Amortized Analysis

Amortized analysis

An ***amortized analysis*** is any strategy for analyzing a sequence of operations to show that the average cost per operation is small, even though a single operation within the sequence might be expensive.

Even though we're taking averages, however, *probability is not involved*!

- An amortized analysis *guarantees* the *average performance* of each operation in the *worst case*.

Amortized Analysis

- Amortized analysis:
  - Consider a sequence of operations
  - Overall costs of them
  - Average that for each operation

- Different from
  - Average cost over all possible inputs
  - Expected cost in randomized analysis

Individual costly operation may occur, but due to constraints, cannot occur frequently

Amortized Analysis – contd.

- Not just consider one operation, but a sequence of operations on a given data structure.
- Average cost over a sequence of operations.
- Probabilistic analysis:
  - Average case running time: average over all possible inputs for one algorithm (operation).
  - If using probability, called expected running time.
- Amortized analysis:
  - No involvement of probability
  - Average performance on a sequence of operations, even some operation is expensive.
  - Guarantee average performance of each operation among the sequence in worst case.

Amortized analysis – contd.

Designing of an algorithm and the analysis of its running time are often closely intertwined.

Amortized analysis is not just an analysis tool, it is also a way of thinking about designing algorithms.

Three Methods of Amortized Analysis

- Aggregate analysis:
  - Total cost of n operations/n,
- Accounting method:
  - Assign each type of operation an (different) amortized cost
  - overcharge some operations,
  - store the overcharge as credit on specific objects,
  - then use the credit for compensation for some later operations.
- Potential method:
  - Same as accounting method
  - But store the credit as "potential energy" and as a whole.

Types of amortized analyses

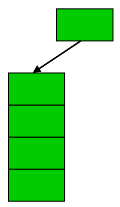
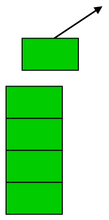
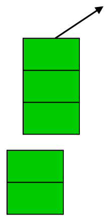
We’ve just seen an aggregate analysis.

The aggregate method, though simple, lacks the precision of the other two methods. In particular, the accounting and potential methods allow a specific ***amortized cost*** to be allocated to each operation.

Aggregate Analysis

- Show that for all n, a sequence of n operations take ***worst-case*** time T(n) in total
- In the worst case, the average cost, or ***amortized cost*** , per operation is T(n)/n.
- The ***amortized cost*** applies to each operation, even when there are several types of operations in the sequence.

Stack Example I

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
Worst-case cost:	O(1)	O(1)	O(min(|S|,k) = O(n)

Amortized cost: O(1) per op

Stack Example II

- Sequence of n push, pop, Multipop operations
  - Worst-case cost of Multipop is O(n)
  - Have n operations
  - Therefore, worst-case cost of sequence is O(n^2)

Stack Example II – contd.

- Stack operations:
  - PUSH(S,x), $O(1)$
  - POP(S), $O(1)$
  - MULTIPOP(S,k), min(s,k)

```
while not STACK-EMPTY(S) and $k>0$
    do POP(S)
        $k=k-1$
```

- Let us consider a sequence of n PUSH, POP, MULTIPOP.
  - The worst case cost for MULTIPOP in the sequence is $O(n)$, since the stack size is at most n.
  - thus the cost of the sequence is $O(n^2)$. Correct, but not tight.

Stack Example – contd.

- Observations
  - Each object can be popped only once per time that it's pushed
  - Have <= n pushes => <= n pops, including those in Multipop
  - Therefore total cost = O(n)
  - Average over n operations => O(1) per operation on average
- Notice that no probability involved

Second Example: Binary Counter

- Array A[0,…, k-1] represents a binary counter
- A[i]: 0 or 1
- A[0] is the least significant bit
- Initially A[i] = 0 for all $0 \leq i \leq k$

Incrementing Binary Counter

```
INCREMENT($A, k$)
$i \leftarrow 0$
while $i < k$ and $A[i] = 1$
    do $A[i] \leftarrow 0$
       $i \leftarrow i + 1$
if $i < k$
  then $A[i] \leftarrow 1$
```

Cost of each operation: #bit flips
can be O(1) to O(k)

Incrementing a Binary Counter

- k-bit Binary Counter: $A[0..k-1]$

$$x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$$

```
INCREMENT(A)
1.  $i \leftarrow 0$
2.  while $i < length[A]$ and $A[i] = 1$
3.      do $A[i] \leftarrow 0$        ▷ reset a bit
4.         $i \leftarrow i + 1$
5.  if $i < length[A]$
6.      then $A[i] \leftarrow 1$      ▷ set a bit
```

k-bit Binary Counter

Ctr	A[4]	A[3]	A[2]	A[1]	A[0]	
0	0	0	0	0	0	
1	0	0	0	0	1	
2	0	0	0	1	0	
3	0	0	0	1	1	
4	0	0	1	0	0	
5	0	0	1	0	1	
6	0	0	1	1	0	
7	0	0	1	1	1	
8	0	1	0	0	0	
9	0	1	0	0	1	
10	0	1	0	1	0	
11	0	1	0	1	1	

k-bit Binary Counter

Ctr	A[4]	A[3]	A[2]	A[1]	A[0]	*Cost*
0	**0**	**0**	**0**	**0**	**0**	*0*
1	**0**	**0**	**0**	**0**	**1**	*1*
2	**0**	**0**	**0**	**1**	**0**	*3*
3	**0**	**0**	**0**	**1**	**1**	*4*
4	**0**	**0**	**1**	**0**	**0**	*7*
5	**0**	**0**	**1**	**0**	**1**	*8*
6	**0**	**0**	**1**	**1**	**0**	*10*
7	**0**	**0**	**1**	**1**	**1**	*11*
8	**0**	**1**	**0**	**0**	**0**	*15*
9	**0**	**1**	**0**	**0**	**1**	*16*
10	**0**	**1**	**0**	**1**	**0**	*18*
11	**0**	**1**	**0**	**1**	**1**	*19*

Worst-case analysis

Consider a sequence of n insertions. The worst-case time to execute one insertion is $\Theta(k)$. Therefore, the worst-case time for n insertions is $n \cdot \Theta(k) = \Theta(n \cdot k)$.

WRONG! In fact, the worst-case cost for n insertions is only $\Theta(n) \ll \Theta(n \cdot k)$.

Let's see why.

Note: You'll be correct If you'd said $O(n \cdot k)$. But, it's an overestimate.

Tighter analysis

Ctr	A[4]	A[3]	A[2]	A[1]	A[0]	*Cost*
0	**0**	**0**	**0**	**0**	0	*0*
1	**0**	**0**	**0**	0	1	*1*
2	**0**	**0**	**0**	**1**	0	*3*
3	**0**	**0**	0	1	1	*4*
4	**0**	**0**	**1**	**0**	0	*7*
5	**0**	**0**	**1**	0	1	*8*
6	**0**	**0**	**1**	**1**	0	*10*
7	**0**	0	1	1	1	*11*
8	**0**	**1**	**0**	**0**	0	*15*
9	**0**	**1**	**0**	0	1	*16*
10	**0**	**1**	**0**	**1**	0	*18*
11	**0**	**1**	0	1	1	*19*

Total cost of n operations

A[0] flipped every op	n
A[1] flipped every 2 ops	$n/2$
A[2] flipped every 4 ops	$n/2^2$
A[3] flipped every 8 ops	$n/2^3$
… … … …	…
A[i] flipped every 2^i ops	$n/2^i$

Tighter analysis (continued)

Cost of n increments

$$= \sum_{i=1}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^i} \right\rfloor$$

$$< n \sum_{i=1}^{\infty} \frac{1}{2^i} = 2n$$

$$= \Theta(n)$$

Thus, the average cost of each increment operation is $\Theta(n)/n = \Theta(1)$.

Observation

Consider a sequence of n Increment (A) operations

bit	frequency of flipping	#times of flips in n operations
0	every time	n
1	half of time	$n/2$
2	1/4 time	$n/4$
⋮	⋮	⋮
i	$1/2^i$ time	$n/2^i$

Total Cost

- Total cost for n operations
  - = total number of flips for all bits

$$
\begin{aligned}
&= \sum_{i=0}^{k-1} \lfloor n/2^i \rfloor \\
&< n \sum_{i=0}^{\infty} 1/2^i \\
&= n\left(\frac{1}{1-1/2}\right) \\
&= 2n\ .
\end{aligned}
$$

Remarks

- Initial condition is important
  - Empty stack/empty binary array
- Not average over input distribution
  - Worst cast in fact
  - But amortized cost
- Use inherent constraints of the problem
  - To limit high-cost operations

Accounting Method

- Same intuition as aggregate analysis
- Assign charges to operations
  - Different operation may get different charge
  - Credit: charge - actual cost
    - Can be used later when charge is less than actual cost for some operation
  - Save total extra credits in a bank account

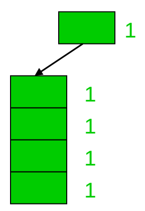
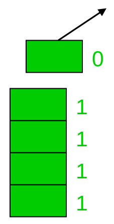
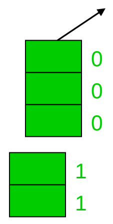
Amortized Analysis: Accounting Method

- Idea:
  - Assign differing charges to different operations.
  - The amount of the charge is called amortized cost.
  - amortized cost is more or less than actual cost.
  - When amortized cost > actual cost, the difference is saved in specific objects as credits.
  - The credits can be used by later operations whose amortized cost < actual cost.
- As a comparison, in aggregate analysis, all operations have same amortized costs.

Accounting Method (cont.)

- Conditions:
  - suppose actual cost is c_i for the ith operation in the sequence, and amortized cost is c_i',
  - $\sum_{i=1}^{n} c_i' \geq \sum_{i=1}^{n} c_i$ should hold.
    - since we want to show the average cost per operation is small using amortized cost, we need the total amortized cost is an upper bound of total actual cost.
    - holds for all sequences of operations.
  - Total credits is $\sum_{i=1}^{n} c_i' - \sum_{i=1}^{n} c_i$, which should be nonnegative,
    - Moreover, $\sum_{i=1}^{t} c_i' - \sum_{i=1}^{t} c_i \geq 0$ for any $\boldsymbol{t} > \mathbf{0}$.

Stack Example

3 ops:			
	Push(S,x)	Pop(S)	Multi-pop(S,k)
•Assigned cost:	2	0	0
•Actual cost:	1	1	min(|S|,k)

Push(S,x) pays for possible later pop of x.

Stack Example

charge

operation	actual cost	amortized cost
PUSH	1	2
POP	1	0
MULTIPOP	$\min(k, s)$	0

Stack Example: Accounting Methods

- When pushing an object, pay \$2
  - \$1 pays for the push
  - \$1 is prepayment for it being popped by either pop or Multipop
  - Since each object has \$1, which is credit, the credit can never go negative
  - Therefore, total amortized cost = O(n), is an upper bound on total actual cost

Accounting Method: Stack Operations

- Actual costs:
  - PUSH :1, POP :1, MULTIPOP: min(s,k).
- Let assign the following amortized costs:
  - PUSH:2, POP: 0, MULTIPOP: 0.
- Similar to a stack of plates in a cafeteria.
  - Suppose \$1 represents a unit cost.
  - When pushing a plate, use one dollar to pay the actual cost of the push and leave one dollar on the plate as credit.
  - Whenever POPing a plate, the one dollar on the plate is used to pay the actual cost of the POP. (same for MULTIPOP).
  - By charging PUSH a little more, do not charge POP or MULTIPOP.
- The total amortized cost for n PUSH, POP, MULTIPOP is $O(n)$, thus $O(1)$ for average amortized cost for each operation.
- Conditions hold: total amortized cost ≥total actual cost, and amount of credits never becomes negative.

Remarks

- Different operation may get different amortized cost
- Careful choose amortized cost
- Save extra credits for expensive operations
- In order to bound total actual cost
  - at any time, total extra credits left ≥ 0

Binary Counter Example

- Charging system: Charge $2 for flipping a bit from *0* to *1*
  - $1 pays for setting a bit to 1
  - $1 is prepayment for flipping it back to 0
  - Have $1 of credit for every 1 in the counter.
  - Therefore, credit >= 0.
- Amortized cost:
  - Cost of resetting bits to 0 is paid by credit.
  - At most 1 bit is set to 1.
  - Therfore, amortized cost =< $2.
  - For n operations, amortized cost = O(n)

Potential Method

- Yet another way to upper bound total costs
- Similar to accounting method: something prepaid is used later.

Potential Method – contd.

- Instead of storing extra credit
  - And assign credit for each object (e.g, each element in the stack, a “1” bit in the array)
- Different from accounting method
  - Use the prepaid work not as credit, but as “potential energy”, or “potential”. *Potential* represents prepaid work.
  - Potential energy is used for future operations.
  - The potential is associated with the entire data structure rather than with specific objects within the data structure.

Potential Method

IDEA: View the bank account as the potential energy (*à la* physics) of the dynamic set.

Framework:

- Start with an initial data structure D_0.
- Operation i transforms D_{i-1} to D_i.
- The cost of operation i is c_i.
- Define a ***potential function*** $\Phi : \{D_i\} \to \mathbb{R}$, such that $\Phi(D_0) = 0$ and $\Phi(D_i) \geq 0$ for all i.
- The ***amortized cost*** $\hat{c}_i$ with respect to Φ is defined to be $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$.

Potential Method II

- Like the accounting method, but think of the credit as ***potential*** stored with the ***entire data structure***.
  - Accounting method stores credit with specific objects while potential method stores potential in the data structure as a whole.
  - Can release potential to pay for future operations
- Most flexible of the amortized analysis methods).

Understanding Potentials

$$\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\textit{potential difference } \Delta\Phi_i}$$

- If $\Delta\Phi_i > 0$, then $\hat{c}_i > c_i$. Operation i stores work in the data structure for later use.
- If $\Delta\Phi_i < 0$, then $\hat{c}_i < c_i$. The data structure delivers up stored work to help pay for operation i.

Amortized Costs Bound the True Costs

The total amortized cost of n operations is

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right)$$

Summing both sides.

Amortized Costs Bound the True Costs

The total amortized cost of n operations is

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0)\end{aligned}$$

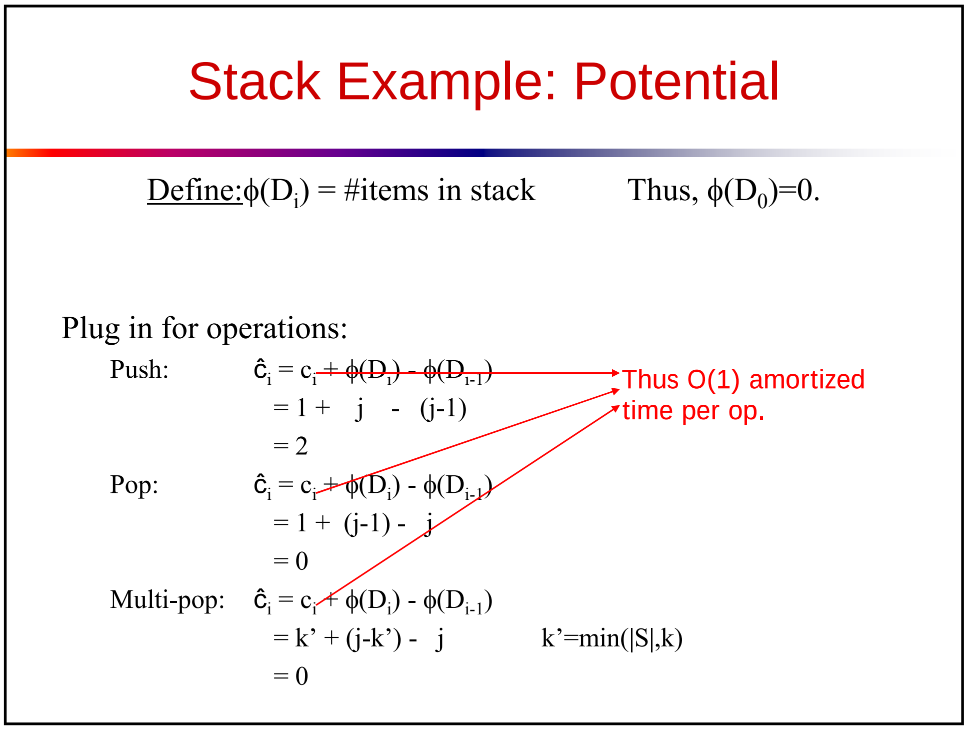
The series telescopes.

Amortized Costs Bound the True Costs

The total amortized cost of n operations is

$$\begin{aligned}\sum_{i=1}^{n} \hat{c}_i &= \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1})\right) \\ &= \sum_{i=1}^{n} c_i + \Phi(D_n) - \Phi(D_0) \\ &\ge \sum_{i=1}^{n} c_i\end{aligned}$$

since $\Phi(D_n) \ge 0$ and $\Phi(D_0) = 0$.



Stack Example: Potential

Define: $\phi(D_i)$ = #items in stack   Thus, $\phi(D_0)=0$.

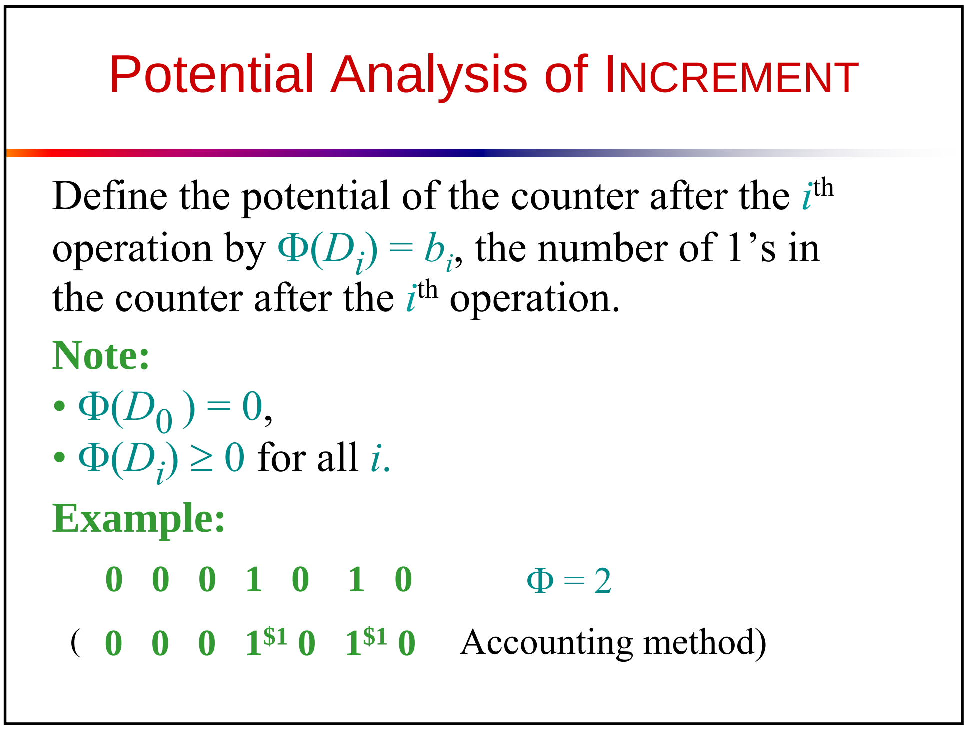
Plug in for operations:

Push: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= 1 + j - (j-1)$
$= 2$

Pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= 1 + (j-1) - j$
$= 0$

Multi-pop: $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$
$= k' + (j-k') - j$ $k'=\min(|S|,k)$
$= 0$

Thus O(1) amortized time per op.



Potential Analysis of INCREMENT

Define the potential of the counter after the ith operation by $\Phi(D_i) = b_i$, the number of 1's in the counter after the ith operation.

Note:

- $\Phi(D_0) = 0$,
- $\Phi(D_i) \geq 0$ for all i.

Example:

0 0 0 1 0 1 0 $\Phi = 2$

(**0 0 0 $1^{\$1}$ 0 $1^{\$1}$ 0**   Accounting method)

Calculation of Amortized Costs

Assume ith INCREMENT resets t_i bits (in line 3).
Actual cost $c_i = (t_i + 1)$
Number of 1's after ith operation: $b_i = b_{i-1} - t_i + 1$

The amortized cost of the ith INCREMENT is

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= (t_i + 1) + (1 - t_i) \\ &= 2\end{aligned}$$

Therefore, n INCREMENTS cost $\Theta(n)$ in the worst case.

Remarks

- All three methods based on the similar observations and intuition
- But flexibility of the methods is different
- Potential method is more abstract, but also most powerful

Summary

- Amortized analysis
  - Different from probabilistic analysis
- Three methods and their differences
- How to analyze

The End