Chapter 4
Amortized Analysis

Algorithm Theory
WS 2018/19

Fabian Kuhn

Amortization

- Consider sequence $o_1, o_2, \dots, o_n$ of n operations (typically performed on some data structure D)

- $\boldsymbol{t_i}$: execution time of operation o_i
- $\boldsymbol{T \coloneqq t_1 + t_2 + \cdots + t_n}$: 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

- $\boxed{\text{Worst case}}$

- $\boxed{\text{Average case}}$ running time for a typical input

  random

- $\boxed{\text{Amortized worst case}}$

What is the average cost of an operation in a worst case sequence of operations?

Example 1: Augmented Stack

Stack Data Type: Operations

- $S.\text{push}(x)$: 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

- $\boldsymbol{S}$**.multipop($\boldsymbol{k}$)** : remove and return top k elements
- Complexity: $O(k)$

- What is the amortized complexity of these operations?

Augmented Stack: Amortized Cost

Amortized Cost

- Sequence of operations $i = 1, 2, 3, \dots, n$
- Actual cost of op. i: $\boldsymbol{t_i}$
- Amortized cost of op. i is $\boldsymbol{a_i}$ if for every possible seq. of op.,

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

Actual Cost of Augmented Stack Operations

- $S.\mathrm{push}(x), S.\mathrm{pop}()$: actual cost $t_i = O(1)$
- $S.\mathrm{multipop}(k)$: 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 $\leq$ 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

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

<u>n operations</u>

$p \le n$ push ops. total push cost $\le c \cdot p$

total # del. elem. $\le p$ total pop/multipop cost $\le c \cdot p$

total cost $\le 2 \cdot c \cdot p$

avg. cost per op. $\le \frac{2 \cdot c \cdot p}{n} \le \frac{2 \cdot c \cdot p}{p} = \underline{\underline{2c}}$

Example 2: Binary Counter

Incrementing a binary counter: determine the bit flip cost:

Operation	Counter Value	Cost
	00000	
1	0000**1**	1
2	000**10**	2
3	0001**1**	1
4	00**100**	3
5	0010**1**	1
6	001**10**	2
7	0011**1**	1
8	0**1000**	4
9	0100**1**	1
10	010**10**	2
11	0101**1**	1
12	01**100**	3
13	0110**1**	1

Accounting Method

Observation:

- Each increment flips exactly one 0 into a 1

$$00100\mathbf{0}1111 \Longrightarrow 00100\mathbf{1}0000$$

Idea:

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

Applied to binary counter:

- Flip from 0 to 1: pay 1 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

Op.	Counter	Cost	To Bank	From Bank	Net Cost	Credit
	0 0 0 0 0					0
1	0 0 0 0 **1**	1	1	0	2	1
2	0 0 0 **1** **0**	2	1	1	2	1
3	0 0 0 1 **1**	1	1	0	2	2
4	0 0 **1** **0** **0**	3	1	2	2	1
5	0 0 1 0 **1**	1	1	0	2	2
6	0 0 1 **1** **0**	2	1	1	2	2
7	0 0 1 1 **1**	1	1	0	2	3
8	0 **1** **0** **0** **0**	4	1	3	2	1
9	0 1 0 0 **1**	1	1	0	2	2
10	0 1 0 **1** **0**	2	1	1	2	2

$$C + \underbrace{B - F}_{\geq 0} = \underline{\underline{A}} \qquad \underline{\underline{\geq 0}}$$

$$\underline{C \leq A}$$