Algorithm Theory

06 – Amortized Analysis

Dr. Alexander Souza

Amortization

- Consider a sequence $a_1, a_2, \ldots, a_n$ of n operations performed on a data structure D
- T_i = execution time of a_i
- $T = T_1 + T_2 + \ldots + T_n$ total execution time
- The execution time of a single operation can vary within a large range, e.g. in $1,\ldots,n$, but the worst case does not occur for all operations of the sequence.
- Average execution time of an operation is small, even though a single operation can have a high execution time.

$$\frac{1}{n} \cdot \sum_{i=1}^{n} T_i$$

Goal: What is the average execution time of an operation in a sequence?

Analysis of algorithms

- Best case
- Worst case
- Average case
- Amortized worst case

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

Amortization

Idea:

- Pay more for inexpensive operations
- Use the credit to cover the cost of expensive operations

Three methods:

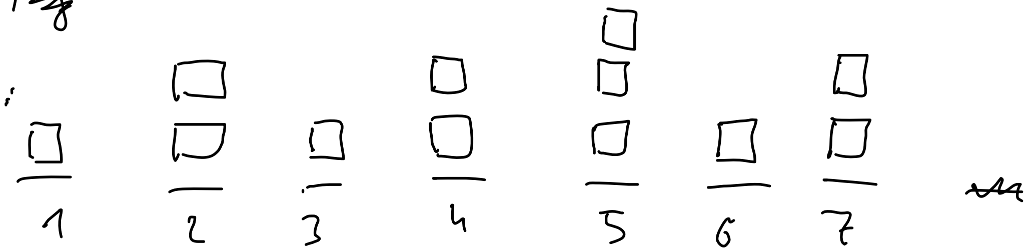
1. Aggregate method
2. Accounting method
3. Potential method

Amortization = Overcharging + Bookkeeping

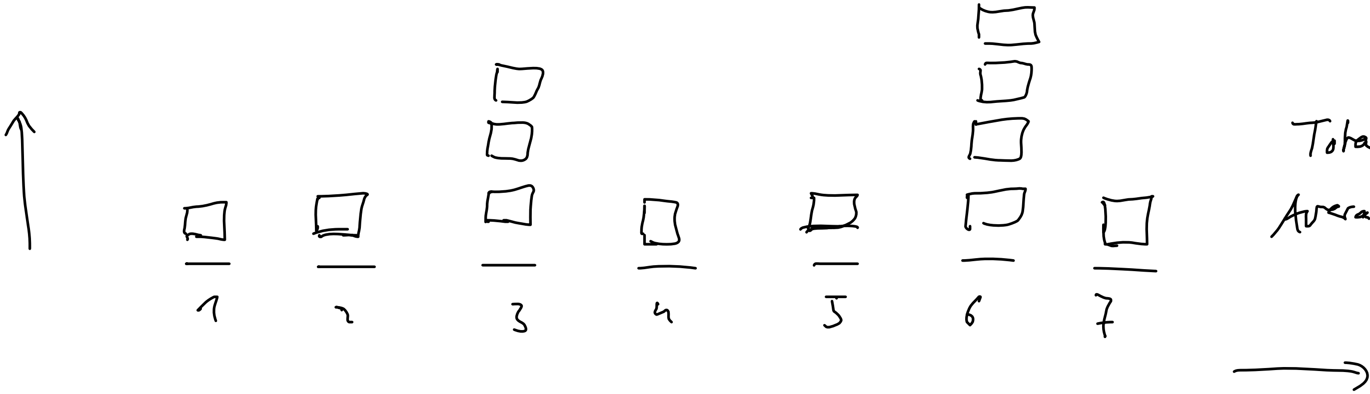
Want to show: Average cost ≤ 2 by paying 2 units per operation and bookkeeping

~~Pay~~

Credit:



Since credit never becomes negative the average cost per operation is at most two.



Total: 12

Average Cost: $\frac{12}{7} < 2$

1. Aggregate method: 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
		23

Average cost $\frac{23}{13} < 2$

on a sequence of 13 operations.

How is this shown in general?

In aggregate method one tries to estimate

$\frac{1}{n} \cdot \sum_{i=1}^{n} T_i$ directly.

Difficult to evaluate in general.

2. The accounting method

Observation:

In each increment exactly one 0 flips to 1.

```
  ..... 0 1 .... 1
        ↓
+       1 1 ... 1  1
  ----------------
  ..... 1 0     0 0
```

Idea:

Pay two cost units for flipping a 0 to a 1

→ each 1 has one cost unit deposited in the banking account (*)

If we can show that the account is never negative, we have shown $O(1)$ ~~am~~ average cost per operation.

(*) We do not pay for flipping a 1 to a 0.

The accounting method

\# bit flips

Operation	Counter value	Cost	Payment	Credit
	0 0 0 0 0			
1	0 0 0 0 1	1	2	1
2	0 0 0 1 0	2	0 + 2 = 2	0, 1
3	0 0 0 1 1	1	2	2
4	0 0 1 0 0	3	0 + 0 + 2 = 2	1, 0, 1
5	0 0 1 0 1	1	2	2
6	0 0 1 1 0	2	⋮	2
7	0 0 1 1 1	1		3
8	0 1 0 0 0	4		1
9	0 1 0 0 1	1		
10	0 1 0 1 0	2		

<u>Observation</u>: The credit is always equal to the number of 1's in the bit string.
This is because we always pay two units into the account when we flip a 1 to 0.

3. The potential method

Potential function ϕ

$\phi : D \longrightarrow \mathbb{R}$

Data structure $D \rightarrow \phi(D)$

t_i = actual cost of the i-th operation

ϕ_i = potential after execution of the i-th operation (= $\phi(D_i)$)

a_i = amortized cost of the i-th operation

Definition:

amortized cost

$$\underbrace{a_i}_{\text{amortized cost}} = \underbrace{t_i}_{\text{actual cost}} + \underbrace{\phi_i - \phi_{i-1}}_{\text{change in potential function}}$$

Example: binary counter

$\phi : D \longrightarrow \#\ 1\text{'s in } D$

D_i = counter value after the i-th operation
$\phi_i = \phi(D_i)$ = # of 1's in D_i

i–th operation	# of 1's
$\rightarrow$ D_{i-1}: .....0/1.....01.....1 (b_i 1's)	$B_{i-1} = \phi_{i-1}$
+1	b_i 1's are destroyed 1 1 is created
$\rightarrow$ D_i : .....0/1.....$\overbrace{10.....0}^{b_i}$	$B_i = B_{i-1} - b_i + 1 = \phi_i$

t_i = **actual bit flip cost of operation *i***
= b_i+1

What is a_i ?

$a_i = t_i + \phi_i - \phi_{i-1} = b_i + 1 + \cancel{B_{i-1}} - b_i + 1 - \cancel{B_{i-1}}$
$= 2.$

Binary counter

t_i = actual bit flip cost of operation i
a_i = amortized bit flip cost of operation i

$$a_i = (b_i + 1) + (B_{i-1} - b_i + 1) - B_{i-1}$$
$$= 2$$
$$\Rightarrow \sum t_i \le 2n \quad \Longrightarrow \quad \frac{1}{n} \cdot \sum_{i=1}^{n} t_i \le 2$$

$$\sum_{i=1}^{n} t_i = \sum_{i=1}^{n} (a_i + \phi_{i-1} - \phi_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} \underbrace{(\phi_{i-1} - \phi_i)}_{\text{telescoping series}} =$$
$$= \sum_{i=1}^{n} a_i + \underbrace{\phi_0}_{=0} - \underbrace{\phi_n}_{\ge 0} \Longrightarrow \sum_{i=1}^{n} t_i \le \sum_{i=1}^{n} a_i$$
$$a_i = t_i + \phi_i - \phi_{i-1} \Rightarrow t_i = a_i - \phi_i + \phi_{i-1}$$

Dynamic tables

Problem:

Maintain a table supporting the operations insert and delete such that

- the table size can be adjusted dynamically to the number of items
- the used space in the table is always at least a constant fraction of the total space
- the total cost of a sequence of n operations (insert or delete) is $O(n)$.

Applications: hash table, heap, stack, etc.

Load factor α_T: number of items stored in the table divided by the size of the table