

# 《算法设计与分析》

## 第三次作业

姓名：曹建钦

学号：20375177

# 1 最大空位问题

## 1.1 Main Idea

长度为  $n$  的 01 串  $S = \langle s_1, s_2, \dots, s_n \rangle$ , 仅有一次机会挑选出其中两个元素  $s_i, s_j (1 \leq i, j \leq n)$  并交换他们的位置, 目标是得到交换之后  $S$  中连续 0 的最大个数。有效的交换必然为 0 和 1 的互换, 则所挑选的两个元素  $s_i, s_j$  互异, 最长的空位一定为交换排列最稀疏的三个 1 的中间的 1 和两边的 1 以外的 0 得到的, 故使用贪心算法。

**贪心策略为:** 考虑到存在类似 00010000 的两端有连续 0 的情况, 在串  $S$  的两端  $s_0, s_{n+1}$  补虚拟的 1, 但其不能与 0 进行交换, 只起到固定边界的作用, 即将 00010000 补为 (1)00010000(1)。而对于类似 (1)00000000(1) 的仅由 0 构成的情况, 即遍历  $S$  后未找到第三个 1, 连续 0 的最大个数便为  $n$ 。

对于一般情况, 从左往右遍历串  $S$ , 以三个 1 为一组  $\langle s_i, s_k, s_j \rangle (0 \leq i < k < j \leq n+1)$  向右推进, 并记录  $s_i$  左侧 0 的个数, 找到  $\max\{j-i\}$ , 此即为  $S$  中向前交换 0 和 1 后可得 0 的最大个数, 再次从右往左遍历串  $S$ , 以三个 1 为一组  $\langle s_i, s_k, s_j \rangle (0 \leq i < k < j \leq n+1)$  向左推进, 并记录  $s_j$  右侧 0 的个数, 找到  $\max\{j-i\}$ , 此即为  $S$  中向后交换 0 和 1 后可得 0 的最大个数, 遍历两遍得到的两个  $\max\{j-i\}$  的最大值即任意交换后连续 0 的最大个数。

算法的伪代码如下:

## 1.2 Pseudo Code

---

**Algorithm 1:** *Maximum – Contiguous – Zero( $S, n$ )*

---

**Input:** 01 串  $S$ :  $\langle s_1, s_2, \dots, s_n \rangle$ , 长度为  $n$

**Output:** 一次交换之后  $S$  中最多的连续 0 的个数  $sum$

```
1 新建空队列  $Q_1, Q_2$  ;
2  $sum \leftarrow 0$  ;
3 //  $S$  首尾补 1
4  $S[0] \leftarrow 1$ ;
5  $S[n+1] \leftarrow 1$  ;
6 // 从左往右遍历串  $S$ 
7  $leftCountZero \leftarrow 0$  ;
8 for  $i \leftarrow 0$  to  $n+1$  do
9   if  $S[i] == 1$  then
10      $Q_1.Enqueue(i)$ ;
11     if  $len(Q_1) == 3$  then
12       //  $count$  为三个 1 中左侧 1 的左侧 0 的个数
13        $count \leftarrow leftCountZero - (Q_1[2] - Q_1[0] - 2)$ ;
14        $sum \leftarrow \max\{maxZeroInThreeOne(Q_1[0], Q_1[1], Q_1[2], count), sum\}$  ;
15        $Q_1.Dequeue()$  ;
16     end
17   else
18      $leftCountZero \leftarrow leftCountZero + 1$ ;
19   end
20 // 从右往左遍历串  $S$ 
21  $rightCountZero \leftarrow 0$  ;
22 for  $j \leftarrow n+1$  to 0 do
23   if  $S[j] == 1$  then
24      $Q_2.Enqueue(j)$ ;
25     if  $len(Q_2) == 3$  then
26        $count \leftarrow rightCountZero - (Q_2[0] - Q_2[2] - 2)$ ;
27        $sum \leftarrow \max\{maxZeroInThreeOne(Q_2[2], Q_2[1], Q_2[0], count), sum\}$  ;
28        $Q_2.Dequeue()$  ;
29     end
30   else
31      $rightCountZero \leftarrow rightCountZero + 1$ ;
32   end
33 return  $sum$ 
```

---

---

**Algorithm 2:** *maxZeroInThreeOne(left, middle, right, count)*

---

**Input:** 3 个连续 1 的下标  $left, middle, right$ 。count 为串  $S$  中下标小于  $left$  元素中 0 的个数或下标大于  $right$  元素中 0 的个数

**Output:** 一次交换之后,  $S$  从  $s_{left}$  到  $s_{right}$  中连续 0 的最大个数

```
1 // 初始化, 连续 0 的最大个数至少为  $right - left - 2$ 
2  $length \leftarrow right - left - 2$ ;
3 if  $count > 0$  then
4   |  $length \leftarrow right - left - 1$ 
5 end
6 return  $length$ 
```

---

### 1.3 Complexity Analysis

算法的时间复杂度  $T(n)$  主要来自于 *Maximum-Contiguous-Zero( $S, n$ )* 算法中 8-19 行和 22-32 行的两次循环遍历, 以比较为基本运算, 有

$$T(n) = O(n)$$

## 2 最大收益问题

### 2.1 Main Idea

某公司有一台机器, 在每天结束时, 该机器产出的受益为  $X_1$  元, 每天开始时, 若当前剩余资金大于等于  $U$  元, 则可支付  $U$  元来升级该机器 (每天最多只能升级一次)。从升级之日开始, 该机器每天多产出  $X_2$  元的收益, 公司初始基金为  $C$  元, 想得到  $n$  天之后拥有总资金的最大值。对于第  $i(1 \leq i \leq n)$  天开始时来说, 如果当前剩余资金数大于等于  $U$  元, 升级机器花费  $U$  元, 而到第  $n$  天相比不升级机器多赚  $X_2 \times (n - i + 1)$  元, 只要  $X_2 \times (n - i + 1) \geq U$ , 那么升级机器就是不亏的, 使用贪心算法:

**贪心策略为:** 如果当天资金足够升级机器, 且所带来的后续收益不小于成本  $U$ , 则升级机器, 否则不升级。

算法的伪代码如下:

## 2.2 Pseudo Code

---

**Algorithm 3:** *mostFunds*( $C, n$ )

---

**Input:** 初始资金  $C$  元, 天数  $n$

**Output:**  $n$  天之后该公司拥有的总资金的最大值

```
1 // 初始化: 第 0 天结束后总资金最大值  $MF$  为  $C$ 
2  $MF \leftarrow C$ ;
3 //  $K$  代表第  $i$  天已经进行了  $K$  次升级
4  $K \leftarrow 0$ ;
5 for  $i \leftarrow 1$  to  $n$  do
6   if  $MF \geq U$  and  $X_2 \times (n - i + 1) \geq U$  then
7      $K \leftarrow K + 1$ ;
8      $MF \leftarrow MF - U + X_1 + K \times X_2$ ;
9   else
10     $MF \leftarrow MF + X_1 + K \times X_2$ ;
11  end
12 end
13 return  $MF$ 
```

---

## 2.3 Complexity Analysis

令算法的时间复杂度为  $T(n)$ , 以比较判断为基本运算, 从伪代码中可以看出, 复杂度集中于 5-12 行的循环, 则有:

$$T(n) = O(n)$$

# 3 探险家分组问题

## 3.1 Main Idea

营地中共有  $n$  个探险家, 第  $i$  个探险家的经验值为  $e_i (1 \leq i \leq n)$ , 优化目标为组建的队伍越多越好。

**贪心策略为:** 经验值最少的探险家优先, 且每支队伍越短越好。

算法的伪代码如下:

### 3.2 Pseudo Code

---

**Algorithm 4:** *largestNumberOfTeam( $e, n$ )*

---

**Input:**  $n$  为营地探险家的数量,  $e$  为  $n$  个探险家各自的经验值构成的序列  $\{e_1, e_2, \dots, e_n\}$

**Output:** 组建的队伍的最大数量  $num$

```
1 将序列  $e$  从小到大进行排序 ;
2  $num \leftarrow 0$  ;
3  $needPerson \leftarrow e[1]$  ;
4  $nowPerson \leftarrow 0$ ;
5 for  $i \leftarrow 1$  to  $n$  do
6    $nowPerson \leftarrow nowPerson + 1$ ;
7   if  $e[i] > needPerson$  then
8      $needPerson \leftarrow e[i]$ 
9   end
10  if  $needPerson == nowPerson$  then
11     $num \leftarrow num + 1$  ;
12    if  $i < n$  then
13       $needPerson \leftarrow e[i + 1]$  ;
14       $nowPerson \leftarrow 0$  ;
15    end
16  end
17 end
18 return  $num$ 
```

---

### 3.3 Complexity Analysis

以  $T(n)$  表示算法的时间复杂度,  $n$  为探险家数量, 以比较判断为基本运算, 第 1 行的排序复杂度为  $T_1(n) = O(n \log n)$ , 在 5-17 行的循环体中, 每一次循环需要进行最多三次判断, 有  $T_2(n) = O(n)$ 。综上可得:

$$T(n) = O(n \log n) + O(n) = O(n \log n)$$

## 4 分店选址问题

### 4.1 Main Idea

调研产生了  $n$  个备选地址，并实地考察到了两组数据  $flow$  和  $cost$ ，其中  $flow[i]$  表示第  $i(1 \leq i \leq n)$  个备选地址的人流量， $cost[i]$  表示在该地址开店所需的最低资金。目标是从  $n$  个备选地址中挑选出  $k(1 \leq k \leq n)$  个组成最终分店名单，使得能够在满足以下约束条件的前提下尽可能降低总投资成本：

- 对每个被选中的地址，应当按照其人流量与其他  $k-1$  个被选中地址人流量的比例投入资金
- 被选中的每个地址的投入资金都不得低于其所需的最低资金

假设选择的  $k$  个备选地址为： $i, i+1, \dots, i+k-1$ 。设编号为  $max$  的地址的  $cost/flow$  最大，根据上面的约束条件可以得到：对于这个方案而言，总投资成本为

$$\frac{cost[max]}{flow[max]} \times \sum_{j=i}^{j=i+k-1} flow[j]$$

受此公式的启发，要想得到最小的总投资成本，设计使用贪心算法。

**贪心策略为：** $cost/flow$  最小者优先，其次  $\sum_{j=i}^{j=i+k-1} flow[j]$  最小者优先，最终目标为两式的乘积最小。

伪代码如下：

## 4.2 Pseudo Code

---

**Algorithm 5:**  $\text{minCost}(\text{flow}, \text{cost}, n, k)$

---

**Input:**  $n$  个备选地址,  $n$  个备选地址的人流量数组  $\text{flow}$ ,  $n$  个备选地址的所需最低资金数组  $\text{cost}$ , 最终分店名单中所选地址的个数为  $k$

**Output:** 最小的总投资成本  $\text{minCost}$

```
1 // 新建一个数组  $\text{divide}$ , 存储  $\text{cost}[i]/\text{flow}[i]$ 
2 for  $i \leftarrow 1$  to  $n$  do
3    $\text{divide}[i] \leftarrow \text{cost}[i]/\text{flow}[i]$ ;
4 end
5  $\text{flow}[1 \dots n]$  按照  $\text{divide}[i]$  的大小进行升序排序;
6  $\text{divide}[1 \dots n]$  进行升序排序;
7 //  $\text{minCost}$  代表最小总投资成本,  $\text{sum}$  代表  $\text{flow}$  前  $j$  个元素
   中最小的  $k$  个元素的和, 为了方便找到前  $k$  个最小元素, 这
   里使用优先队列 (大顶堆), 并下面进行初始化
8 用  $\text{flow}[1 \dots k]$  构建优先队列  $Q$ ;
9  $\text{sum} \leftarrow \text{flow}[1] + \dots + \text{flow}[k]$ ;
10  $\text{minCost} \leftarrow \text{sum} \times \text{divide}[k]$ ;
11 for  $j \leftarrow k + 1$  to  $n$  do
12   if  $\text{flow}[j] < Q[0]$  then
13      $\text{maxBefore} \leftarrow Q.\text{ExtractMax}()$ ;
14      $Q.\text{Insert}(\text{flow}[j])$ ;
15      $\text{sum} \leftarrow \text{sum} - \text{maxBefore} + \text{flow}[j]$ ;
16     if  $\text{minCost} > \text{sum} \times \text{divide}[j]$  then
17        $\text{minCost} \leftarrow \text{sum} \times \text{divide}[j]$ ;
18     end
19   end
20 end
21 return  $\text{minCost}$ 
```

---



4.3 Complexity Analysis

以  $T(n,k)$  表示算法的时间复杂度,  $n$  为备选地址数,  $k$  为挑选地址数, 在伪代码中, 第 2-4 行遍历复杂度为  $O(n)$ ; 第 5-6 行两次排序复杂度为  $O(n \log n)$ ; 第 8 行构建优先队列的时间复杂度为  $O(k)$ ; 第 9 行计算  $flow[1]$  到  $flow[k]$  的和同样需要遍历, 时间复杂度为  $O(k)$ ; 第 11 到 20 行的循环体中, 一次循环最坏情况下需要执行第 13 行的  $ExtractMax()$  和  $Insert()$ , 复杂度均为  $O(\log k)$ , 则循环复杂度为  $O((n - k) \log k)$ 。综上有:

$$O(n,k) = O(n) + O(n \log n) + O(k) + O((n - k) \log k) = O(n \log n) + O(k \log k)$$

5 交通建设问题

5.1 Main Idea

- 简要描述策略
- 贪心算法正确性证明

5.2 Pseudo Code

Algorithm 6: 算法名	
Input:	输入参数
Output:	输出参数
<pre>1 if A then 2     A ← B 3 else 4     B ← A 5 end 6 return getKthElement(A, 1, n, B, 1, m, midIndex)</pre>	

5.3 Complexity Analysis