



# Chapter 4 Amortized Analysis

Algorithm Theory WS 2019/20

**Fabian Kuhn** 

## **Amortization**



- Consider sequence  $o_1, o_2, ..., o_n$  of n operations (typically performed on some data structure D)
- $t_i$ : execution time of operation  $o_i$
- $T := t_1 + t_2 + \cdots + t_n$ : total execution time
- The execution time of a single operation might vary within a large range (e.g.,  $t_i \in [1, O(i)]$ )
- The worst case overall execution time might still be small
  - → average execution time per operation might be small in the worst case, even if single operations can be expensive

# Analysis of Algorithms



- Best case
- Worst case
- · Average case Lushally: random input
- Amortized worst case

What is the <u>average cost</u> of an operation in a <u>worst case sequence</u> of operations?

# Example 1: Augmented Stack



## **Stack Data Type: Operations**

- S.push(x) : inserts x on top of stack
- S.pop(): removes and returns top element

## **Complexity of Stack Operations**

• In all standard implementations: O(1)

## **Additional Operation**

- S.multipop(k): remove and return top k elements
- Complexity: O(k)
- What is the amortized complexity of these operations?

```
induitory: constant amortited cost

so can only delete items from S that were pushed to S
```

# Augmented Stack: Amortized Cost



#### **Amortized Cost**

- Sequence of operations i = 1, 2, 3, ..., n
- Actual cost of op. i: t<sub>i</sub>
- Amortized cost of op. i is  $a_i$  if for every possible seq. of op.,

$$T = \sum_{i=1}^{n} t_i \le \sum_{i=1}^{n} a_i$$

## **Actual Cost of Augmented Stack Operations**

- S. push(x), S. pop(): actual cost  $t_i = Q(1)$
- $S. \operatorname{multipop}(k)$  : actual cost  $t_i = O(k)$
- Amortized cost of all three operations is constant
  - The total number of "popped" elements cannot be more than the total number of "pushed" elements: cost for pop/multipop ≤ cost for push

# Augmented Stack: Amortized Cost



#### **Amortized Cost**

$$T = \sum_{i} t_i \le \sum_{i} a_i$$

## **Actual Cost of Augmented Stack Operations**

- S.push(x), S.pop(): actual cost  $t_i \le c$
- S. multipop(k) : actual cost  $t_i \leq c \cdot k$

N operations

$$P \le n$$
 push ops  $\longrightarrow$  total push cost  $\le C \cdot P$ 

total # deleted elem:  $\le P$   $\longrightarrow$  total pop/multipop cost  $\le C \cdot P$ 
 $\longrightarrow$  total cost  $\le 2CP$ 
 $QVS. cost parap. : \le \frac{2CP}{N} \le \frac{2CP}{P} = 2C$ 

# Example 2: Binary Counter



## Incrementing a binary counter: determine the bit flip cost:

Operation	Counter Value	Cost
	00000	
1	00001	1
2	00010	2
3	0001 <mark>1</mark>	1
4	00 <b>100</b>	3
5	0010 <mark>1</mark>	1
6	001 <b>10</b>	2
7	0011 <mark>1</mark> )	1
8	01000	4
9	0100 <mark>1</mark>	1
10	010 <b>10</b>	2
11	0101 <mark>1</mark>	1
12	01 <b>100</b>	3
13	0110 <mark>1</mark>	1

# Accounting Method



#### **Observation:**

Each increment flips exactly one <u>0</u> into a <u>1</u>

$$00\underline{100011111} \Rightarrow \underline{0010010000}$$

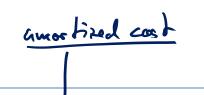
#### Idea:

- Have a bank account (with initial amount 0)
- Paying <u>x</u> to the bank account costs <u>x</u>
- Take "money" from account to pay for expensive operations

#### **Applied to binary counter:**

- Flip from <u>0 to 1</u>: pay <u>1</u> to bank account (cost: 2)
- Flip from 1 to 0: take 1 from bank account (cost: 0)
- Amount on bank account = number of ones
  - → We always have enough "money" to pay!

# Accounting Method





					<u> </u>	
Op.	Counter	Cost	To Bank	From Bank	Net Cost	Credit
	00000					0
1	00001	1	ļ	0	2	1
2	00010	2	(	1	2	1
3	00011	1	1	0	2	2
4	00100	3	l	2	2	1
5	00101	1	1	0	2	2
6	00110	2	1	(	2	2
7	00111	1	1	0	2	3
8	01000	4	1	3	2	1
9	01001	1	1	0	2	2
10	01010	2		با	, 2,	2
				<b>—</b>		

## Potential Function Method



- Most generic and elegant way to do amortized analysis!
  - But, also more abstract than the others...
- State of data structure / system:  $S \in \mathcal{S}$  (state space)

Potential function  $\underline{\Phi} : \underline{\mathcal{S}} \to \mathbb{R}_{\geq 0}$ 

#### Operation *i*:

- $t_i$ : actual cost of operation i
- $-\overline{S_i}$ : state after execution of operation i ( $S_0$ : initial state)
- $-\Phi_i := \Phi(S_i)$ : potential after exec. of operation i

$$a_i$$
: amortized cost of operation  $i$ :  $a_i \coloneqq t_i + \Phi_i - \Phi_{i-1}$ 

## Potential Function Method



#### Operation *i*:

$$t_i = \alpha_i + \phi_{i-1} - \phi_i$$

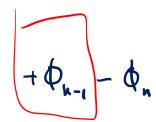
actual cost:  $t_i$  amortized cost:  $a_i = t_i + \Phi_i - \Phi_{i-1}$ 

#### **Overall cost:**

$$T \coloneqq \sum_{i=1}^{n} t_i = \left(\sum_{i=1}^{n} a_i\right) + \Phi_0 - \Phi_n \le \underbrace{\geq a_i}_{\geq 0} + \Phi_0$$

$$\begin{aligned}
& \leq \epsilon_i = \alpha_1 + \phi_0 - \phi_1 \\
& + \alpha_2 \\
& + \alpha_3
\end{aligned}$$

$$\begin{aligned}
& + \phi_1 - \phi_2 \\
& + \phi_2 - \phi_3
\end{aligned}$$



# Binary Counter: Potential Method



Potential function:

Φ: number of ones in current counter

- Clearly,  $\Phi_0 = 0$  and  $\Phi_i \ge 0$  for all  $i \ge 0$
- Actual cost t<sub>i</sub>:
  - 1 flip from 0 to 1
  - $t_i 1$  flips from 1 to 0
- Potential difference:  $\Phi_i \Phi_{i-1} = 1 (t_i 1) = 2 t_i$
- Amortized cost:  $a_i = t_i + \Phi_i \Phi_{i-1} = 2$