Applied Algorithms

CSCI-B505 / INFO-I500

Lecture 6.

Amortized Analysis - 1

M. Oguzhan Kulekci, September 12/13, 2022

- Amortized Analysis

  - Aggregate Method

  - Accounting Method

  - Potential Method

Amortized Analysis ?

Amortize:
- *gradually write off the initial cost of (an asset) over a period*
- *reduce or pay off (a debt) with regular payments*

- In an algorithm, there may be cheap operations and expensive operations.

- Regular worst-case analysis assumes the expensive operations always dominate the execution.

- However, there can be a **deterministic** limit on the number of times *expensive* happens.

Let's see on an example…

Queue Implementation with Two Stacks

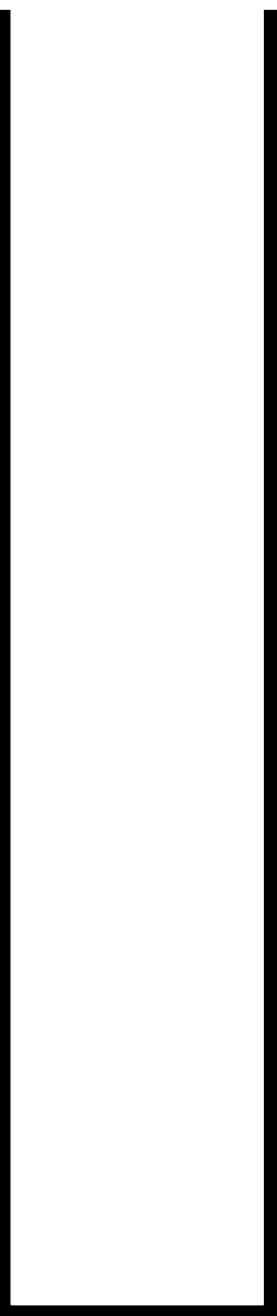Implement a queue by using two stacks.

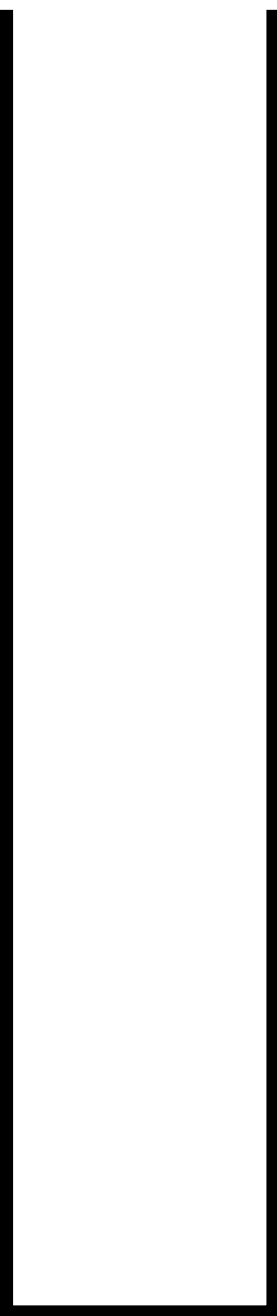**Enqueue(x):**
        Push x into stack-1.

Dequeue():
  If stack-2 is empty, then
        Pop everything from stack-1 and push into stack-2;
  Pop from stack-2

Enqueue(7)	Enqueue(2)	Enqueue(9)	Dequeu()	Enqueue(8)
Dequeu()	Dequeu()	Enqueue(1)	Enqueue(2)	Dequeu()

Stack-1   Stack-2

Queue Implementation with Two Stacks

Assume n insert or fetch operations will be executed.
What will be the time-complexity on this implementation?

```
for (i=1 to n){
    operation = randomSelect(enqueue, dequeue);
    execute the operation;}
```

- Enqueue() is cheap, worst case $O(1)$-time.
- Dequeue() is expensive, worst-case $O(n)$-time.
- If I always do the expensive operation then worst-case complexity becomes $O(n^2)$!?

Is this correct ?

Queue Implementation with Two Stacks

Stack-1   Stack-2

Enqueue(7)

Enqueue(2)

Enqueue(9)

Stack-1   Stack-2

Dequeu()

Stack-1   Stack-2

Enqueue(1), Dequeu()
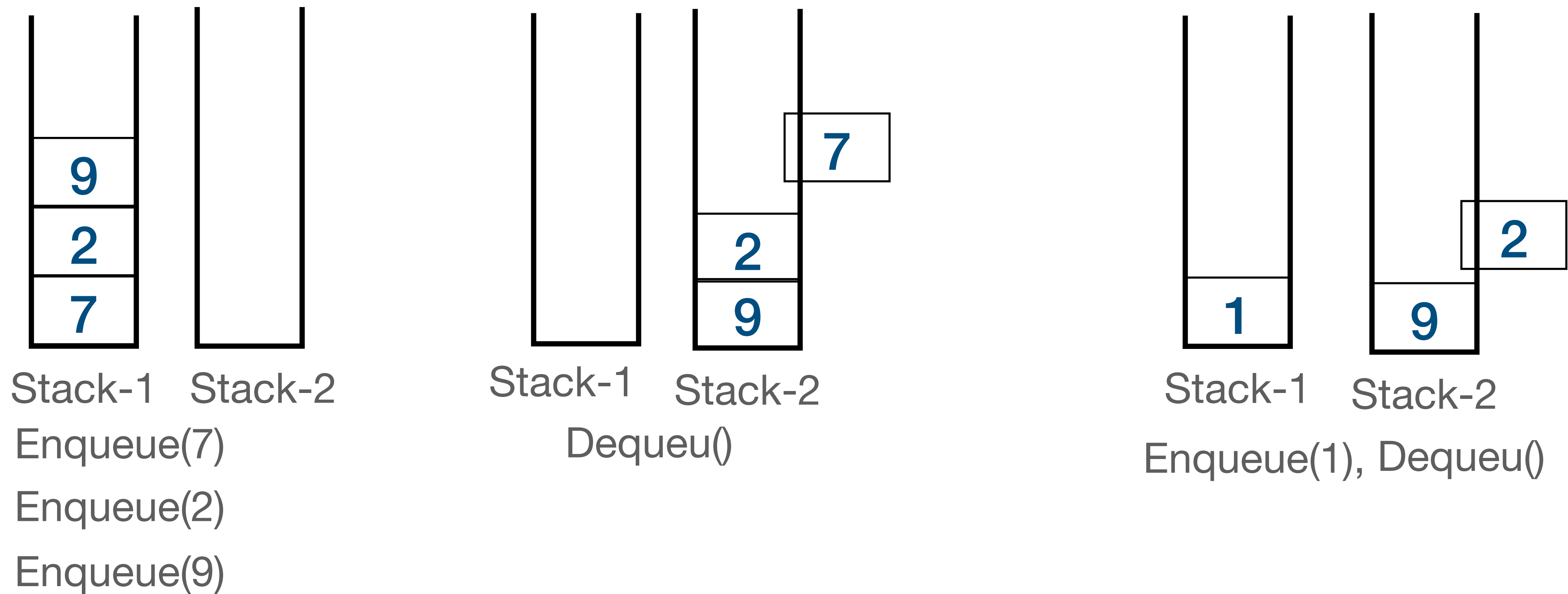
**Once an expensive 'dequeue' happens,
some of the following 'dequeue's will always be cheap.**

Queue Implementation with Two Stacks

Once an expensive dequeue happens, the following ones are always cheap.

- How many times an item is inserted into the stack-1 and stack-2 ?

- How many times it is popped from stack-1 and stack-2 ?

- For n enquee/dequee, **at most** $4n$ push/pop are achieved

- $\dfrac{4n}{n} = 4 \in O(1)$ **per each enqueue/dequeu operation.**

This is **NOT average-case** analysis,
but a **worst-case** analysis.

Binary Counter

Assume we have a k-bit binary counter, and the cost of incrementing this counter is defined as being equal to the number of bits flipped. What is the cost of incrementing this counter n times?

A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	COST
0	0	0	0	0	0	
0	0	0	0	0	**1**	**1**
0	0	0	0	**1**	**0**	**2**
0	0	0	0	1	**1**	**1**
0	0	0	**1**	**0**	**0**	**3**
0	0	0	1	0	**1**	**1**

INCREMENT(A)

1 $i = 0$
2 **while** $i < A.length$ and $A[i] == 1$
3      $A[i] = 0$
4      $i = i + 1$
5 **if** $i < A.length$
6      $A[i] = 1$

Binary Counter

Regular worst-case analysis:

- At most how many bits can be flipped ?

  - All of the k bits, e.g., 011111 →100000

- Thus, if we consider n increments then it makes $O(n \cdot k)$

A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	COST
0	0	0	0	0	0	
0	0	0	0	0	**1**	**1**
0	0	0	0	**1**	0	**2**
0	0	0	0	1	**1**	**1**
0	0	0	**1**	**0**	**0**	**3**
0	0	0	1	0	1	**1**

....

$\text{INCREMENT}(A)$

1 $i = 0$
2 **while** $i < A.length$ and $A[i] == 1$
3      $A[i] = 0$
4      $i = i + 1$
5 **if** $i < A.length$
6      $A[i] = 1$

However, it is not possible to have consecutive increments with k-bits flip! So…

Binary Counter

A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	COST
0	0	0	0	0	0	
0	0	0	0	0	**1**	**1**
0	0	0	0	**1**	**0**	**2**
0	0	0	0	1	**1**	**1**
0	0	0	**1**	**0**	**0**	**3**
0	0	0	1	0	**1**	**1**
0	0	0	1	**1**	**0**	**2**

A[0] flips at each increment

A[1] flips once at each 2 increments

A[2] flips once at each 4 increments

......

A[k] flips once at each 2^i increments

So total cost of n increment operations is

$$n + \left\lfloor \frac{n}{2} \right\rfloor + \left\lfloor \frac{n}{4} \right\rfloor + \ldots + \left\lfloor \frac{n}{2^{k-1}} \right\rfloor < \sum_{i=0}^{\infty} \frac{1}{2^i} = 2n$$

n increment operations cost less than 2n flips.

Thus, each increment costs 2, which makes $O(1)$ time per increment.

Aggregate Method

- Compute the total cost of n operations.

- Divide this cost by n to compute the cost of one operation.

- This is the **aggregate** method of amortized analysis.

- We have two alternative approaches, *accounting* and *potential*

Accounting Method

- Again we assume n operations will be achieved.

- We compute an **amortized cost** per operation

- Before each operation, we deposit in an account the **amortized cost** of that operation.

- Each operation drops exactly the regular **required** amount from the account, where the excess amount from cheap operations are expected to **amortize** the expensive ones.

- If there appears a case that there is not enough money in the account (**bankruptcy**), then the operation can not be performed. Thus, it should be strictly avoided.

- **What should we assume the amortized cost to avoid the bankruptcy ?**

Accounting Method

$$\sum_{i=1}^{k} c_i \leq \sum_{i=1}^{k} \hat{c}_i$$

- The regular cost of the ith operation is c_i

- The amortized cost of ith operation is $\hat{c}_i$

- **For any $k = 1\ldots n$, the total regular cost should never exceed the total amortized cost.**

Accounting Method
Queue with Two Stacks

	Stack operations	Stack-1	Stack-2	Actual Cost	Deposit	Remaining Balance
Enqueue(7)	1-push	7		1	$2	$1
Enqueue(2)	1-push	7.2		1	$2	$2
Enqueue(9)	1-push	7,2,9		1	$2	$3
Dequeue()	3-pop, 3-push, 1-pop		9.2	7	$0	**BANKRUPTCY** !
Dequeue()	1-pop		2	1		
Enqueue(8)	1-push	8	2	1		
Dequeue()	1-pop	8		1		
Enqueue(7)	1-push	8.7		1		
Dequeue()	2-pop, 2-push, 1-pop		2.7	5		

Assume the amortized cost for enqueue is **$2** , and dequeue is **free** !

Not a good choice ! We may face a bankruptcy

Accounting Method

Queue with Two Stacks

	Stack operations	Stack-1	Stack-2	Actual Cost	Deposit	Remaining Balance
Enqueue(7)	1-push	7		1	$4	$3
Enqueue(2)	1-push	7.2		1	$4	$6
Enqueue(9)	1-push	7,2,9		1	$4	$9
Dequeue()	3-pop, 3-push, 1-pop		9.2	7	**$0**	$2
Dequeue()	1-pop		2	1	**$0**	$1
Enqueue(8)	1-push	8	2	1	$4	$4
Dequeue()	1-pop	8		1	**$0**	$3
Enqueue(7)	1-push	8.7		1	$4	$6
Dequeue()	2-pop, 2-push, 1-pop		7	5	**$0**	$1

If we the amortized cost for enqueue is **$4** , and dequeue is free, then it seems no bankruptcy !
We know it takes no more than 4 stack operations per each item in the queue. So, we pay $4 dollars at the enqueue phase, and use the remaining $3 during the later dequeue operations.

Accounting Method

Binary Counter

INCREMENT(A)

```
1   i = 0
2   while i < A.length and A[i] == 1
3       A[i] = 0
4       i = i + 1
5   if i < A.length
6       A[i] = 1
```

$1 \to 0$ flipping, FREE, no cost

$0 \to 1$ flipping, **$? amortized cost**

- There are two different flip operations as $1 \to 0$ and $0 \to 1$.
- At each increment **some number of** ones flip into 0, and **one** zero at the end flips to 1.
- Assume $1 \to 0$ is free, so no worries on '*some number of*'
- What should be the amortized cost of $0 \to 1$ to accommodate this?

Potential Method

$$\sum_{i=0}^{i=n} \hat{c}_i \;=\; \sum_{i=0}^{i=n} c_i + \phi(D_i) - \phi(D_{i-1}) \;=\; \phi(D_n) - \phi(D_0) + \sum_{i=0}^{i=n} c_i$$

- Again we consider n operations, but the focus is on the used data structure D.

- Function $\phi(D)$ that defines the potential energy of the data structure D.

- The amortized cost $\hat{c}_i = c_i + \phi(D_i) - \phi(D_{i-1})$, where c_i is the actual real cost.

- If $\phi(D_n) \geq \phi(D_0)$ can be maintained, then the amortized cost is fine since potential never goes negative.

- **The issue is to propose such a ϕ function.**

Potential Method

Queue with two stacks

Assume $\phi = 2 \cdot x$, where x is the number of items in stack-1.

After enqueue operation the potential increases by $2 = \phi(k+1) - \phi(k)$.

Therefore, the amortized cost is $\hat{c}_E = c_E + \phi(k+1) - \phi(k) = 1 + 2 = 3$.

How about amortized cost of $\hat{c}_D$. Two cases:
1) Stack-2 is not empty. Then, no change in the potential and the amortized cost is equal to actual cost of 1.
2) Stack-2 is empty, and stack-1 has k elements. Then the potential difference is $-2k$. The actual cost is $2k + 1$ (why?). So, amortized cost is $-2k + 2k + 1 = 1$.

Potential Method

Binary counter

ϕ **is the number of set bits (equal to 1) in the counter.**

After an increment, assume t_i bits are flipped from 1 to 0. Then the actual cost is $t_i + 1$.

$$\phi(D_i) - \phi(D_{i-1}) = [\phi(D_{i-1}) - t_i + 1] - \phi(D_{i-1}) = 1 - t_i \, .$$

Then the amortized cost is $t_i + 1 + 1 - t_i = 2 \, .$

Reading assignment

- Read chapter 14 Amortized Analysis from Cormen.

- Read the paper 'Amortized Computational Complexity' by Tarjan, which dates back to 1985, on a very nice review of what amortized analysis is. Here is the link https://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/Amortized.pdf

- Yet another paper I suggest you to look at is Amortized Efficiency of List Update and Paging Rules available at https://scholar.google.com/scholar?output=instlink&q=info:gEIaOowSipkJ:scholar.google.com/&hl=en&as_sdt=0,15&scillfp=3816853723830843295&oi=lle

- We will study some further examples in the next lecture.