Xingya Ren
260784116

COMP251 Assignment 4 Long Answer

3) Let the k^{th} bit in the binary counter be denoted by A[k]

Then observe, we flip A[0] every time we do an increment

we flip A[1] every other time

we flip A[2] every 4^{th} time and so on.

The cost spent on flipping A[0] is \$1 each (a total of \$n)

A[1] gets flipped every other time

for cost of \$2 each (a total of at most \$n)

A[2] gets flipped every 4^{th} time

for cost of \$4 each (then the total cost

would be at most \$n)

and so on up to $A[\lfloor \lg n \rfloor]$,

which gets flipped once for a cost of

at most \$n.

Thus, the total cost is at most

$n(\log n + 1)$

and we have n operations.

So the amortized cost of increment is

$O(\frac{n(\log n+1)}{n})$

which is $O(\log n)$.

To find the upper and lower bounds, referring to Amortized Analysis Lec. Slide #7,

total number of flips is now

$$\sum_{i=0}^{k} \lfloor \frac{n}{2^i} \rfloor \times 2^i$$

Since the cost of flipping the bit at index i is 2^i

total cost: $\sum_{i=0}^{k} \lfloor n/2^i \rfloor \times 2^i < \sum_{i=0}^{k} \frac{n}{2^i} \times 2^i$

$< \sum_{i=0}^{k} n$

$< (k+1)n$

Let K be the largest index of bit reached in n operations,

Since we are representing the number n in this binary counter

$n = \sum_{i=0}^{K} A[i] \times 2^i$ where $A[i]$ is either 1 or 0

Thus, K is bounded

$2^K \leq n < 2^{K+1}$

$\log_2(n) - 1 < K \leq \log_2 n$

Thus, we can bound the total cost ($\sum_{0}^{K} n = (K+1)n$ we found earlier

$$n \cdot \log_2 n < \sum_{i=0}^{K} n < n(\log_2 n + 1)$$

Since we are using the aggregating method,

we want to divide the total cost of all operations by

the number of operations.

As stated in the question, we have a sequence of n operations.

Thus, the amortized cost of the function seen in class now

has an upper bound of $\frac{n(\log_2 n + 1)}{n}$,

which is $(\log_2 n + 1)$.

$$\log_2 n = \frac{\log_{10} n}{\log_{10} 2} = \frac{1}{\log_{10} 2} \cdot \log_{10} n$$

$\frac{1}{\log_{10} 2}$ is just a constant.

Thus, the amortized cost of the function increment

is now $O(\log n)$.

□

4) Since the question asks to indicate the cost of increment and reset first: the cost of reset is (2^i) and the cost of increment is $(2 \cdot 2^i)$

The cost of resetting all the bits to zero:

We have $0, 1, 2, \cdots, k'$ bits, and the cost for resetting a bit at index i is 2^i

So, the total cost is $\sum_{i=0}^{k'} 2^i$

Sum of finite geometric series:

$$S_n = \sum_{i=1}^{n} a_i r^{i-1} = a_1 \left(\frac{1-r^n}{1-r}\right)$$

In our case, the first term (a_1) is when $i=0$, $2^i = 2^0$

n, the number of terms is k' since we can only go from 0th to $(k'-1)$ th bit in the binary counter; thus, $n = k'$.

r is 2 since $\frac{a_{i+1}}{a_i} = 2$ in this series.

Thus, the total cost of resetting is $2^0 \left(\frac{1-2^{k'}}{1-2}\right)$

$$= \left(\frac{1-2^{k'}}{-1}\right)$$

$$= 2^{k'} - 1$$

Similar to the previous proof, we can bound this value by considering the maximum number of bits we need to represent a number.

We need $\lceil \log_2 n \rceil$ number of bits to represent a number n.

Thus, we can bound k', the number of bits by

$$\log_2 n - 1 < k' < \log_2 n + 1$$

Thus,

$$2^{\log_2 n - 1} - 1 < 2^{k'} - 1 < 2^{\log_2 n + 1} - 1$$

$$\frac{1}{2} \cdot n - 1 < 2^{k'} - 1 < 2n - 1$$

So the total cost is bounded by $(2n-1)$

And we have n operations, thus, dividing by the number of operations, the amortized cost of the operation reset is $O(1)$.

Now we need to use the accounting method to show that the amortized cost of the increment algorithm is $O(1)$

Let us charge $(2 \cdot 2^i)$ to set a bit to 1.

Now we need to show that:

~~1) this scheme has low total cost~~

2) we have charged enough to pay for all the operations (so the credit never goes to negative)

~~For 1),~~ each time we call increment, the function sets at most ~~one~~ bit to a 1.

So we charge $(2 \cdot 2^i)$ ~~each~~ time increment is called.

And we call increment for a total of $2^{k'}-1$ times,

~~then~~ we would charge $((2^{k'}-1)(2 \cdot 2^i))$ in total.

~~Thus, each time we double what is needed, 2^i, where~~ i is ~~the index of~~ that bit we are concerning, ~~for the increment~~.

For 2), each time we set a bit to 1, we collect $(2 \cdot 2^i)$ credits

We use (2^i) to pay for the cost of setting the bit to a 1. Then we store the (2^i) left on that bit.

When, at the end, all the bits get flipped from 1 to 0, we use that (2^i) we stored earlier on that bit to cover the cost of the flip.

Thus, what we have stored before when we were incrementing is enough for the resetting. Credit, therefore, never goes to negative

Therefore, we have shown that we can use the $(2 \cdot 2^i)$ credits each time a bit is set to a 1 to pay for all operations that flip a bit in the counter.

Also, we charge $(2 \cdot 2^i \cdot n)$ over n calls to the increment function.

Hence, the amortized cost per call to increament is $O(1)$

$O(2^i + 1) = O(2^n) = O(1)$

prove 2^i is $O(1)$:

i is bounded because we reset the binary counter when it reaches $(k-1)$ no matter how big n is.

$(k-1)$ is a constant.

Thus, 2^i is $O(1)$

Similarly, the cost for increment is $2 \cdot 2^i$ which is also bounded by $O(1)$.

In conclusion, under the conditions stated in the question, by using the accounting method, the amortized cost of the operations increment and reset is $O(1)$

THANK YOU FOR YOUR TIME!!

Collaborators:

Yuexin (Virna) Xi

Younggue Kim