Chapter 4
Amortized Analysis

Algorithm Theory
WS 2019/20

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

typical input

usually: random input

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

Example 1: Augmented Stack

Stack Data Type: Operations

- $S.\mathrm{push}(x)$: 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 : constant amortized cost

→ can only delete items from S that were pushed to S

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

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

n operations

$p \leq n$ push ops $\longrightarrow$ total push cost $\leq c \cdot p$

total # deleted elem: $\leq p$ $\longrightarrow$ total pop/multipop cost $\leq c \cdot p$

$\longrightarrow$ total cost $\leq 2cp$

avg. cost per op. : $\leq \frac{2cp}{n} \leq \frac{2cp}{p} = \underline{\underline{2c}}$

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

$$0010001111 \Longrightarrow 0010010000$$

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	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

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

$$C \leq A$$

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 $\boldsymbol{\Phi}: \boldsymbol{\mathcal{S}} \to \mathbb{R}_{\geq 0}$

- **Operation $\boldsymbol{i}$:**
  - $\boldsymbol{t_i}$: actual cost of operation i
  - $\boldsymbol{S_i}$: state after execution of operation i (S_0: initial state)
  - $\boldsymbol{\Phi_i} \coloneqq \Phi(S_i)$: potential after exec. of operation i
  - $\boldsymbol{a_i}$: amortized cost of operation i:

$$\boldsymbol{a_i \coloneqq t_i + \Phi_i - \Phi_{i-1}}$$

Potential Function Method

Operation i:

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

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

Overall cost:

$$T := \sum_{i=1}^{n} t_i = \left(\sum_{i}^{n} a_i\right) + \Phi_0 - \underbrace{\Phi_n}_{\geq 0} \leq \sum a_i + \phi_0$$

$$\begin{aligned}\sum t_i = a_1 &+ \phi_0 - \phi_1 \\ + a_2 &+ \phi_1 - \phi_2 \\ + a_3 &+ \phi_2 - \phi_3 \\ &\vdots \\ + a_n &+ \phi_{n-1} - \phi_n\end{aligned}$$

Binary Counter: Potential Method

- **Potential function:**

  $\mathbf{\Phi}$**: number of ones in current counter**

- Clearly, $\Phi_0 = 0$ and $\Phi_i \geq 0$ for all $i \geq 0$

- Actual cost t_i:
  - 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: $a_i = t_i + \Phi_i - \Phi_{i-1} = 2$