Chapter 4
Amortized Analysis

Algorithm Theory
WS 2018/19

Fabian Kuhn

Amortized Analysis

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

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.,

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

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!

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

binary counter

$\Phi(S) :=$ # 1's in the bit representation

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

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

Potential Function Method

Operation i:

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

Overall cost:

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

$$\begin{aligned}\sum_{i=1}^{n} a_i = \; & t_1 - \phi_0 + \phi_1 \\ & + t_2 - \phi_1 + \phi_2 \\ & + t_3 - \phi_2 + \phi_3 \\ & \vdots \\ & + t_{n-1} - \phi_{n-2} + \phi_{n-1} \\ & + t_n - \phi_{n-1} + \phi_n = \sum_{i=1}^{n} t_i - \phi_0 + \phi_n\end{aligned}$$

Binary Counter: Potential Method

- **Potential function:**

  Φ: 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: $\Phi_i - \Phi_{i-1} = 1 - (t_i - 1) = 2 - t_i$

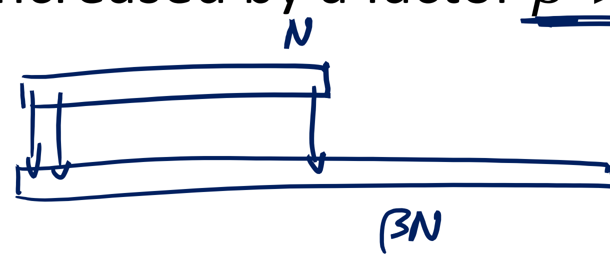
- Amortized cost: $a_i = t_i + \Phi_i - \Phi_{i-1} = 2$

Example 3: Dynamic Array

- How to create an array where the size dynamically adapts to the number of elements stored?
  - e.g., Java “ArrayList” or Python “list”

Implementation:

- Initialize with initial size N_0
- Assumptions: Array can only grow by appending new elements at the end
- If array is full, the size of the array is increased by a factor $\beta > 1$



Operations (array of size N):

- read / write: actual cost $O(1)$
- append: actual cost is $O(1)$ if array is not full, otherwise the append cost is $O(\beta \cdot N)$ (new array size)

Example 3: Dynamic Array

Notation:

- n: number of elements stored
- N: current size of array

Cost t_i of i^{th} append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

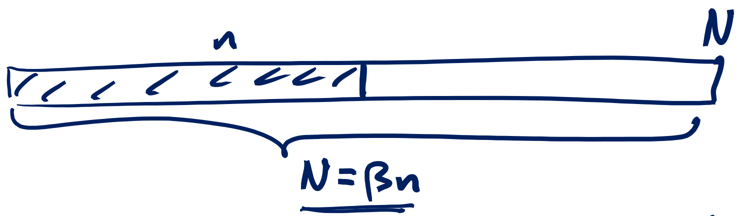
Claim: Amortized append cost is $O(1)$

Potential function Φ?

- should allow to pay expensive append operations by cheap ones
- when array is full, Φ has to be large
- immediately after increasing the size of the array, Φ should be small again

Dynamic Array: Potential Function

Cost t_i of i^{th} append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \underline{\beta \cdot N} & \text{if } n = N \end{cases}$



ϕ small ($\phi = 0$)

at the beginning

$N = N_0$

$n = 0$

ϕ large ($\phi \geq \beta N$)

$$\phi(n, N) = c \cdot (\beta n - N) + c N_0$$

$$c(\beta N - N) \geq \beta N$$

$$c(\beta - 1) \geq \beta$$

$$c \geq \frac{\beta}{\beta - 1}$$

$$\boxed{\phi(n, N) = \frac{\beta}{\beta - 1}(\beta n - N) + \frac{\beta}{\beta - 1} N_0}$$

Dynamic Array: Amortized Cost

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

Cost t_i of i^{th} append operation: $t_i = \begin{cases} 1 & \text{if } n < N \\ \beta \cdot N & \text{if } n = N \end{cases}$

$$\phi(n,N) = \frac{\beta}{\beta-1}\left(\beta n - N + N_0\right)$$

amortized cost a_i

Case 1 ($n<N$): $a_i = 1 + \frac{\beta^2}{\beta-1}(n+1-n) = 1 + \frac{\beta^2}{\beta-1}$

Case 2 ($n=N$):

$$a_i = \beta\cdot N + \left[\underbrace{\frac{\beta}{\beta-1}\left(\beta(N+1)-\beta N\right) - \frac{\beta}{\beta-1}\left(\beta N - N\right)}_{\frac{\beta^2}{\beta-1} - \underbrace{\frac{\beta}{\beta-1}(\beta-1)\cdot N}_{\beta N}}\right] = \frac{\beta^2}{\beta-1}$$

$$\boxed{\text{amortized cost: } \le 1 + \frac{\beta^2}{\beta-1}}$$