Algorithm Theory

Chapter 4
Amortized Analysis

Part I:
Basics & Accounting Method

Fabian Kuhn

Amortization

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

- $\boldsymbol{t_i}$: execution time of operation o_i
- $\boldsymbol{T := t_1 + t_2 + \cdots + t_n}$: 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
- Amortized worst case

The best case usually does not occur and is not really interesting.

The **standard** way of algorithm analysis.

Assume that the input is **random** according to some given distribution.

Average cost per operation in a **worst case sequence of operations**
- a form of worst-case analysis for sequences 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

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

What is the amortized complexity of these operations?

Intuitively: amortized cost per operation is constant

- We can only delete items from S that were previously pushed to S.
- The total time for deleting is not more than for pushing.

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

$$T = \sum_i t_i \leq \sum_i a_i$$

Actual Cost of Augmented Stack Operations

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

$\boldsymbol{n}$ **operations:** p push operations, the rest are pop and multipop op.

- $p \leq n$ push op. $\Longrightarrow$ total push cost $\leq c \cdot p$
- total #deleted elem. $\leq p$ $\Longrightarrow$ total pop/multipop cost $\leq c \cdot p$

  $\Longrightarrow$ total cost $\leq 2 \cdot c \cdot p$

- **Average cost per operation** $\boldsymbol{\leq \frac{2cp}{n} \leq \frac{2cp}{p} = 2c}$

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	00011	1
4	00100	3
5	00101	1
6	00110	2
7	00111	1
8	01000	4
9	01001	1
10	01010	2
11	01011	1
12	01100	3
13	01101	1

Accounting Method

Observation:

- Each increment flips exactly one 0 into a 1

$$0010001111 \Longrightarrow 0010010000$$

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

amortized cost

Op.	Counter	Cost	To Bank	From Bank	Net Cost	Balance
	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

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

$$\Longrightarrow A \geq C$$