Recitation 2

Amortized Analysis & Competitive Analysis

Rebecca Lin

rebeccayelin.github.io/6.1220-SP25

Agenda

Amortized Analysis

- **Overview**
- **Methods: Aggregate, Accounting, and Potential**
- **Worked Example: Dynamic List (On the Board)**

Competitive Analysis (On the Board)

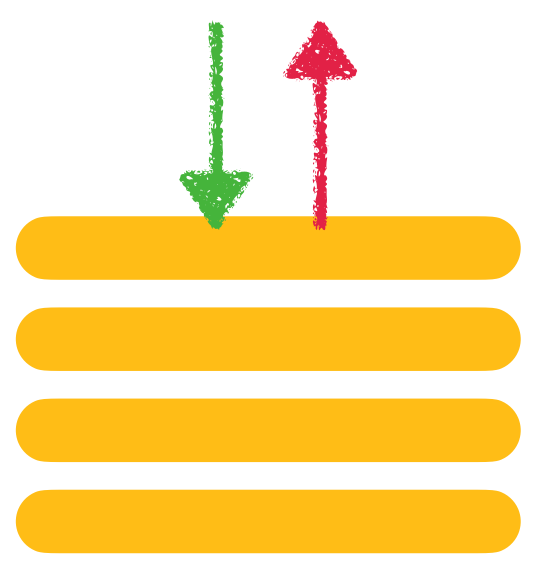
- **Worked Example: Competitive Scheduling**
- **Worked Example: LRU Paging**

Amortized Analysis

- Idea: Tight upper bound for a *sequence* of operations
  - Not interested in the cost of any individual operation, but total cost of entire sequence of operations
  - Not to be confused with average-case analysis
- Methods for analysis: Aggregate, Accounting, Potential

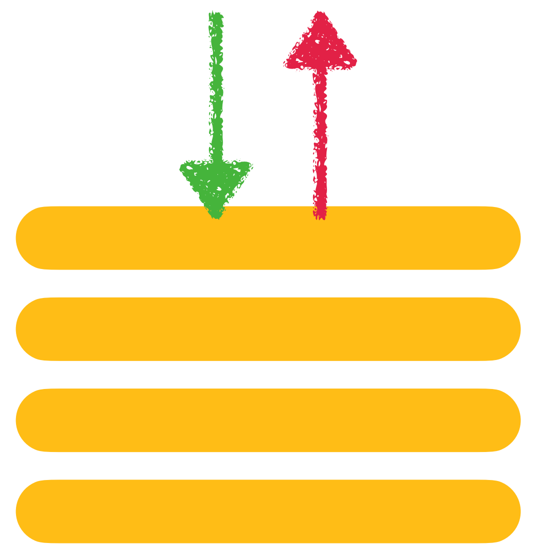
Stacks and Queues

- **Stack** — last in, first out (LIFO), comprising two **cost-1** operations:
  - PUSH(x) adds x to the top of the stack
  - POP() removes and returns the item at the top of the stack

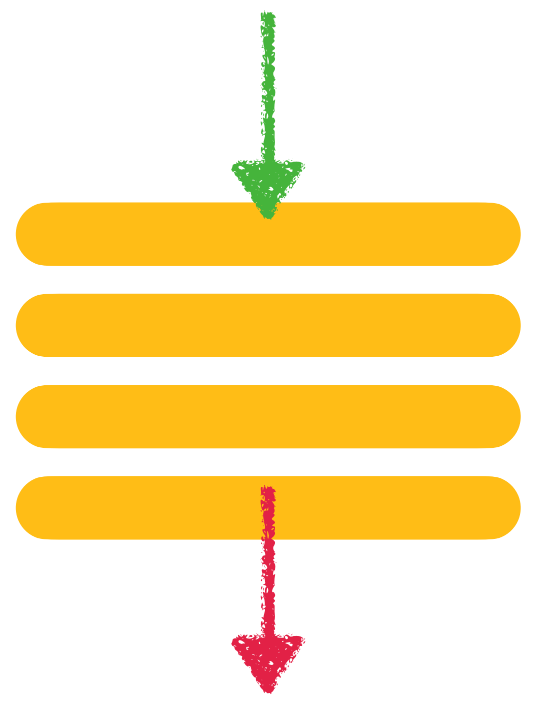


Stacks and Queues

- **Stack** — last in, first out (LIFO), comprising two **cost-1** operations:
  - PUSH(x) adds x to the top of the stack
  - POP() removes and returns the item at the top of the stack



- **Queue** — first in, first out (FIFO):
  - ENQUEUE(x) add x to the back of the queue
  - DEQUEUE() removes and returns the item at the front of the queue



Queue Using Two Stacks

Worked Example

Implement a queue given two stacks s_1 and s_2:

- ENQUEUE(x): Push x onto s_1
- DEQUEUE():
  - If s_2 is empty, transfer all elements from s_1 to s_2 (pop + push)
  - Pop from s_2

Cost? ENQUEUE is $O(1)$, DEQUEUE is $O(n)$ in the worst case:
Hence, we can naively bound the cost of n operations by $O(n^2)$.

Aggregate Method

Worked Example

Idea:

- Aggregate the cost of n operations
- Divide the total cost by n to achieve amortized cost

Let's observe any mixture of n ENQUEUE and DEQUEUE operations:

- Every element added to the queue incurs at most **four** cost-1 operations:
  - PUSH to s_1, POP + PUSH to move from s_1 to s_2, POP from s_2
- At most n elements are added, as # of DEQUEUEs $\leq$ # of ENQUEUEs
- Hence, total cost of n operations is $O(n)$, so amortized cost is $O(1)$

Accounting Method

Definition

Idea: "Pay in advance" extra coins during low-cost operations to "subsidize" the cost of later expensive ones.

- Let c_i be the cost of operation i
- Assign an amortized cost $\hat{c}_i$ to each operation such that $\sum_i c_i \leq \sum_i \hat{c}_i$

Note: It is possible we assign $\hat{c}_i < c_i$, however, a nonnegative balance must maintained for any sequence of operations.

Accounting Method

Worked Example

Claim: Assign an amortized cost of 4 coins to ENQUEUE. Then the amortized cost for DEQUEUE is 0 coins.

- 1 coin pays for the initial push of element x to s_1
- 2 coins used if x is ever moved from s_1 to s_2
- 1 coin used if x is ever popped from s_2 due to a DEQUEUE

DEQUEUE is considered free due to sufficient credit stored.

We pay at most 4 coins per operation, so the amortized cost is $O(1)$.

Potential Method

Definition

Idea: The potential function Φ assigns each state of the data structure a nonnegative value representing prepaid work, i.e., the "potential energy" at that state.

Amortized Cost:

- For operation i $\quad \hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = c_i + \Delta\Phi_{i-1}$
- For k operations $\quad \sum_i^k \hat{c}_i = \sum_i^k c_i + \Phi_i - \Phi_{i-1} = \sum_i^k c_i + \Phi_k - \Phi_0$

Require $\Phi_i - \Phi_0 \geq 0$ but small. Why?

For simplicity: Set $\Phi_0 = 0$ and ensure $\Phi_i \geq 0$ for all i.

Potential Method

Worked Example

Intuition: Choose Φ to increase during inexpensive operations (i.e., prepay) and drop during expensive ones (i.e., cash in).

For simulating a queue using stacks:

- Let $s_1^{(i)}$ refer to s_1 immediately following operation i
- Define $\Phi_i = 2|s_1^{(i)}|$
  - How does the intuition relate?
  - Is this a valid potential function?

Potential Method

Worked Example

ENQUEUE:

- **Actual Cost:** $c_i = 1$ for one push onto s_1
- **Change in Potential:** $\Delta\Phi_i = 2$ since $|s_1|$ increases by 1
- **Amortized Cost:** $\hat{c}_i = c_i + \Delta\Phi_1 = 1 + 2 = 3$

Potential Method

Worked Example

DEQUEUE:

Case 1: s_2 is not empty

- **Actual Cost:** $c_i = 1$ for a pop from s_2
- $\Delta\Phi_i = 0$ since $|s_1|$ is unchanged
- **Amortized Cost:** $\hat{c}_i = 1 + 0 = 1$

Case 2: s_2 is empty

- **Actual Cost:** $c_i = 2|s_1| + 1$
  - $|s_1|$ pops from s_1
  - $|s_1|$ pushes to s_2
  - 1 pop from s_2
- $\Delta\Phi_i = -2|s_1|$ since s_1 becomes empty
- **Amortized Cost:**
  $\hat{c}_i = (2|s_1| + 1) - 2|s_1| = 1$