment:**

#bit sets (  $0 \rightarrow 1$  ),  $b_1 \leq 1$

#bit resets (  $1 \rightarrow 0$  ),  $b_0 \leq k - b_1$

#bit flips  $= b_1 + b_0 \leq k$

**Worst-case cost of  $n$  increments:**

#bit flips  $\leq nk$

This turns out to be a very loose upper bound!

# Aggregate Analysis

A better upper bound can be obtained as follows.

Each increment sets  $(0 \rightarrow 1)$  at most one bit, i.e.,  $b_1 \leq 1$

So, total number of bits set by  $n$  increments,  $B_1 = b_1 n \leq n$

Since at most  $n$  bits are set, there cannot be more than  $n$  bit resets  $(1 \rightarrow 0)$ , i.e.,  $B_0 \leq B_1 \leq n$

So, total number of bit flips  $= B_1 + B_0 \leq n + n = 2n$

Thus worst-case cost of a sequence of  $n$  increments,  $T(n) \leq 2n$

Hence, in the worst case, average cost of an increment  $= \frac{T(n)}{n} \leq 2$

This *worst-case average cost* is called the *amortized cost* of an increment in a sequence of  $n$  increments.

# A Binary Counter

counter value	counter	#bit flips	#bit resets ( 1 → 0 )	#bit sets ( 0 → 1 )	total #bit flips
0	0 0 0 0 0 0 0 0				
1	0 0 0 0 0 0 0 1	1	0	1	1
2	0 0 0 0 0 0 1 0	2	1	1	3
3	0 0 0 0 0 0 1 1	1	0	1	4
4	0 0 0 0 0 1 0 0	3	2	1	7
5	0 0 0 0 0 1 0 1	1	0	1	8
6	0 0 0 0 0 1 1 0	2	1	1	10
7	0 0 0 0 0 1 1 1	1	0	1	11
8	0 0 0 0 1 0 0 0	4	3	1	15
9	0 0 0 0 1 0 0 1	1	0	1	16
10	0 0 0 0 1 0 1 0	2	1	1	18
11	0 0 0 0 1 0 1 1	1	0	1	19
12	0 0 0 0 1 1 0 0	3	2	1	22
13	0 0 0 0 1 1 0 1	1	0	1	23
14	0 0 0 0 1 1 1 0	2	1	1	25
15	0 0 0 0 1 1 1 1	1	0	1	26
16	0 0 0 1 0 0 0 0	5	4	1	31

# Amortized Analysis

- often obtains a tighter worst-case upper bound on the cost of a sequence of operations on a data structure by reasoning about the interactions among those operations
- the actual cost of any given operation may be very high, but that operation may change the state of the data structure in such a way that similar high-cost operations cannot appear for a while
- tries to show that there must be enough low-cost operations in the sequence to average out the impact of high-cost operations
- unlike average case analysis proves a worst-case upper bound on the total cost of the sequence of operations
- unlike expected case analysis no probabilities are involved

# Accounting Method ( Banker's View )

Consider a  $k$ -bit counter initialized to 0 ( i.e., all bits are 0's ).

## **Worst-case cost of a single increment:**

$$\text{\#bit sets ( } 0 \rightarrow 1 \text{ ), } b_1 \leq 1$$

$$\text{\#bit resets ( } 1 \rightarrow 0 \text{ ), } b_0 \leq k - b_1$$

$$\text{\#bit flips} = b_1 + b_0 \leq k$$

Thus each increment is paying for the bit it sets ( fair ).

But also paying for resetting bits set by prior increments ( unfair )!

## **A fairer cost accounting for each increment:**

**(1)** Pay for the bit it sets.

**(2)** Pay in advance for resetting this bit ( by some other increment ) in the future. Store this advanced payment as a *credit* associated with that bit position.

**(3)** When resetting a bit use the credit stored in that bit position.

# Accounting Method ( Banker's View )

counter value	counter	actual cost ( $c_i$ )	amortized cost ( $\hat{c}_i$ )		$\sum c_i$	$\leq$	$\sum \hat{c}_i$
0	0 0 0 0 0 0 0 0						
1	0 0 0 0 0 0 0 1	1	2 (overcharged)	1 coin	1	$\leq$	2
2	0 0 0 0 0 0 1 0	2	2	2 coins	3	$\leq$	4
3	0 0 0 0 0 0 1 1	1	2 (overcharged)	3 coins	4	$\leq$	6
4	0 0 0 0 0 1 0 0	3	2 (undercharged)	4 coins	7	$\leq$	8
5	0 0 0 0 0 1 0 1	1	2 (overcharged)	5 coins	8	$\leq$	10
6	0 0 0 0 0 1 1 0	2	2	6 coins	10	$\leq$	12
7	0 0 0 0 0 1 1 1	1	2 (overcharged)	7 coins	11	$\leq$	14
8	0 0 0 0 1 0 0 0	4	2 (undercharged)	8 coins	15	$\leq$	16
9	0 0 0 0 1 0 0 1	1	2 (overcharged)	9 coins	16	$\leq$	18



# Accounting Method ( Banker's View )

counter value	counter	actual cost ( $c_i$ )	amortized cost ( $\hat{c}_i$ )		$\sum c_i \leq \sum \hat{c}_i$
0	0 0 0 0 0 0 0 0				
1	0 0 0 0 0 0 0 1	1	2 (overcharged)	1	$\leq 2$
2	0 0 0 0 0 0 1 0	2	2	3	$\leq 4$
3	0 0 0 0 0 0 1 1	1	2 (overcharged)	4	$\leq 6$
4	0 0 0 0 0 1 0 0	3	2 (undercharged)	7	$\leq 8$

Total credits remaining after  $n$  increments,  $\Delta_n = \sum_{i=1}^n \hat{c}_i - \sum_{i=1}^n c_i$

We must make sure that for all  $n$ ,  $\Delta_n \geq 0$

$$\Rightarrow \sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$$

This will ensure that *the total amortized cost is always an upper bound on the total actual cost.*

# Potential Method ( Physicist's View )

**Banker's View:** Store prepaid work as credit with specific objects in the data structure.

**Physicist's View:** Represent total remaining credit in the data structure as a single potential function.

**Suppose:** state of the initial data structure =  $D_0$

state of the data structure after the  $i$ -th operation =  $D_i$

potential associated with  $D_i$  is =  $\Phi(D_i)$

Then amortized cost of the  $i$ -th operation,

$$\begin{aligned}\hat{c}_i &= \text{actual cost} + \text{potential change due to that operation} \\ &= c_i + \Phi(D_i) - \Phi(D_{i-1})\end{aligned}$$

# Potential Method ( Physicist's View )

Then amortized cost of the  $i$ -th operation,

$$\begin{aligned}\hat{c}_i &= \text{actual cost} + \text{potential change due to that operation} \\ &= c_i + \Phi(D_i) - \Phi(D_{i-1})\end{aligned}$$

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) = \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0)$$

Since we do not know  $n$  in advance, if we make sure that for all  $n$ ,  $\Phi(D_n) \geq \Phi(D_0)$ , we ensure that always  $\sum_{i=1}^n \hat{c}_i \geq \sum_{i=1}^n c_i$ .

In other words, in that case, *the total amortized cost will always be an upper bound on the total actual cost.*

One way of achieving that is to find a  $\Phi$  such that  $\Phi(D_0) = 0$  and for all  $n$ ,  $\Phi(D_n) \geq 0$ .

# Potential Method ( Physicist's View )

For the binary counter,

$\Phi(D_i)$  = number of set bits ( i.e., 1 bits ) after the  $i$ -th operation

counter value	counter	actual cost ( $c_i$ )	$\Phi(D_i)$	amortized cost ( $\hat{c}_i$ )		$\sum c_i \leq \sum \hat{c}_i$
0	0 0 0 0 0 0 0 0		0			
1	0 0 0 0 0 0 0 1	1	1	2 (overcharged)	1	$\leq 2$
2	0 0 0 0 0 0 1 0	2	1	2	3	$\leq 4$
3	0 0 0 0 0 0 1 1	1	2	2 (overcharged)	4	$\leq 6$
4	0 0 0 0 0 1 0 0	3	1	2 (undercharged)	7	$\leq 8$
5	0 0 0 0 0 1 0 1	1	2	2 (overcharged)	8	$\leq 10$
6	0 0 0 0 0 1 1 0	2	2	2	10	$\leq 12$
7	0 0 0 0 0 1 1 1	1	3	2 (overcharged)	11	$\leq 14$
8	0 0 0 0 1 0 0 0	4	1	2 (undercharged)	15	$\leq 16$