The First Homework | 第一次作业

ピカピカピ

2021 年 11 月 3 日

1 Different-sized sub-problems

$$\begin{aligned} T(n) &= T(n/2) + 2T(n/8) + n \\ &\leq c \cdot n/2 + 2c \cdot n/8 + n \\ &= (\frac{3c}{4} + 1) \cdot n \\ &\leq cn \qquad\qquad \text{when } c \geq 1 \end{aligned}$$

因而 $T(n) = O(n)$

2 Asymptotic bound with Maximum operator

由于 $f(n)$ 和 $g(n)$ 都是渐进非负函数, 根据定义有:
$\exists\, N_f,\ N_g$ 使得:

$$\begin{aligned} \text{当 } n > N_f \text{ 时 有 } f(n) \geq 0 \\ \text{同时, 当 } n > N_g \text{ 时 有 } g(n) \geq 0 \end{aligned}$$

取 $N_0 = max(f(n),\ g(n))$
当 $n > N_0$ 时, 有:

$$\text{当 } n > N_0 \text{ 时 有 } f(n) \geq 0,\ g(n) \geq 0$$

另一方面, 当 $n > N_0$ 时, 显然有:

$$\begin{aligned} f(n) &\leq max(f(n),\ g(n)) \\ g(n) &\leq max(f(n),\ g(n)) \\ (f(n) + g(n))/2 &\leq max(f(n),\ g(n)) \\ max(f(n),\ g(n)) &\leq f(n) + g(n) \end{aligned}$$

因而当 $n > N_0$ 时, 可以得到:

$$0 \leq (f(n) + g(n))/2 \leq max(f(n), g(n)) \leq f(n) + g(n)$$

即:

$$max(f(n), g(n)) = \Theta(f(n) + g(n))$$

3 Proof of correctness

对于每一次的比较操作而言, 若 $a_i \leq a_j$, 那么 a_i, a_j 保持不变, 于是有 $a_i \leq a_j$; 若 $a_i \geq a_j$, 那么 a_i, a_j 交换, 于是有 $a_i \leq a_j$. 换言之, 进行过一次比较交换操作后, 恒有 $a_i \leq a_j$.
证明如下:
对于每一次以下标 i 开始的循环, 对 $j - i$ 作数学归纳法.
进行了第 $j - i = 1$ 次比较后, 由上面结论知, $a_i \leq aj$. 所以 a_i 是 a_i, ..., a_j 中最小的数.
假设当 $j - i = k$ 时“a_i 是 a_i, ..., a_j 中最小的数”成立, 则当 $j - i = k + 1$ 时. $a_i \leq a_j$, 于是 a_j 不小于 a_i, 根据假设, 即 a_j 不小于 a_i, ..., a_{j-1} 中最小的数. 于是 a_i, ..., a_{j-1} 中最小的数 a_i 仍是 a_i, ..., a_j 中最小的数.
由数学归纳法原理可知, a_i 是 a_i, ..., a_n 中最小的数.
完成排序后, 可知:

$$\begin{aligned} a_1 &= min(a_1, ..., a_n), \\ a_2 &= min(a_2, ..., a_n), \\ &..., \\ a_{n-1} &= min(a_{n-1}, a_n) \end{aligned}$$

因为 $max(a_1, ..., a_n) \leq max(a_2, ..., a_n) \leq ... \leq max(a_{n-1}, a_n) \leq a_n$, 所以有 $a_1 \leq a_2 \leq ... \leq a_n$. 证毕.

4 COVID-19 Risk Detection

4.1 (a)

见 Algorithm 1.

4.2 (b)

见 Algorithm 2.

正如算法所示, 由于算法内部有一个两层的 for 循环遍历, 易知其时间复杂度为 $\Theta(n^2)$.

Algorithm 1 O(n) retrieve customers

```
1: procedure O(n)RETRIEVE(c_0)                          ▷ c_0 代表给定的顾客
2:     c_0_leavtime ← c_0's Leave Time
3:     for c_i in Customer do                           ▷ c_i 代表顾客列表中除 c_0 外的顾客
4:         c_i_enttime ← c_i's Enter Time
5:         c_i_leavtime ← c_i's Leave Time
6:         if c_i_enttime ≤ c_0_leavtime ≤ c_i_leavtime then
7:             Print c_i
```

Algorithm 2 $O(n^2)$ retrieve customers

```
1: procedure O(n^2)RETRIEVE(c_list)                     ▷ c_list 代表顾客列表
2:     cnt = 0
3:     for c_i in c_list do                             ▷ c_i 代表顾客列表中每个顾客
4:         c_i_leavtime ← c_i's Leave Time
5:         for c_j in c_list do                         ▷ c_j 代表顾客列表中除 c_i 外的顾客
6:             c_j_enttime ← c_j's Enter Time
7:             c_j_leavtime ← c_j's Leave Time
8:             if c_j_enttime ≤ c_i_leavtime ≤ c_j_leavtime then
9:                 cnt ++
10:    Print cnt
```

4.3 (c)

见 Algorithm 3.
由于算法中仅使用 merge sort 与一层循环遍历, 因而其时间复杂度为:

$$T(n) \ = \ T(2n) + O(nlgn)$$

即其时间复杂度为 $O(nlgn)$

5 SELECT-7 algorithm

5.1 (a)

对小组被以 7 为个数划分的 SELECT-7 算法而言, 就代码编写上, SELECT-7 部分的伪代码不需要进行改动, MEDIAN_OF_MEDIANS-7 部分的伪代码由于每组现在包含了 7 个元素, 因而在求每组中位数时算法内部的计算数值需要做出改动, 同时求中位数的中位数时数值也需要进行改动
换言之，应以 7 为循环的步长来执行寻找每组中位数的循环, 同时调用的 partition7 函数内部也应进行重写

Algorithm 3 $O(nlgn)$ retrieve customers

procedure MERGESORT(A, left, right) ▷ A 是数组, left, right 是下标]
  if $left \geq right$ **then**
    return A[left..right]
  $mid \leftarrow \lfloor \frac{left+right}{2} \rfloor$
  MERGESORT (A,left,mid)
  MERGESORT (A,mid+1,right)
  MERGE (A,left,mid,right)
  return A[left..right]
procedure MERGE(A, left, mid, right) ▷ A 是数组, left, right 是下标]
  $A'[left..right] \leftarrow A[left..right]$
  $i \leftarrow left$
  $j \leftarrow mid + 1$
  $k \leftarrow 0$
  while $i \leq mid$ *and* $j \leq right$ **do**
    if $A'[i] \leq A'[j]$ **then**
      $A[left + k] \leftarrow A'[i]$
      $k++,\ i++$
    else
      $A[left + k] \leftarrow A'[j]$
      $k++,\ j++$
  if $i \leq mid$ **then**
    $A[left + k..right] \leftarrow A'[i..mid]$
  else
    $A[left + k..right] \leftarrow A'[j..right]$
  return A[left..right]
procedure $O(nlgn)$RETRIEVE(c_list) ▷ c_list 代表顾客列表
  cnt = 0
  presentc_list = 0
  timelist.add(EnterTimeList)
  timelist.add(LeaveTimeList) ▷ 三个时间列表里的时间都以结构体中的一部分为形式存储, 即可以通过该时间查询到与时间绑定的顾客以及该时间是进店时间或离店时间
  MERGESORT(timelist, 0, timelist.size()-1)
  for t in timelist **do**
    if t 是进店时间 **then**
      presentc_list ++ ▷ presentc_list 存储了目前在店的顾客数量
      cnt += presentc_list.size()
    else ▷ 即如果 t 是离店时间
      presentc_list –
  return cnt

5.2 (b)

根据题设, 可以得知原数组被划分为 $\lceil n/7 \rceil$ 个小组, 因而一共有 $\lceil n/7 \rceil$ 个中位数, 根据题设 n 为 7 的倍数, 因而 MEDIAN_OF_MEDIANS-7 内部对于 SELECT-7 的调用基于的数组大小最多为 $n/7$

5.3 (c)

对 MEDIAN_OF_MEDIANS-7 而言, 根据题设 n 为 7 的倍数, 则根据被选出的 pivot, 令 $m = \lceil n/7 \rceil$, 则比 pivot 小的数至少有 $4(\lceil m/2 \rceil - 1) + 3$ 个; 比 pivot 大的数至多有 $n - 1 - (4(\lceil m/2 \rceil - 1) + 3)$ 个
由于:

$$\begin{aligned} & n - 1 - (4(\lceil m/2 \rceil - 1) + 3) \\ = \ & n - 4 - 4(\lceil m/2 \rceil - 1) \\ = \ & n - 4\lceil m/2 \rceil \\ \leq \ & n - 2n/7 + 7 \\ = \ & 5n/7 + 7 \end{aligned}$$

因而可以得知, SELECT-7 内部的递归调用所基于的数组大小至多为 $5n/7 + 7$

5.4 (d)

在对 SELECT-7 和 MEDIAN_OF_MEDIANS-7 的一次调用过程中, 对大小为 n 的数组仅仅执行了几次的循环遍历操作, 不涉及到递归调用和嵌套循环等操作, 因而其时间复杂度是 $\Theta(n)$

5.5 (e)

$$T(n) \leq T(n/7) + T(5n/7) + O(n)$$

5.6 (f)

$$\begin{aligned} T(n) & \leq T(n/7) + T(5n/7) + O(n) \\ & \leq c \cdot n/7 + c \cdot 5n/7 + dn \\ & = 6/7 \cdot cn + dn \end{aligned}$$

令 $c = 7d$, 有 $T(n) \leq cn$
即 SELECT-7 的时间复杂度为 $O(n)$

6 Find out honest monkeys

6.1 (a)

见 Algorithm 4.

Algorithm 4 Reduce Size

```
1: procedure ReduceSize(mlist)                          ▷ mlist 是存储 Monkey 类型的环形链表
2:     n ← mlist.size()
3:     for i from 0 to (n-2) by 2 do                    ▷ 做 n/2 次 monkey-to-monkey comparisons
4:         m1 = mlist.remove()                          ▷ m1 是 mlist 的第一个元素, 此元素从链表移除
5:         m2 = mlist.remove()      ▷ m2 是 mlist 的第一个元素 (原先是第二个), 此元素从链表移除
6:         if m1 says m2 honest and m2 says m1 honest then
7:             mlist.add(m1)                            ▷ 把 m1 加到链表尾部
8:     return mlist
```

6.2 (b)

见 Algorithm 5.

Algorithm 5 Find Honest

```
1: procedure FindHonest(mlist)                          ▷ mlist 是存储 Monkey 类型的环形链表
2:     originmlist ← mlist
3:     while mlist.size() > 2 do
4:         REDUCESIZE(mlist)
5:     if mlist.size() == 2 then
6:         m1 = mlist.remove()                          ▷ m1 是 mlist 的第一个元素, 此元素从链表移除
7:         m2 = mlist.remove()      ▷ m2 是 mlist 的第一个元素 (原先是第二个), 此元素从链表移除
8:         count ← 0
9:         for mi in originmlist do
10:            if mi says m1 honest then
11:                count ++
12:                if count ≥ originmlist.size()/2 then
13:                    return m2
14:                else
15:                    return m1
16:            else
17:                return mlist.remove()
```

6.3 (c)

1) 当环形链表中有 1 个元素时, 显然可以找出诚实的猴子;
当环形链表中有 2 个元素时, 根据算法将全部猴子与其中一只一一进行配对, 根据其他猴子对该猴子的评价数目便可以判断找出诚实的猴子以 hu 表示猴子的诚实或不可靠

2) 以 hu 表示猴子的诚实或不可靠
假若根据算法, 当环形链表中有 $k-1$ 只猴子时可以找出诚实的猴子, 假设其 hu 属性列表为 hu_i, hu_{i+1}, ..., hu_{i+k-2};
当环形链表中有 k 只猴子时亦可以找出诚实的猴子, 假设其 hu 属性列表为 hu_{i-1}, hu_i, ..., hu_{i+k-2}

3) 那么当环形链表中有 $k+1$ 只猴子时, 假设其 hu 列表为 hu_{i-2}, hu_{i-1}, ..., hu_{i+k-2}, 根据算法对环形链表的前两项进行判断, 即对 hu_{i-2}, hu_{i-1} 进行判断
若两只猴子都是诚实的, 则会得到都诚实的答案，那么随便选一只加入列表尾部并抛弃另一只，可以得到新列表, 其 hu 列表为 hu_i, hu_{i+1}, ..., hu_{i+k-2}, hu_{i-1} 或 hu_i, hu_{i+1}, ..., hu_{i+k-2}, hu_{i-2}, 其共 $k-1$ 项;
若两只猴子有一只不可靠的, 那么至少一只猴子会说对方是不可靠的, 那么抛弃两只猴子, 可以得到新列表, 其 hu 列表为 hu_i, hu_{i+1}, ..., hu_{i+k-2}, 其共 k 项;
若两只猴子都是不可靠的, 若得到都诚实的答案，那么随便选一只加入列表尾部并抛弃另一只，可以得到新列表, 其 hu 列表为 hu_i, hu_{i+1}, ..., hu_{i+k-2}, hu_{i-1} 或 hu_i, hu_{i+1}, ..., hu_{i+k-2}, hu_{i-2}, 其共 $k-1$ 项; 否则全部丢弃, 可以得到新列表, 其 hu 列表为 hu_i, hu_{i+1}, ..., hu_{i+k-2}, 其共 k 项
同时我们注意到, 由于诚实的猴子多于不可靠的猴子, 在我们进行抛弃猴子的过程中, 抛弃的不可靠的猴子总是比诚实的猴子多, 因而可以保证最终总有至少一个诚实的猴子在链表中
根据 2) 我们可以得知, 当环形链表含有 k+1 只猴子时可以由算法推演转化为含有 k 或 k-1 只猴子的情况, 因而可以找到诚实的猴子

4) 由此我们得以证明算法的正确性

6.4 (d)

见 Algorithm 6.

Algorithm 6 Find Honests

```
1: procedure FindHonests(mlist)                          ▷ mlist 是存储 Monkey 类型的环形链表
2:     originmlist ← mlist
3:     while mlist.size() > 2 do
4:         n ← mlist.size()
5:         m1 = mlist.remove()                           ▷ m1 是 mlist 的第一个元素, 此元素从链表移除
6:         m2 = mlist.remove()      ▷ m2 是 mlist 的第一个元素 (原先是第二个), 此元素从链表移除
7:         if m1 says m2 honest and m2 says m1 honest then
8:             mlist.add(m1)                                          ▷ 把 m1 加到链表尾部
9:     if mlist.size() == 2 then
10:        m1 = mlist.remove()                           ▷ m1 是 mlist 的第一个元素, 此元素从链表移除
11:        m2 = mlist.remove()      ▷ m2 是 mlist 的第一个元素 (原先是第二个), 此元素从链表移除
12:        count ← 0
13:        for mi in originmlist do
14:            if mi says m1 honest then
15:                count ++
16:                if count ≥ originmlist.size()/2 then
17:                    honest0 = m2
18:                else
19:                    honest0 = m1
20:            else
21:                honest0 = mlist.remove()
22:    honestlist.add(honest0)
23:    for mi in originmlist except honest0 do
24:        if honest0 says mi honest then
25:            honestlist.add(mi)
26:    return honestlist
```