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 $\langle \text{INCREMENT}, \text{DECREMENT}, \text{INCREMENT}, \text{DECREMENT}, \cdots \rangle$ cost $\Theta(nk)$ in total.

17.1-3

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

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.

```cpp
struct Counter
{
    int length;
    std::vector<bool> bits;
    int high_order_one;

    Counter(int length) : length(length),
        bits(length, 0), high_order_one(-1) {}
};
```

```

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;
}
```

17.3

17.3-1

Let $\Phi'(D_i) = \Phi(D_i) - \Phi(D_0)$. Clearly, $\Phi'(D_0) = 0$. We claim the amortized costs using Φ' are the same as the amortized costs using Φ.

$$\begin{aligned}\hat{c_i} &= c_i + \Phi'(D_i) - \Phi(D_{i-1}) \\ &= c_i + (\Phi(D_i) - \Phi(D_0)) - (\Phi(D_{i-1}) - \Phi(D_0)) \\ &= c_i + \Phi(D_i) - \Phi(D_{i-1})\end{aligned}$$

17.3-2

Let $\Phi(D_0) = 0$ and $\Phi(D_i) = 2(i - 2^{\lfloor \lg i \rfloor})$ for $i \geq 1$.

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= 2(i - 2^{\lfloor \lg i \rfloor}) - 2((i-1) - 2^{\lfloor \lg (i-1) \rfloor}) \\ &= 2 - 2(2^{\lfloor \lg i \rfloor} - 2^{\lfloor \lg (i-1) \rfloor})\end{aligned}$$

Note that

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

Case 1. i is an exact power of 2.

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= 2 - 2(i - \frac{i}{2}) \\ &= 2 - i\end{aligned}$$

$$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= i + 2 - i \\ &= 2\end{aligned}$$

Case 2. i is not an exact power of 2.
Then $2^{\lfloor \lg i \rfloor} = 2^{\lfloor \lg (i-1) \rfloor}$.

$$\Phi(D_i) - \Phi(D_{i-1}) = 2$$

$$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 2 \\ &= 3\end{aligned}$$

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

17.3-3

The idea is to let the potential be proportional to the sum of the height of every node in the min-heap. Note that an binary heap is a complete binary tree.

$$\sum_{j=1}^{n} \lfloor \lg j \rfloor \leq \lg(n!) \leq n \lg n$$

Let Φ be

$$\Phi(D_i) = \begin{cases} 0 & \text{if } n_i = 0, \\ k n_i \lg n_i & \text{if } n_i > 0 \end{cases}$$

for some constant k where n_i is the number of nodes in D_i. Also, we have

$$c_i \leq \begin{cases} k_1 \lg n_i & \text{if I}\textsc{nsert} \text{ is performed in the } i\text{th operation and } n_i \geq 2, \\ k_2 \lg n_{i-1} & \text{if E}\textsc{xtract}\text{-M}\textsc{in} \text{ is performed in the } i\text{th operation and } n_{i-1} \geq 2 \end{cases}$$

Let $k = \max(k_1, k_2)$.

Case 1. INSERT is performed in the ith operation. Then $n_i - 1 = n_{i-1}$.
If $n_i = 1$,

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i \end{aligned}$$

If $n_i \geq 2$,

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k \lg n_i + k n_i \lg n_i - k n_{i-1} \lg n_{i-1} \\ &= k(\lg n_i + n_i \lg n_i - n_{i-1} \lg n_{i-1}) \\ &= k(\lg n_i + n_i \lg n_i - (n_i - 1) \lg(n_i - 1)) \\ &= k(\lg n_i + n_i \lg n_i - n_i \lg(n_i - 1) + \lg(n_i - 1)) \\ &< k(2 \lg n_i + n_i(\lg n_i - \lg(n_i - 1))) \end{aligned}$$

Note that $\forall x \in \mathbb{R}$, $1 + x \leq e^x$. Then

$$\begin{aligned} n_i(\lg n_i - \lg(n_i - 1)) &= n_i \lg \frac{n_i}{n_i - 1} \\ &= n_i \lg(1 + \frac{1}{n_i - 1}) \\ &\leq n_i \lg(e^{\frac{1}{n_i - 1}}) \\ &= \frac{n_i}{n_i - 1} \lg e \\ &= (1 + \frac{1}{n_i - 1}) \lg e \\ &\leq 2 \lg e \end{aligned}$$

Hence

$$\hat{c_i} < k(2 \lg n_i + 2 \lg e)$$

We conclude $\hat{c_i} = O(\lg n)$ for INSERT.

Case 2. EXTRACT-MIN is performed in the ith operation. Then $n_{i-1} - 1 = n_i$.
If $n_{i-1} = 1$,

$$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i\end{aligned}$$

If $n_{i-1} \geq 2$,

$$\begin{aligned}\hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq k \lg n_{i-1} + kn_i \lg n_i - kn_{i-1} \lg n_{i-1} \\ &= k(\lg n_{i-1} + n_i \lg n_i - n_{i-1} \lg n_{i-1}) \\ &= k(\lg n_{i-1} + (n_{i-1} - 1) \lg(n_{i-1} - 1) - n_{i-1} \lg n_{i-1}) \\ &< k(\lg n_{i-1} - \lg(n_{i-1} - 1)) \\ &= k \lg(1 + \frac{1}{n_{i-1} - 1}) \\ &\leq k \lg e^{\frac{1}{n_{i-1}-1}} \\ &= \frac{k}{n_{i-1} - 1} \lg e\end{aligned}$$

We conclude $\hat{c_i} = O(1)$ for EXTRACT-MIN.

17.3-4

$$\Phi(D_n) - \Phi(D_0) = s_n - s_0$$

Since $\hat{c_i} = 2$,

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{i=1}^{n} \hat{c_i} - \Phi(D_n) + \Phi(D_0) \\ &= 2n + s_0 - s_n\end{aligned}$$

17.3-5

$$\Phi(D_0) = b$$

Since $\hat{c_i} \leq 2$,

$$\begin{aligned}\sum_{i=1}^{n} c_i &= \sum_{i=1}^{n} \hat{c_i} - \Phi(D_n) + \Phi(D_0) \\ &\leq 2n + b - \Phi(D_n)\end{aligned}$$

Since $\Phi(D_n) \geq 0$,

$$\sum_{i=1}^{n} c_i \leq 2n + b$$

Since $n = \Omega(b)$,

$$\sum_{i=1}^{n} c_i = O(n)$$

17.3-6

```
template <typename T>
class Queue
{
public:
    void Enqueue(T& x);
    void Enqueue(T&& x);
    T Dequeue();
private:
    std::stack<T> s_a_;
    std::stack<T> s_b_;
};

template <typename T>
void Queue<T>::Enqueue(T& x)
{
    s_a_.push(x);
}

template <typename T>
void Queue<T>::Enqueue(T&& x)
{
    s_a_.emplace(std::move(x));
}

template <typename T>
T Queue<T>::Dequeue()
{
    if (s_b_.empty())
    {
        while (s_a_.empty() == false)
        {
            s_b_.emplace(std::move(s_a_.top()));
            s_a_.pop();
        }
    }
```

```
        T top = std::move(s_b_.top());
        s_b_.pop();
        return std::move(top);
    }
```

Assume each of $s\_a\_.push$ (or $emplace$), $s\_a\_.pop$, $s\_b\_.push$ (or $emplace$), $s\_b\_.pop$ costs \$1. Then

$$c_i = \begin{cases} 1 & \text{if } \textsc{Enqueue} \text{ is performed in the } i\text{th operation,} \\ 1 & \text{if } \textsc{Dequeue} \text{ is performed in the } i\text{th operation and } D_{i-1}.s\_b\_ \text{ is not empty,} \\ 2 \cdot (D_{i-1}.s\_a\_.size()) + 1 & \text{if } \textsc{Dequeue} \text{ is performed in the } i\text{th operation and } D_{i-1}.s\_b\_ \text{ is empty} \end{cases}$$

Let

$$\Phi(D_i) = 3 \cdot (D_i.s\_a\_.size()) + (D_i.s\_b\_.size())$$

Case 1. ENQUEUE is performed in the ith operation.

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 3 \cdot (D_i.s\_a\_.size() - D_{i-1}.s\_a\_.size()) + (D_i.s\_b\_.size() - D_{i-1}.s\_b\_.size()) \\ &= 1 + 3 \cdot 1 + 0 \\ &= 4 \end{aligned}$$

Case 2. DEQUEUE is performed in the ith operation and $D_{i-1}.s\_b\_$ is not empty.

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 3 \cdot (D_i.s\_a\_.size() - D_{i-1}.s\_a\_.size()) + (D_i.s\_b\_.size() - D_{i-1}.s\_b\_.size()) \\ &= 1 + 3 \cdot 0 - 1 \\ &= 0 \end{aligned}$$

Case 3. DEQUEUE is performed in the ith operation and $D_{i-1}.s\_b\_$ is empty.

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= (2 \cdot (D_{i-1}.s\_a\_.size()) + 1) + 3 \cdot (D_i.s\_a\_.size() - D_{i-1}.s\_a\_.size()) + (D_i.s\_b\_.size() - D_{i-1}.s\_b\_.size()) \\ &= (2 \cdot (D_{i-1}.s\_a\_.size()) + 1) - 3 \cdot (D_{i-1}.s\_a\_.size()) + (D_{i-1}.s\_a\_.size() - 1) \\ &= 0 \end{aligned}$$

Thus, we conclude that the amortized cost of each ENQUEUE and each DEQUEUE operation is $O(1)$.

17.3-7

Note that section 9.3 provides an approach of selection in worst-case linear time.

```cpp
class DataStructure
{
public:
    void Insert(int x);
    void DeleteLargerHalf();
    const std::vector<int>& Get() const;
private:
    std::vector<int> arr_;
};

void DataStructure::Insert(int x)
{
    arr_.push_back(x);
}

void DataStructure::DeleteLargerHalf()
{
    size_t median = arr_.size() >> 1;
    LinearSelect(arr_, 0, arr_.size() - 1, (arr_.size() - 1) >> 1);
    arr_.erase(arr_.begin() + median, arr_.end());
}

const std::vector<int>& DataStructure::Get() const
{
    return arr_;
}
```

Assume

$$c_i = \begin{cases} 1 & \text{if INSERT is performed in the } i\text{th operation,} \\ n_{i-1} & \text{if DELETE-LARGER-HALF is performed in the } i\text{th operation} \end{cases}$$

where n_i is $|S|$ after the ith operation. Let

$$\Phi(D_i) = 2n_i$$

be the potential function of the data structure.

Case 1. INSERT is performed in the ith operation.

$$\begin{aligned} \hat{c_i} &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + 2(n_i - n_{i-1}) \\ &= 1 + 2 \cdot 1 \\ &= 3 \end{aligned}$$

Case 2. DELETE-LARGER-HALF is performed in the ith operation.

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= c_i + 2(n_i - n_{i-1}) \\ &= n_{i-1} + 2(\frac{n_{i-1}}{2} - n_{i-1}) \\ &= n_{i-1} - n_{i-1} \\ &= 0\end{aligned}$$

Updating...