100

计算机学院《算法设计与分析》第三次作业

计算机学院　20373673　于敬凯

November 16, 2022

1　最长空位问题

给定一长度为 n 的 01 串 $S =< s_1, s_2, \cdots, s_n >$，仅有一次机会挑选其中两个元素 $s_i, s_j (1 \leq i, j \leq n)$ 并交换他们的位置。请设计算法求出交换之后的 S 中最多有几个连续的 0。

例如，串 $S =$ “10010101” 通过交换 s_4 和 s_7 可以变为 “10000111”，连续的 0 的数量为 4。

请先简要描述策略，然后写出伪代码，最后分析时间复杂度。

1.1　策略

由题意知，更换两个相同的元素没有意义，因此所更换的元素一定一个为 0 一个为 1。因此，我们的目标是找到最长的最多包含 1 个 1 的字符串。

为此，我们首先在字符串首尾各添加一个 1，即 $s_0 = 1, s_{n+1} = 1$，形成新字符串 S'。接着遍历字符串 S' 并记录所有 1 的位置 $loc[i]$（其中 i 表示字符串 S' 中第 i 个 1）。

当 S' 中 1 的个数大于 2 时，遍历字符串中的 1，计算 $loc[i+1] - loc[i-1] - 3$ 并记录最大值即为答案，返回；当 S' 中 1 的个数等于 2 时，说明原字符串 S 中全是 0，直接返回 n；因为 S' 首尾都是 1，因此 1 的个数不会小于 2。

1.2 伪代码

Algorithm 1: 最长空位问题——贪心算法

Input: 正整数 n，为字符串 S 长度，长度为 n 的字符串 $S[]$

Output: 通过一次交换元素后 S 中最多连续 0 的数量

```
function main(n, S[]):
    // 通过在字符串S首尾分别追加一个1构造新字符串S'
    S' ←' 1' + S +' 1';
    // 初始化loc数组，用于后续记录1的位置
    loc[] ← [];
    // 初始化字符串S'中1的数量的计数器
    cnt ← 0;
    // 记录字符串S'中1的位置并填充记录数组loc
    for i ← 0 to n + 1 do
        // 记录S'中所有1的位置
        if S'[i] =' 1' then
            loc[cnt] ← i;
            cnt ← cnt + 1;
        end
    end
    // 初始化答案变量ans
    ans ← 0;
    if cnt > 2 then
        // 遍历计算答案
        for i ← 1 to  cnt − 2 do
            ans ← Max{ans, loc[i + 1] − loc[i − 1] − 3};
        end
    else
        // 此时cnt = 2，说明原字符串中只有0直接返回n
        ans ← n;
    end
    return n;
end
```

1.3 时间复杂度分析 20

由上述策略分析和伪代码知，对 S 遍历时间复杂度为 $O(n)$，对 loc 遍历时间复杂度为 $O(n)$，综上所述本题时间复杂度为 $O(n)$。

2 最大收益问题

某公司有一台机器，在每天结束时，该机器产生的收益为 X_1 元。在每天开始时，若当前剩余资金大于等于 U 元，则可以支付 U 元来升级该机器（每天最多只能升级一次）。从升级之日起，该机器每天可以多产出 X_2 元的收益。即是说，在执行 K 次升级之后，这台机器每天的产出为 $X_1 + K \times X_2$ 元。

该公司初始资金为 C 元，请你设计算法求出 n 天之后该公司拥有的总资金的最大值。

请先简要描述策略，然后写出伪代码，最后分析时间复杂度。

2.1 策略

分析对第 $i(1 \leq i \leq n)$ 天，分析升级机器和不升级机器的不同点：

- 以不升级机器时从第 i 天到第 n 天产生的收益记为 $Profit_{base}$

- 升级机器时，当天要付出成本 U 元，从第 i 天到第 n 天每天可以额外获得收益 X_2 元，因此该情况下从第 i 天到第 n 天产生的收益为 $Profit_{base} - U + (n - i + 1) \times X_2$

综上所述，比较第 i 天不升级机器和升级的收益，若 $-U + (n - i + 1) \times X_2 > 0$，则说明升级机器带来的收益更大，反之说明不升级机器带来的收益更大，因此可以做出决策是否升级机器。

从 1 遍历到 n，对每天进行如上判断做出是否升级机器的决策并累加收益获得答案。

2.2 伪代码

Algorithm 2: 最大收益问题——贪心算法

Input: 初始资金 C 元，初始机器每天收益 X_1 元，升级机器成本 U 元，升级机器后每天额外收益 X_2 元，经营周期 n 天

Output: n 天后该公司拥有的总资金的最大值

```
function main(C, X1, U, X2, n):
    // 初始化公司总资金为C
    total ← C;
    // 初始化机器每天产生收益为X1
    daily ← X1;
    // 遍历每天决策是否升级机器并记录总资金变化
    for i ← 1 to n do
        // 如果升级机器收益更多，则更新机器每天产生收益，并且总资金减去升级成本
        if −U + (n − i + 1) × X2 > 0 then
            daily ← daily + X2;
            total ← total − U;
        end
        total ← total + daily;
    end
    return total;
end
```


2.3 时间复杂度分析

由上述策略和伪代码知会遍历每一天，因此时间复杂度为 $O(n)$。

3 探险家分组问题

营地中共有 n 个探险家，第 i 个探险家的经验值为 $e_i(1 \leq i \leq n)$。现他们希望组成尽可能多的队伍前去探险。探险家组建队伍需满足如下规则：

1. 探险家可以不参加任何队伍，即留在营地；

2. 如果第 i 个探险家参加了某支队伍，那么该队伍的人数应不小于其经验值 e_i。

请设计一个尽可能高效的算法求出最多可组建几支队伍前去探险，并分析其时间复杂度。

例如有 $n = 5$ 名探险家，其经验值分别为 $e = \{2, 1, 2, 2, 6\}$，则可组建 $(1, 2), (2, 2)$ 两支队伍，把经验值为 6 的探险家留在营地。

请先简要描述策略，然后写出伪代码，最后分析时间复杂度。

3.1 策略

首先将探险家经验值升序排序得到数组 arr_{sorted}；

然后，初始化变量 $cnt = 0$ 表示目前已经选择了 0 个冒险家；

接着，初始化变量 $num = 0$ 表示需要 num 个冒险家才可以组成一个队伍；

接着，遍历 arr_{sorted}，访问到第 i 个元素时，将 $cnt \leftarrow cnt+1$，并更新 $num \leftarrow Max\{num, arr_{sorted}[i]\}$，如果 $num = cnt$，则说明当前选择的 cnt 个冒险家可以组成一支队伍，将答案计数器自增 1，并将 cnt 清空表示还清空已选择冒险家，将 num 置 0。

经过上述遍历，返回组成队伍数。

3.2 伪代码

Algorithm 3: 探险家分组问题——贪心算法

Input: 探险家数量 n，长度为 n 的数组 arr 表示探险家经验值

Output: 可以组成的最多队伍数

```
1  function main(n, arr):
       // 将arr升序排序获得有序数组
2      arr_sorted ← Array.Sort(arr);
       // 初始化变量cnt ← 0表示当前已选择的冒险家数量
3      cnt ← 0;
       // 初始化变量num表示对当前已选择的冒险家们需要num个冒险家才可以组成一支队伍
4      num ← 0;
       // 初始化答案变量ans ← 0表示组成的队伍数
5      ans ← 0;
       // 遍历arr_sorted计算可以组成的队伍数
6      for i ← 1 to n do
           // 以cnt ← cnt + 1表示将当前冒险家选入队伍
7          cnt ← cnt + 1;
           // 并更新num表示当前已选择的冒险家们组成队伍至少需要的人数
8          num ← Max{num, arr_sorted[i]};
           // 如果已选择冒险家数等于组成队伍所需数，说明组成队伍
9          if num = cnt then
               // 答案自增1
10             ans ← ans + 1;
               // 计数器清零
11             cnt ← 0;
               // 队伍所需人数清零
12             num ← 0;
13         end
14     end
15     return ans;
16 end
```


3.3 时间复杂度分析

对 arr 数组排序时间复杂度为 $O(nlogn)$，对 arr_{sorted} 遍历时间复杂度为 $O(n)$，综上所述，时间复杂度为 $O(nlogn)$。

4 分店选址问题

某奶茶品牌想在全国投资开分店来扩大规模提高影响力，通过向新老顾客发放问卷的形式调研产生了 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 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。由于当前总资金有限，该奶茶品牌希望根据这两组数据，从 n 个备选地址中挑选出 k 个组成最终分店名单，使得能够满足在以下约束条件的前提下尽可能降低总投资成本。

1. 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来投入资金；

2. 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。

请设计一个算法求满足上述条件的前提下，该奶茶品牌需要投入的总资金的最小值。

请先简要描述策略，然后写出伪代码，最后分析时间复杂度。

4.1 策略

首先，我们定义 $\frac{cost[i]}{flow[i]}$ 为单位人流量投入资金。

为了满足第一条条件，每个分店的单位人流量投入资金是相同的。为了满足第二条条件，单位人流量投入资金应为分店中 $\frac{cost[i]}{flow[i]}$ 最大的。

通过上述分析可以知道，对于一个特定分店选择方案而言，其成本由两点决定：$\frac{cost[i]}{flow[i]}$ 最大的店的单位人流量投入资金 $cost_{unit}$ 和该方案中所有分店的人流量和 $flow_{total}$，总成本即 $cost_{unit} \times flow_{total}$。

因此，贪心思路即首先按照单位人流量投入资金从小到大排序得到数组 $cost_{unit}[n]$，遍历第 $i(k < i \leq n)$ 个分店，以其作为分店方案中单位人流量投入资金最大的，接着只需要在 $cost_{unit}[1...i-1]$ 中找到 $k-1$ 个质量最小的分店，将这 k 个分店人流量求和后乘以第 i 个分店的单位人流量投入资金即得该方案下的最优解，遍历结束即求得全局最优解。

4.2 伪代码

Algorithm 4: 分店选址问题——贪心算法

Input: 备选地址数量 n，长度为 n 的人流量数组 $flow[]$，长度为 n 的所需最低资金数组 $cost[]$，所需分店数 k

Output: 选择 k 个分店的最低总投资成本

```
1  function main(n, flow[], cost[], k):
       // 将(i, cost[i]/flow[i])升序排序获得有序数组cost_unit
2      cost_unit ← Arrays.sort(i, cost[i]/flow[i]);
       // 初始化分店方案总人流量
3      flow_sum ← 0;
       // 将前k − 1个单位人流量投入资金最小的分店加入优先队列pq以初始化分店方案
4      pq = new PriorityQueue();
5      for i ← 1 to k − 1 do
           // 获取分店编号
6          index ← cost_unit[i][0];
           // 将分店人流量加入优先队列
7          pq.offer(flow[index]);
           // 更新当前总人流量
8          flow_sum ← flow_sum + flow[index];
9      end
       // 初始化全局答案
10     ans ← +∞;
       // 从第k个分店开始遍历cost_unit，以其作为最大单位人流量投入资金
11     for i ← k to n do
           // 获取分店编号
12         index ← cost_unit[i][0];
           // 获取当前分店单位人流量投入资金
13         unit ← cost_unit[i][1];
           // 将当前分店加入方案并更新总人流量
14         flow_sum ← flow_sum + flow[index];
           // 计算当前方案投资成本并更新答案
15         ans ← Min{ans, flow_sum × unit};
           // 移除当前分店方案中人流量最大的店铺以维持当前分店方案中为前i个分店中人流量最小的
16         temp ← pq.poll();
17         flow_sum ← flow_sum − temp;
18     end
19     return ans;
20 end
```

20

4.3 时间复杂度分析

对分店进行按单位人流量投入资金排序的时间复杂度为 $O(nlogn)$，遍历 $cost_{unit}$ 时间复杂度为 $O(n)$，维护优先队列时间复杂度为 $O(logn)$。综上所述，本题时间复杂度为 $O(nlogn)$

5 交通建设问题

现有 n 个城市，初始时任意两个城市之间均不可互。现有两种交通建设方案：

1. 花费 $c_{i,j}$ 的代价，在城市 i 和城市 j 之间建设一条道路，可使这两个城市互相可达。

2. 花费 a_i 的代价，在城市 i 建设一个机场，可以使得其与其他所有建设了机场的城市互相可达。

只要两个城市 i 和 j 之间存在一条可达的路径，则这两个城市也互相可达。请设计一个尽可能高效的算法求出所有城市之间互相可达所需的最小花费。

请先简要描述策略，然后写出伪代码，最后分析时间复杂度。

5.1 策略

分析题意，满足所有城市互相可达的情况有以下两种：

1. 全图为一张连通图。为了达到最优解，这张连通图实际上应当是一颗最小生成树。

2. 全图划分为若干个连通图，每个连通图内至少有一个城市建有机场以达到和其他连通图可达。为了达到最优解，每张连通图应当是一颗最小生成树，同时有且仅有一个机场位于该图内建设机场成本最低的城市。

以下分析实现算法：

首先，考虑不建设机场时的最优解，即满足上述分析第一条。直接使用 Prim 算法求解最小生成树即可。

然后，考虑建若干个机场的情况。我们虚拟一个 $airport$ 结点，令城市 i 到 $airport$ 的距离为 a_i。在新增了 $airport$ 结点的图中使用 Prim 算法求解最小生成树，得到建设机场时的最优解，即满足上述分析第二条。

综合以上两种情况的局部最优解，以最小值作为全局最优解。

5.2 伪代码

Algorithm 5: 交通建设问题——贪心算法

Input: 城市数量 n，大小为 $n \times n$ 的二维数组 $c[][]$ 表示两个城市之间建设道路成本，长度为 n 的数组 $a[]$ 表示建设机场成本

Output: 使所有城市之间互相可达所需的最小花费

function $main(n, c[][], a[])$**:**

// 获取全局结点集合

$V \leftarrow \{1 \cdots n\}$;

// 获取全局边集合

$E \leftarrow \{< u, v, w > | u, v, \in V, u \neq v, w = c[u][v]\}$;

// 计算不建设机场时的边集

$T \leftarrow Prim(V, E)$;

// 计算不建设机场时的最小建设成本

$cost_{min} \leftarrow \sum_{t \in T} c[t.u][t.v]$;

// 初始化虚拟结点$airport$

$airport \leftarrow n + 1$;

// 将$airport$全局点集合

$V' \leftarrow V \cup \{airport\}$;

// 将各个城市与$airport$的边a_i加入边集合

$E' \leftarrow E \cup \{< u, airport, w | u \in V, w = a[u]\}$;

// 计算建设机场时的边集

$T' \leftarrow Prim(V', E')$;

// 计算建设机场时的最小建设成本

$cost'_{min} \leftarrow \sum_{tinT'c[t.u][t.v]}$;

// 初始化答案变量

$ans \leftarrow 0$;

// 比较获得全局最优解

if $cost_{min} < cost'_{min}$ **then**

  $ans \leftarrow cost_{min}$;

else

  $ans \leftarrow cost'_{min}$;

end

return ans;

end

5.3 时间复杂度分析 20

由上述伪代码分析，一共进行了两次 $Prim$ 算法求解最小生成树，点的规模是 n，边的规模是 n^2，因此总时间复杂度为 $O(n^2)$。