Chapter 17 Solusion

https://github.com/frc123/CLRS

12/28/2021

17.1

17.1-1

No. Consider we operate MULTPUSH(S, n) n times. Such n operations cost $\Theta(n^2)$, so the amortized cost is $\Theta(n)$.

Actually, we can MULTPUSH incredible large amount of items, so $O(1)$ of course cannot be bound on the amortized cost of stack operations.

17.1-2

Consider a k-bit counter where each bit in the counter is 1. Now, we perform INCREMENT which flips $k+1$ bits. Then, we perform DECREMENT which flips $k+1$ bits again. Hence perform a sequence of length n operations $\langle$INCREMENT, DECREMENT, INCREMENT, DECREMENT, $\cdots\rangle$ cost $\Theta(nk)$ in total.

17.1-3

$$n + \sum_{i=1}^{\lfloor \lg n \rfloor} (2^i - 1) \leq n + \sum_{i=0}^{\lg n} 2^i = n + 2^{\lg n + 1} - 1 = n + 2n - 1 = 3n - 1$$

Hence the amortized cost per operation is $O(1)$.

17.2

17.2-1

operation	actual cost	amortized cost
PUSH	1	2
POP	1	2
Copy	s	0

where s is the stack size when it is called which has an upper bound k.

Each operation (PUSH or POP) charges an amortized cost of 2 and actual use 1. After k operations, we have k credits, and copy operation cost at most k. Hence we conclude the total amortized cost is greater than the total actual cost at all times.

17.2-2

Let the amortized cost of each operation be 3. We want to show that

$$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

for all integers n where

$$c_i = \begin{cases} i & \text{if } i \text{ is an exact power of 2,} \\ 1 & \text{otherwise} \end{cases}$$

and $\hat{c}_i = 3$ for all integers i. That is we want to show that

$$3n \geq n + \sum_{i=1}^{\lfloor \lg n \rfloor} (2^i - 1).$$

By exercise 17.1-3, we have

$$n + \sum_{i=1}^{\lfloor \lg n \rfloor} (2^i - 1) \leq 3n - 1.$$

Hence the amortized cost per operation is $O(1)$.

17.2-3

As the hint mentioned, we keep a pointer to the high-order 1 and maintain it during the operations. In each INCREMENT operation, we check if we the high-order 1 moved to a higher order.

Fliping a bit charges 1. Moving the pointer to the high-order 1 charges \$1. Let the amortized cost of each INCREMENT operation be \$4, and let the amortized cost of each RESET operation be \$1. When we set a bit to 1, we actually cost \$1 and retain \$2 as credits for the purpose of setting to 0 and resetting. If we need to update pointer, we charge another \$1. Hence amortized cost of each INCREMENT operation is \$4. Each RESET operation need to move the pointer to -1, so it costs \$1.

```
1  struct Counter
2  {
3      int length;
4      std::vector<bool> bits;
5      int high_order_one;
6
7      Counter(int length) : length(length),
8          bits(length, 0), high_order_one(-1) {}
9  };
10
11 void Increment(Counter& counter)
```

```
{
    int i;
    i = 0;
    while (i < counter.length && counter.bits[i] == 1)
    {
        counter.bits[i] = 0;
        ++i;
    }
    if (i < counter.length)
    {
        counter.bits[i] = 1;
        counter.high_order_one = std::max(i, counter.high_order_one);
    }
    else
    {
        // overflow
        counter.high_order_one = -1;
    }
}

void Reset(Counter& counter)
{
    int i;
    for (i = 0; i < counter.length; ++i)
    {
        counter.bits[i] = 0;
    }
    counter.high_order_one = -1;
}
```

Updating...