CSCE 411: Design and Analysis of Algorithms

Lecture 7: Amortized Analysis

Date: February 4, 2025 *Nate Veldt, updated by Samson Zhou*

Course Logistics

- Amortized analysis: Chapter 17
- First test is next Thursday; Review will be held next Tuesday

1 Simple Iterative Runtime Analysis

For simple iterative algorithms, a basic approach is to proving a runtime is to:

2 The Multi-pop Stack

Consider a stack S that has the following three operations

- Push(S, v): puts value v on the top of the stack
- Pop(S): removes value on the top of the stack
- IsEmpty(S): returns whether S is empty
- Multipop(S, k):
  - Calls Pop
  - Pops
  - k can be any integer number, different calls may involve different values of k

Question 1. Assume that a single push or pop operation takes $O(1)$ time. If we start with an empty stack, what is the worst-case runtime bound we obtain for applying n pop, push, and multipop operations, by applying a simple iterative runtime analysis?

A $O(1)$

B $O(n)$

C $O(nk)$

D $O(n^2)$

PushPopFun(n)

```
Initialize empty multipop stack S
for i = 1 to n do
    Generate random number b ∈ [0, 1]
    if b < 1/3 then
        Push(S, 1)
    else if b ∈ [1/3, 2/3) then
        Pop(S)
    else
        Generate random integer k ∈ [1, n]
        MultiPop(S, k)
    end if
end for
```

---

3 Amortized Analysis

In *amortized analysis*, we obtain improved runtime bounds for iterative methods by bounding the ____

3.1 Technique 1: Aggregate Analysis

Let $T(n)$ be:

____

Note that MultiPop(S, k) performs ____________________

The average cost per iteration (or *amortized cost*) is then

Theorem 3.1. PushPopFun*(n) makes at most* _______ *individual push/pop operations.*

An important distinction Our runtime analysis did not involve probabilistic reasoning. There are other notions of "average cost" for probabilistic algorithms, but we are not considering these. Our analysis holds independent of the random numbers generated in Algorithm PushPopFun.

3.2 Another approach for analyzing multipop stacks

If S is a multipop stack, and if it starts empty, then it is possible to pop an element v only if *we previously we pushed v onto S.*

Key idea: Let's "pay" in advance for an eventual Pop of that element.

- Whenever we push a new element,________________________________

- If/when that element is popped _____________________________________

- This is true whether we called _________________________________

- In n iterations, there are at most n push operations and so _______________________

Alternative view: Each time we "overpay" for a push, we put money in the bank to pay for future pops.

3.3 Technique 2: The accounting method

Define:

- c_i = the actual cost incurred at step i
- $\hat{c}_i$ the ____

Here's how the accounting method works:

- The runtime we want to bound is:
- We choose convenient $\hat{c}_i$ for each iteration so that

Multipop Stack Example In this example,

$$c_i = \text{number of pop/push operations in iteration } i.$$

$$\hat{c}_i = \begin{cases} \underline{\qquad\qquad} & \text{if we push in step } i \\ \\ \underline{\qquad\qquad} & \text{if we pop or multipop in step } i \end{cases}$$

3.4 Technique 3: The potential method

In the potential method, we identify some data structure and define:

- c_i: the actual cost incurred at step i
- D_i: the state of a data structure after the ith iteration
- Φ: potential function defined on data structure; $\Phi(D_i)$ is a real number
- $\hat{c}_i =$ ____________________________________ is the amortized cost

If ___________________________, we can bound actual cost in terms of amortized costs as follows:

Multipop Stack Example Define:
c_i = number of push/pop operations at step i
The data structure is ________________

Define $\Phi(D_i) =$ _____

Notice then that

$$\Phi(D_i) - \Phi(D_0) = \underline{\hspace{6cm}}$$

At step i, we "pay" $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$

In-Class Activity

1. Write down the amortized cost $\hat{c}_i$ in three different cases:
   (a) In iteration i, a pop happens
   (b) In iteration i, a push happens
   (c) In iteration i, a multipop happens
2. Bound the runtime using step 1.

Similarities and differences with the accounting method

- Similarity: both store "credit" and "pay" ahead of time
- The accounting method stores credit in individual steps
- The potential method stores credit as "potential" in a data structure
- For potential method, choose D_i and Φ and prove $\Phi(D_i) \geq \Phi(D_0)$. ____________________
- For accounting, there is no D_i or Φ. You must choose $\hat{c}_i$ and prove ________________________
- For both, bound _______________ to prove runtime guarantee.

4 Representing numbers in binary

We represent a number $x \in \mathbb{N}$ in binary by a vector of bits $A[0..k]$, where $A[i] \in \{0, 1\}$ and

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

5 The Binary Counter Problem

Let Increment be an algorithm that takes in a binary vector and adds one to the binary number that it represents.

Increment(A)
  $i = 0$
  while $i < A.length$ and $A[i] == 1$ **do**
    $A[i] = 0$
    $i = i + 1$
  end while
  if $i < A.length$ **then**

    ____

  end if

Question 2. If A is length k binary vector, what is the worst-case runtime for calling Increment(A)?

A $O(1)$

B $O(\log k)$

C $O(k)$

D $O(n)$

5.1 The actual cost of each iteration

Assume it takes $O(1)$ time to check an entry of A or to flip its bit. At each step, the runtime is then just $O(\text{number of flipped bits})$, so we will say the cost of an iteration is

$c_i =$ ____

Key idea: Separate the costs that happen ______________________

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

5.2 Technique 1: Aggregate Analysis

Let $T(n)$ be the total number of number of flipped bits in n increments.

Question 3. Look at the table on the last page, and observe how often the bit in position $A[j]$ is flipping. If you wish, fill in the number of total flipped bits at each increment. What is the total cost of calling INCREMENT n times?

A $O(n)$

B $O(k)$

C $O(nk)$

D $O(n^2)$

5.3 Technique 2: The accounting method

- c_i = the actual number of bits flipped
- $\hat{c}_i$ amortized cost for flipping bits: ______________________________

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

```
INCREMENT(A)
  i = 0
  while i < A.length and A[i] == 1 do
      A[i] = 0
      i = i + 1
  end while
  if i < A.length then
      A[i] = 1
  end if
```

5.4 Technique 3: The potential method

Step 0: Identify data structure, potential function, and costs.

- c_i: the actual number of flipped bits at iteration i
- data structure: ________________________, with state D_i after iteration i
- $\Phi(D_i) = b_i =$ ____
- $\hat{c}_i =$ ____________________________________

Step 1: Check that $\Phi(D_i) \geq \Phi(D_0)$

Step 2: Compute and bound amortized costs

Let t_i be the number of bits that are set to 0 (*while* loop in algorithm) in iteration i.

If $b_i = 0$, this means we had $A = [1\ 1\ 1\ \cdots\ 1]$ in iteration i, and incrementing turned it into $A = [0\ 0\ 0\ \cdots\ 0]$, meaning $b_{i-1} = t_i = k$.

Question 4. *If $b_i > 0$, which of these is the right expression for b_i?*

A $b_i = b_{i-1} + 1$

B $b_i = t_i + 1$

C $b_i = b_{i-1} + t_i$

D $b_i = b_{i-1} - t_i$

E $b_i = b_{i-1} + t_i + 1$

F $b_i = b_{i-1} - t_i + 1$

Whether or not $b_i = 0$, we have a bound of ____________________________

We can then compute the amortized cost and overall runtime bound: