计算机学院《算法设计与分析》第三次作业

November 20, 2024

1 医院排队问题

在一个医院中，有 n 个病人排队等待就诊，医院每次只能接诊 1 人。每个病人有一个病情严重度 s_i，不同病人的病情严重度可以相同。s_i 越大表示病情越严重，病情越严重的病人应当越优先就诊，形式化地说：对于任意两个病人 i, j，如果接诊顺序 $o_i < o_j$，那么一定有 $s_i \geq s_j$。同时，每个病人还需要一定的时间 t_i 来完成就诊。记每个病人在排队时的等待时间为 w_i，即从第一个人开始看病，到自己看完病为止的时间。

请你安排病人接诊的顺序，在关注病人病情严重程度的前提下最小化总的等待时间 $\sum w_i$。

例如，有 $1, 2, 3$ 号三位病人前来就诊，$s = [1, 1, 2]$, $t = [1, 2, 3]$，那么就诊顺序应该安排为 3 号 $\rightarrow$ 1 号 $\rightarrow$ 2 号，总等待时间为 $\sum w_i = (3) + (3 + 1) + (3 + 1 + 2) = 13$。

请设计一个高效算法，描述算法的核心思想，给出算法伪代码并分析其对应的时间复杂度。

1.1 贪心策略

该算法显然为贪心策略算法

核心思路：

1. 优先考虑病情严重程度：首先，病情越严重的病人应当越早得到治疗，这是首要原则。

2. 在相同严重程度下优化等待时间：对于具有相同病情严重程度的病人，我们应该优先选择需要较短时间完成就诊的病人，以减少后续病人的等待时间。

算法步骤：

1. 将所有病人按照病情严重程度 (s_i) 降序排序；如果病情严重程度相同，则按所需就诊时间 (t_i) 升序排序。

2. 按照上述排序结果依次接诊病人。

1.2 时间复杂度分析

排序操作：排序操作的时间复杂度为 $O(n \log n)$ 遍历操作：遍历病人列表以计算总等待时间和更新当前时间的操作时间复杂度为 $O(n)$。因此，整个算法的时间复杂度为 $O(n \log n)$。

1.3 医院排队伪代码

Algorithm 1: waiting queue($p[1..n]$)

Input: 输入的病人信息 $p[n]$，其中每个病人含有紧急程度 s 和就诊时间 t

Output: 一个正整数 $time$，为最短的等待时间

```
1 global p[n]
2 sort(p[n]);        将病人按病情严重程度降序排序，若相同则按就诊时间升序排序快速排序
3 time ← 0;
4 currenttime ← 0;
5 for i ← 1 to n do
6 |   time ← time + currenttime
7 |   currenttime ← currenttime + p[i].t;
8 end
9 return time
```

2 生产线规划问题

某制造公司拥有一条生产线，工作时每天可获得收益 X_1 元，初始资金为 C 元。若当前资金大于等于升级费用 U 元，公司可以选择对生产线进行升级。每次升级会使生产效率提高，每天生产可以带来额外收益 X_2 元，且升级可以叠加，也就是说在第 K 次升级后，生产一天的收益为 $X_1 + K \times X_2$。但升级过程需要停产（包括升级当天）T 天，期间无法产生收益。

在接下来的 n 天内，公司希望通过合理安排升级时机以最大化最终资金。

请设计一个高效的算法，规划生产线的策略，以在 n 天内最大化公司的最终资金。请描述算法的核心思想，给出算法伪代码并分析其时间复杂度。

2.1 贪心策略

根据机器产出的公式可以看出，每次升级其实是相互独立的。我们可以将每次升级看作花费 U 元购买了一台新机器，这台新机器每天可以产出 X_2 元。

因此，在第 i 天升级（购买新机器），这台新机器的总产出为 $(n - i - T + 1) \times X_2$ 元。只要该总产出大于升级的成本 $U + T \times X_1$，我们就应该在第 i 天进行升级。当然，还需要根据第 i 天时的剩余资金来判断这一天是否可以进行升级。

2.2 时间复杂度分析

遍历一次生产天数即可得到最终解，故时间复杂度为 $O(n)$。

2.3 作业题目 3 伪代码

Algorithm 2: profit(X_1, X_2, U, C, n, T)

Input: 机器初始产出 X_1，升级的收益 X_2，升级所需成本 U，初始资金 C ，总天数 n，以及停工天数 T

Output: 该公司 n 天后总资金的最大值

```
for i ← 1 to n do
    if (n − i − T + 1) × X2 > U + T × X1 and C ≥ U then
        C ← C − U;
        X1 ← X1 + X2;
        i ← i + T;
    end
    else
        C ← C + X1;
    end
end
return C
```

3 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，使得能够在满足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。例如，在 $k=3$ 的情况下，如果最终分店选址为 a, b, c 三地，而这三地的人流量分别为 100, 200, 300，那么最终投入在 a, b, c 三地资金比例应严格保证为 1:2:3。

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。请描述算法的核心思想，给出算法伪代码并分析其时间复杂度。

3.1 贪心策略

假设我们最终挑选出的分店名单为 $[h_1, h_2, \ldots, h_k]$，其中 h_i 表示编号为 h_i 的备选地址，这组选址的总人流量为 totalFlow，总投资成本为 totalCost。那么按照题目的要求，对于每个选址 h_i 需要满足：

$$\text{totalCost} \times \frac{\text{flow}[h_i]}{\text{totalFlow}} \geq \text{cost}[h_i]$$

即：

$$\text{totalCost} \geq \text{totalFlow} \times \frac{\text{cost}[h_i]}{\text{flow}[h_i]}$$

所以当总人流量固定时，总投资成本仅与这 k 个选址的 $\max_{1 \leq i \leq k} \frac{\text{cost}[h_i]}{\text{flow}[h_i]}$ 有关。

因此贪心思路为：设每个选址 i 的权重为 $\text{weight}[i] = \frac{\text{cost}[i]}{\text{flow}[i]}$，那么当我们以某个选址 x 作为最终选址名单中权重最高的时，选址名单中的其他选址只需要从权重小于等于 $\text{weight}[x]$ 的集合中选择人流量最低的 $k-1$ 个地址来组成最终选址名单即可，此时便能达到以地址 x 为权重最高的最终选址名单的总人流量最小，从而达到总投资成本最小。

我们可以枚举以每一个能成为最终选址名单中权重最大的地址来计算最少总投资成本，然后取其中的最小值即可。在处理过程中可以先将备选地址按照权重进行升序排列，然后在遍历过程中可以用优先队列维护之前人流量最低的 $k-1$ 个备选地址。

3.2 时间复杂度分析

时间开销主要为排序和每个元素进出优先队列的时间复杂度，故总时间复杂度为 $O(n \log n)$。

3.3 选址问题伪代码

Algorithm 3: Address($flow[1..n], cost[1..n], k$)

Input: 各个备选地址的人流量 $flow[1..n]$，各个备选地址的最低投资成本 $cost[1..n]$，开设的分店数目 k

Output: 一个正整数 $minCost$，奶茶品牌需要投入的总资金的最小值

```
sort(h[1..n]);   对备选地址按照最低投资成本与人流量的比值作为权重来进行升序排列
priorityQueue ← 初始化一个基于大顶堆的优先队列;
totalFlow ← 0;
minCost ← ∞;
for i ← 1 to k − 1 do
    totalFlow ← totalFlow + flow[h[i]];
    priorityQueue.push(flow[h[i]]);
end
for i ← k − 1 to n do
    totalFlow ← totalFlow + flow[h[i]];
    priorityQueue.push(flow[h[i]]);
    totalCost ← totalFlow × cost[h[i]] / flow[h[i]];
    minCost ← min(minCost, totalCost);
    totalFlow ← totalFlow − priorityQueue.top();
    priorityQueue.pop();
end
return minCost
```

4 食物链计数问题

对于一个生态系统，存在一个食物网 G，该食物网中有 n 个结点，捕食关系为二元组的集合 $A = \{(i, j)\}$，其中 (i, j) 表示 j 捕食 i，体现在食物网中为 i 指向 j 的一条有向边。保证该食物网中的任一结点都存在与之相连的边，且食物网中不存在环。

食物链是食物网中以一种生物为起点，另一种生物为终点的有向路径。极大食物链定义为：位于链条起点的是生产者，它们不捕食其他生物而是通过光合作用等方式获取能量，即该节点的入度为 0；而位于链条终点的是顶级消费者，这类生物不再被其他生物所捕食，即该节点的出度为 0。

请设计一个高效的算法统计该食物网中极大食物链的数量。请描述算法的核心思想，给出算法伪代码并分析其时间复杂度。

4.1 算法核心思路

由题意知：

1. 我们要入度为零的点开始查找；
2. 求路径数，当且仅当一个点的入度变为零时才需要入队，并不是数据更新一次就要入队；
3. 出度为零的点的路径总数和就是答案

故思路呼之欲出：**拓扑排序**

拓扑排序的精髓就在于每个点只会入队一次，每条边只会通过一次，所以时间复杂度就有很好的保证，$O(N + M)$

题目的说明告诉我们这是一张 DAG（有向无环图），因此必定存在一个入度为 0 的点，也因此每一个点都会被遍历。

我们遍历这张 DAG 时，当一个点的入度变为零，即所有它能吃的东西都已经搜索过了，这时它的数值就不会发生变化，就可以入队了。这样保证了队列里的所有数值都不会发生变化。

以下为代码的变量解释：

- f_i 表示到达 i 时的路径数
- h_i 表示在可以直接吃掉 i 的所有关系中最后的一条的编号（邻接矩阵用）
- in_i 表示 i 的入度，是整个程序的核心数组
- out_i 表示 i 的出度
- 用节点 $node$ 记录 m 条捕食关系，$node.a$ 记录被捕食者，$node.b$ 记录捕食者，$node.n$ 记录 h_a

4.2 时间复杂度分析

由前文关于拓扑排序的时间复杂度分析可知，时间复杂度为 $O(n+m)$，其中 n 为 DAG 中点的个数，m 为 DAG 中边的个数。

4.3 食物链计数伪代码

Algorithm 4: chain(n, w_n, s_n, v_n)

Input: n 个动物，m 条捕食关系

Output: 最大食物链条数 num

```
//初始化邻接表，假设对于 m 条捕食关系，其中 a 为被捕食者，b 为捕食者，都进行了以下记录
node[i].a ← a, node[i].b ← b, node[i].n ← h[a], h[a] ← i;
out[a] ← out[a] + 1, in[b] ← in[b] + 1;   //第 i 条关系
//初始化拓扑排序所需模版队列 q
for i ← 1 to n do
    if in[i] = 0 then
        f[i] ← 1;
        q.push(i);
    end
end
while !q.empty() do
    a ← q.front();
    q.pop();
    for k ← h[a]; k ∉ ∅; k ← node[k].n do
        b ← d[k].b;
        f[b] ← f[b] + f[a];
        in[b] ← in[b] − 1;
        if in[b] = 0 then
            if out[b] = 0 then
                num ← num + f[b];
            end
            else
                q.push(b);
            end
        end
    end
end
return num
```

5 迷宫逃离问题

给定一个 $m \times n$ 的迷宫，其入口和出口分别为 $(1,1)$ 和 (m,n)（行列坐标都从 1 开始）。每个格子的状态用 $c_{i,j}$ 表示，并存在两种取值：

1. $c_{i,j} = 0$，表示这个格子是空格子，可以通过；
2. $c_{i,j} = 1$，表示这个格子是障碍物，不可通过。

入口 $(1,1)$ 和出口 (m,n) 均为空格子。在迷宫中可从某个格子 (i,j) 移动到与其相邻的空格子 $((i,j-1),(i,j+1),(i-1,j),(i+1,j))$ 其中之一，且需保证移动后的横坐标必须位于 $[1,m]$ 之间，纵坐标必须位于 $[1,n]$ 之间），消耗体力为 1。

现可将至多 1 个障碍物移除，使其对应的格子的变为空格子，移除障碍物不消耗体力。

请在此基础上设计一个尽可能高效的算法，求出从入口 $(1,1)$ 到出口 (m,n) 需消耗的最小体力，描述算法的核心思想，给出算法伪代码并分析其时间复杂度。

5.1 搜索状态设计

经过每个格子时有尚未移除障碍物，和已经移除障碍物两种状态，故状态空间是 $O(n \times m \times 2)$ 的。可以尝试使用广度优先搜索。

用状态 $(x, y, 0)$ 表示在 (x, y) 点还没有移除障碍物，状态 $(x, y, 1)$ 表示在 (x, y) 点已经移除了某一个障碍物。

5.2 搜索过程

在状态 $(x, y, 0)$ 可以转移到如下几个状态:

1. $(x, y-1, 0)$, $(x-1, y, 0)$, $(x, y+1, 0)$, $(x+1, y, 0)$ 如果对应的新的格子是可以通过的空格子;
2. $(x, y-1, 1)$, $(x-1, y, 1)$, $(x, y+1, 1)$, $(x+1, y, 1)$ 如果对应的新的格子是障碍物，则移除之。

而状态 $(x, y, 1)$ 只能转移到四个方向相邻的空格子中去: $(x, y-1, 1)$, $(x-1, y, 1)$, $(x, y+1, 1)$, $(x+1, y, 1)$。

5.3 时间复杂度分析

由于每个格子只会被搜索一次，搜索时每次转移格子时间是常数级的，所以时间复杂度为 $O(n \times m)$

5.4 迷宫逃离伪代码

Algorithm 5: maze$(m, n, C[1..m][1..n])$

Input: 给定的迷宫 $C_{m\times n}$

Output: 消耗的最少体力 ans

```
Q ← ∅;
Q.push(1, 1, 0);
vis[1..m][1..n][0..1] ← 0;    //初始化记忆数组
vis[1][1][0] ← 1;
energy[1][1][0] ← 0;
while Q ≠ ∅ do
    (x, y, z) ← Q.pop();
    for (x', y') ← (x−1, y), (x + 1, y), (x, y−1), (x, y + 1) do
        if C[x'][y'] = 0 then
            if vis[x'][y'][z] = 0 then
                vis[x'][y'][z] ← 1;
                energy[x'][y'][z] ← energy[x][y][z] + 1;
                Q.push(x', y', z);
            end
        end
        else
            if vis[x'][y'][1] = 0 ∩ z = 0 then
                vis[x'][y'][1] ← 1;
                energy[x'][y'][1] ← energy[x][y][z] + 1;
                Q.push(x', y', 1);
            end
        end
    end
end
ans ← ∞;
if vis[m][n][0] = 1 then
    ans ← min(ans, energy[m][n][0]);
end
if vis[m][n][1] = 1 then
    ans ← min(ans, energy[m][n][1]);
end
return ans
```