CSC263 Tutorial #6

Amortized Analysis

February 17, 2023

Things covered in this tutorial

- ⋆ What is amortized analysis?
- ⋆ What is the accounting method?
- ⋆ How do I simulate a queue with two stacks efficiently?

Amortized analysis

... analyzes the **average runtime per operation** of a **sequence of operations**.
Not analyzing the runtime of a single operation!

Amortized analysis

```
dynamic_array_insert(A, x):
  if A is empty:
    A = new array of size 1
  if A is full:
    make new array A’ with length 2 * |A|
    copy everything in A into A’
    A = A’
  insert x into A
```

Amortized analysis

```
dynamic_array_insert(A, x):
  if A is empty:
    A = new array of size 1
  if A is full:
    make new array A’ with length 2 * |A|
    copy everything in A into A’
    A = A’
  insert x into A
```

Question: Suppose there are n elements in a dynamic array A. How many array write operations are needed to insert another element into A?

Amortized analysis

```
dynamic_array_insert(A, x):
  if A is empty:
    A = new array of size 1
  if A is full:
    make new array A’ with length 2 * |A|
    copy everything in A into A’
    A = A’
  insert x into A
```

Question: Suppose there are n elements in a dynamic array A. How many array write operations are needed to insert another element into A?

Answer:

- ⋆ If $n = 2^k$, then it takes $n + 1$ array writes: n to copy the n existing elements into the new array A', and 1 for the new element.
- ⋆ Otherwise it takes 1 array write: just copy the new element.

Amortized analysis

Question: Suppose there are n elements in a dynamic array A. How many array write operations are needed to insert another element into A?

Answer:

- ⋆ If $n = 2^k$, then it takes $n + 1$ array writes: n to copy the n existing elements into the new array A', and 1 for the new element.
- ⋆ Otherwise it takes 1 array write: just copy the new element.

Worst case runtime for a single insert operation:

Amortized analysis

Question: Suppose there are n elements in a dynamic array A. How many array write operations are needed to insert another element into A?

Answer:

- ⋆ If $n = 2^k$, then it takes $n + 1$ array writes: n to copy the n existing elements into the new array A', and 1 for the new element.
- ⋆ Otherwise it takes 1 array write: just copy the new element.

Worst case runtime for a single insert operation: $O(n)$!

Amortized analysis

Question: Suppose there are n elements in a dynamic array A. How many array write operations are needed to insert another element into A?

Answer:

- ⋆ If $n = 2^k$, then it takes $n + 1$ array writes: n to copy the n existing elements into the new array A', and 1 for the new element.
- ⋆ Otherwise it takes 1 array write: just copy the new element.

Worst case runtime for a single insert operation: $O(n)$!

But the worst case doesn't occur too often...

Accounting method

Question: Suppose I have an empty dynamic array, and I insert into the array n times. What's the worst case runtime **per operation**?

Accounting method

Question: Suppose I have an empty dynamic array, and I insert into the array n times. What's the worst case runtime **per operation**?

Answer: $O(1)$. Proven using the **accounting method**!

Accounting method

Question: Suppose I have an empty dynamic array, and I insert into the array n times. What's the worst case runtime **per operation**?

Answer: $O(1)$. Proven using the **accounting method**!

Accounting method

Let's say we earn \$1 per insert call, and pay \$1 per array write. Do we ever run out of money?

Accounting method

Let's say we earn \$1 per insert call, and pay \$1 per array write. Do we ever run out of money?

We run out of money!

Accounting method

Let's say we earn \$3 per insert call, and pay \$1 per array write. Do we ever run out of money?

Accounting method

Let's say we earn \$3 per insert call, and pay \$1 per array write. Do we ever run out of money?

No, we will always have enough.

Accounting method

$\leftarrow 3$

$'s left for each element.

Accounting method

$\boxed{2} \qquad \leftarrow 3$

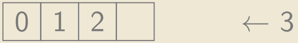
\$'s left for each element.

Accounting method

$\boxed{1}\boxed{2} \qquad \leftarrow 3$

$'s left for each element.

Accounting method



$\boxed{0}\boxed{1}\boxed{2}\boxed{} \qquad \leftarrow 3$

$'s left for each element.

Accounting method

$$\boxed{0}\boxed{1}\boxed{2}\boxed{2} \qquad \leftarrow 3$$

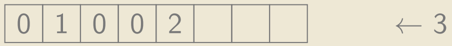
\$'s left for each element.

Accounting method

$$\boxed{1}\boxed{2}\boxed{1}\boxed{1} \qquad \leftarrow 3$$

$'s left for each element.

Accounting method



$'s left for each element.

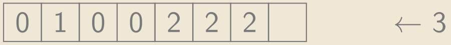
Accounting method

0	1	0	0	2	2		

$\leftarrow 3$

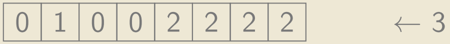
$'s left for each element.

Accounting method



$'s left for each element.

Accounting method



$'s left for each element.

Accounting method

1	2	1	1	1	1	1	1

$\leftarrow 3$

$'s left for each element.

Accounting method

0	1	0	0	0	0	0	0	2							

$\leftarrow 3$

$'s left for each element.

Queue with two stacks

```
def Enqueue(Q, x):
    if Size(S1) >= 12 and IsEmpty(S2):
        while not IsEmpty(S1):
            Push(S2, Pop(S1))
    Push(S1, x)
    return

def Dequeue(Q):
    if IsEmpty(S2):
        if IsEmpty(S1):
            error "Dequeuing from an empty queue!"
        else:
            while not IsEmpty(S1):
                Push(S2, Pop(S1))
    return Pop(S2)
```

Task: Enqueue the numbers 1-15. Then dequeue 15 times.