Amortized analysis

Study a sequence of operations with individual costs (that may vary)
We want to compute the average cost of an operation (over the entire sequence)

amortize = "spread cost over time"

→ accounting

Real-life example: WASHING MACHINE.

Cost of washing: 2€ (water, detergent, electricity)

Daily expenses:

2€
2€
2€
..
2€
2€
500€ (machine broke, had to buy a new one)
2€
2€
2€

Want to spread cost to get a better sense of daily cost:

Approach 1.

Whenever you wash (2€), put 1€ into a piggy-bank.

If machine replaced after every ~500 washes, we are fine.
(Have some initial deposit to cover variation.)

=> amortized cost of washing: 3€ / day
4

Approach 2.

Redefine cost as
"current, actual cost" + "decrease in value of assets" → washing machine

normal wash:
2€ + ~1€ ← decrease in value

replacing the machine and washing:
500€ + 2€ + (- 500€) + 1€ = 3€

In data structures:

sequence of operations op1, op2, ..., opm
m

cost is usually running time.

approach 1. For the purpose of analysis, pretend we can deposit time and use later.

approach 2. Define a suitable potential function.

Example 1. Binary counter

MSB LSB n digits

b_n	b_{n-1}	$\dots$	b_3	b_2	b_1	
0	0		0	0	0	
						cost 1
0	0		0	0	1	
						cost 2
0	0		0	1	0	
						cost 1
0	0		0	1	1	
						cost 3
0	0		1	0	0	
						...
1	1		1	1	1	

Operation: increment by 1

Assumption: Flipping one digit takes unit time

Some operations costlier than others

Claim: amortized time of operation <= 2

digits
0111111
10000000

Method 1. (Bank account method)

Put 1€ on each 1-digit.

Initially no money needed.

A single increment operation:

- one flip 0->1
- t flips 1->0 (actual cost t+1)

operation removes t €, these can pay for the t flips 1->0.

We just need to pay for the single 0->1 (1€)

AND we need to deposit 1€ on the new 1-digit (1€)

} 2 €

Total cost: 2€

(Intuition: we charged cost of some operations to earlier, cheaper operations)

€ € €
0 0 1 1 0 0 1
€ € €
0 0 1 1 0 1 0

t
€ € € € €
0 1 1 1 1
€
1 0 0 0 0

Assume 1€ = 1 digit flip

Similar for Array-based stack

Method 2. (Direct counting)

$b_n \dots b_2\, b_1$
0 0 0 . 0
- - -
1 1 1 1 1 1 1

b_1 flipped 2^n times

b_2 flipped 2^{n-1} times

b_k flipped 2^{n-k+1} times

b_n flipped once

Total number of digit-flips: $1+2+4+\cdots+2^n = 2^{n+1}-1$

incr. operations 2^n

Averaged over operations, amortized cost < 2.

Method 3. potential-functions

recall: m operations op1, ..., opm

Define:

Φ : function that describes the state of the system (data structure)

$cost_i$: actual ("real") cost of i-th operation

Φ_i : potential before i-th operation

Total cost $= \sum_{i=1}^{m} cost_i$

increase

Change in potential due to i-th operation: $\overbrace{\phi_{i+1} - \phi_i}^{\Delta \phi}$

amortized_cost$_i \overset{\text{def}}{=} cost_i + \underbrace{\phi_{i+1} - \phi_i}_{\text{incr. in potential}}$

$cost_i$: actual cost

$$\sum_{i=1}^{m} \text{amortized\_cost}_i = \sum_{i=1}^{m} \text{cost}_i + \overbrace{\phi_{final} - \phi_0}^{\text{telescoping sum}}$$

$$\sum \text{cost}_i = \sum \text{amortized\_cost}_i + \phi_0 - \phi_{final} \leq \sum \text{amortized\_cost}_i$$

(Define ϕ s.t. $\phi_0 = 0$

$\phi \geq 0$, $\phi_{final} \geq 0$)

Question:

How to define suitable ϕ ?

Intuition: ϕ measure "badness" system

initial $\phi_0 = 0$ "good"

costly operations should reduce ϕ (improve system)

Example: binary counter

$\phi = \#$ 1-digits in counter

$000\overbrace{111111}^{t}$

0010000000

$cost_i = t+1$

$\Delta\phi = -(t-1)$

amortised_cost$_i \leq t+1-(t-1) = 2$

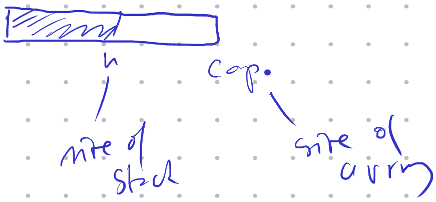
$\phi_0 = 0$

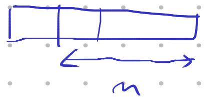
$\phi \geq 0$

$\sum$ cost $\leq \sum$ am. cost

avg cost ≤ 2

Example: Stack implemented w. array





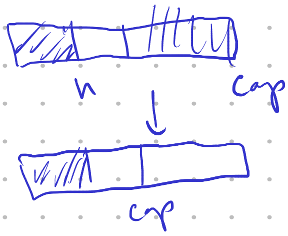
$\phi = |2n - cap|$ $\qquad \phi_0 = 3$

$\phi \geq 0$

$\text{amortised\_cost}_i = \text{cost}_i + \phi_{i+1} - \phi_i$

simple pop/push op: $\left.\begin{array}{l} \text{cost}_i = 1 \\ \Delta\phi \leq 2 \end{array}\right\}$ $\text{am. cost}_i \leq 3$

pop with resize



$n = \frac{cap}{4}$

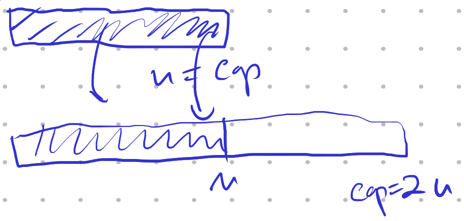
actual $\text{cost}_i = 1$

$\underline{\Delta\phi < 0}$ $\qquad \phi_i = 2n$

$\phi_{i+1} = 0$

$\text{am. cost}_i \leq 1$

push with resize



actual $\text{cost}_i = n + 1$

$\phi_i = n$

$\phi_{i+1} = 0$

$\text{amortised cost}_i = n + (-n) + 1 \leq 1 \in O(1)$

$\text{am. cost}_i \leq 3$

$\text{avg. am. cost} \leq 3$

$\Downarrow$

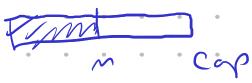
$\text{avg. cost}_i \leq 3$

$\Rightarrow$ am. cost is constant

$\phi = |2n - cap|$

ϕ = "badness" of system

$\phi = 0 \Rightarrow n = \frac{cap}{2}$



Summary

1. Direct counting (good, when it works)
2. Bank account method (transfer costs from op_i to op_j)
3. Potential-fct. method