CS4311
Design and Analysis of Algorithms

Lecture 14: Amortized Analysis I

About this lecture

- Given a data structure, amortized analysis studies in a sequence of operations, the average time to perform an operation
- Introduce amortized cost of an operation

- Three Methods for the Same Purpose

  (1) Aggregate Method

  (2) Accounting Method

  (3) Potential Method

This Lecture

Super Stack

- Your friend has created a super stack, which, apart from PUSH/POP, supports:

    SUPER-POP(k): pop top k items

- Suppose SUPER-POP never pops more items than current stack size

- The time for SUPER-POP is $O(k)$
- The time for PUSH/POP is $O(1)$

Super Stack

- Suppose we start with an empty stack, and we have performed n operations
  - But we don't know the order

Questions:

- Worst-case time of a SUPER-POP ?

  Ans. $O(n)$ time [why?]
- Total time of n operations in worst case ?

  Ans. $O(n^2)$ time [correct, but not tight]

Super Stack

- Though we don't know the order of the operations, we still know that:
  - There are at most n PUSH/POP

    ➔ Time spent on PUSH/POP = $O(n)$
  - # items popped by all SUPER-POP cannot exceed total # items ever pushed into stack

    ➔ Time spent on SUPER-POP = $O(n)$

So, total time of n operations = $O(n)$!!!

Amortized Cost

- So far, there are no assumptions on n and the order of operations. Thus, we have:

For any n and any sequence of n operations,
worst-case total time = $O(n)$

- We can think of each operation performs in average $O(n) / n = O(1)$ time

→ We say amortized cost = $O(1)$ per operation (or, each runs in amortized $O(1)$ time)

Amortized Cost

- In general, we can say something like:
  - OP_1 runs in amortized $O(x)$ time
  - OP_2 runs in amortized $O(y)$ time
  - OP_3 runs in amortized $O(z)$ time

Meaning:

For any sequence of operations with
$\#OP_1 = n_1$, $\#OP_2 = n_2$, $\#OP_3 = n_3$,
worst-case total time $= O(n_1x + n_2y + n_3z)$

Binary Counter

- Let us see another example of implementing a k-bit binary counter
- At the beginning, count is 0, and the counter will be like (assume k=5):

0	0	0	0	0

which is the binary representation of the count

Binary Counter

- When the counter is incremented, the content will change
- Example: content of counter when:

0	0	1	0	1

count = 5

cost = 2

0	0	1	1	0

count = 6

- The cost of the increment is equal to the number of bits flipped

Binary Counter

Special case:

When all bits in the counter are 1, an increment resets all bits to 0

1	1	1	1	1

$\Rightarrow$

0	0	0	0	0

count = MAX   cost = k count = 0

- The cost of the corresponding increment is equal to k, the number of bits flipped

Binary Counter

- Suppose we have performed n increments

Questions:

- Worst-case time of an increment ?

  Ans. $O(k)$ time

- Total time of n operations in worst case ?

  Ans. $O(nk)$ time [correct, but not tight]

Binary Counter

Let us denote the bits in the counter by

$$b_0, b_1, b_2, \ldots, b_{k-1},$$

starting from the right

b_4	b_3	b_2	b_1	b_0

Observation:

b_i is flipped only once in every 2^i increments

Precisely, b_i is flipped at x^{th} increment $\Leftrightarrow$ x is divisible by 2^i

Amortized Cost

- So, for n increments, the total cost is:

$$\sum_{i=0 \text{ to } k} \lfloor n / 2^i \rfloor$$

$$\leq \sum_{i=0 \text{ to } k} (n / 2^i) \quad < \quad 2n$$

- By dividing total cost with #increments,

➔ amortized cost of increment = O(1)

Aggregate Method

- The computation of amortized cost of an operation in super stack or binary counter follows similar steps:

1. Find total cost (thus, an "aggregation")
2. Divide total cost by #operations

This method is called Aggregate Method