浙江大學

本科生课程报告

学年、学期：2021 — 2022 学年 秋冬 学期

课程名称：数据分析与算法设计

任课教师：李东晓

题　　目：自选编程作业 2

学生姓名：黄嘉欣

学　　号：3190102060

数据分析与算法设计：自选编程作业 2

3190102060 黄嘉欣 信工 1903 班

一、题目：871.最低加油次数(难度：困难)

汽车从起点出发驶向目的地，该目的地位于出发位置东面 target 英里处。沿途有加油站，每个 stations[i]代表一个加油站，它位于出发位置东面 stations[i][0]英里处，并且有 stations[i][1]升汽油。假设汽车油箱的容量是无限的，其中最初有 startFuel 升燃料。它每行驶 1 英里就会用掉 1 升汽油。当汽车到达加油站时，它可能停下来加油，将所有汽油从加油站转移到汽车中。为了到达目的地，汽车所必要的最低加油次数是多少？如果无法到达目的地，则返回-1 。

注意：如果汽车到达加油站时剩余燃料为 0，它仍然可以在那里加油。如果汽车到达目的地时剩余燃料为 0，仍然认为它已经到达目的地。

各变量满足条件：

① 1 <= target, startFuel, stations[i][1] <= 10^9

② 0 <= stations.length <= 500

③ 0 <stations[0][0] < ... < stations[stations.length-1][0] < target

二、算法设计：动态规划

由题意，我们需要求出汽车到达目的地所需的最低加油次数，则必须将所有的可能情况一一列举，从中找出最小值。面对此穷举问题，可以考虑动态规划算法，根据状态转移方程推导状态，利用空间换取时间，巧妙地解决问题。其具体思路分析如下：

由题意，当汽车行驶到加油站时，可以根据自己剩余油量的多少选择是否加油，其所能行驶的剩余距离与当前剩余油量有关。因此，可将汽车行驶的最远距离作为动态规划对象，其可与已经过的加油站数和已加油次数相联系，即设 F(i,j)表示汽车通过第 i 个加油站、加过 j 次油后所能到达的最远距离(j<=i)，若令 n=stations.length,，则当且仅当 F(n,j)>=target 有解时，此问题有解，此时汽车的加油次数为 j。从所有满足条件的 j 中选择最小的一个，即为汽车所必要的最低加油次数；反之，若 j 不存在，则表示汽车无法到达目的地，应返回-1。

通过上述假设，当汽车到达第 i 个加油站时，其可以选择是否加油：若不加油，显然，此时所能够行驶的最远距离与其通过第 i-1 个加油站、加过 j 次油后所能到达的距

离相同，即 F(i,j)=F(i-1,j)。若选择加油，则汽车此时所能行驶的最远距离将在前一次加油后能够行驶距离的基础上加上此加油站存有的油量(1L 汽油可行驶 1 英里)，即 F(i,j)=F(i-1,j-1)+stations[i][1]。通过选取加油与不加油两种情况取值相对更大的 F(i,j)，即可得到汽车通过第 i 个加油站、加过 j 次油后所能到达的最远距离。对初始状态，考虑汽车不加油(j=0)，则必有 F(i,0)=startFuel。综上，此问题的状态转移方程为：

$$\begin{cases} F(i,j) = F(i-1,\ j),\ 0 \le j \le i \le n,\ \text{第 i 站不加油} \\ F(i,j) = \max\{F(i,\ j), F(i-1,\ j-1) + \text{stations}[i][1]\},\ 0 \le j \le i \le n,\ \text{第 i 站加油} \\ F(i,0) = \text{startFuel},\ 0 \le\ i \le n \end{cases}$$

为了在计算 F(i,j)之前确保 F(i-1,j)和 F(i-1,j-1)的值已被算出，我们需要通过从上往下的遍历方式逐行填充二维数组，最终判断 F(n,j)与 target 的大小关系。此算法的伪代码为：

```
算法：minRefuelStops(int target, int startFuel, int** stations)
    // 求解汽车所必要的最低加油次数
    // 输入：目的地距离 target，最初燃料升数 startFuel，加油站 stations，
    // 其中 stations[i][0]表示加油站位置，stations[i][1]表示加油站燃料
    // 输出：最低加油次数，若无解则返回-1
    // 注意：F(,)为 n*n 数组，其中 n=stations.length，不再额外定义
    n <- stations.length
    for i <- 0 to n do
        F(i,0) = startFuel; // 初始条件
    for i <- 0 to n do // 从上到下逐行填写
        for j <- 0 to i do
            if 第 i 站不加油 do // 第 i 站不加油
                F(i,j) = F(i-1,j)
            else do // 第 i 站加油
                F(i,j) = max(F(i,j),F(i-1,j-1)+stations[i][1])
    for j <-0 to n do // 从小到大遍历，找到最小的 j
        if F(n,j) >= target
            return j
    return -1
```

三、代码实现

根据(二)中伪代码，当汽车行驶到第 i 站不加油时，其行驶到第 i-1 站时的可到达最远距离必须满足 F(i-1,j)>=stations[i-1][0]；同理，若汽车在第 i 站加油，则有 F(i-1,j-1)>=stations[i-1][0]，否则汽车无法到达第 i-1 站。综上，可得 C 语言代码实现为：

代码：
```
int minRefuelStops(int target, int startFuel, int** stations,
                   int stationsSize, int* stationsColSize)
{
    int n = stationsSize;
    long F[n+1][n+1]; // 最大距离矩阵
    // 大小设置为(n+1)*(n+1)是为了防止越界
    int i, j;
    for (i=0;i<n+1;i++){
        for (j=0;j<n+1;j++){
            F[i][j] = 0; // 二维数组初始化
        }
    }
    for (i=0;i<n+1;i++){
        F[i][0] = startFuel; // 初始条件
    }
    for (i=1;i<n+1;i++){ // 从上到下逐行填写
        for (j=1;j<=i;j++){ // 注意下标不能小于 0
            if (F[i-1][j]>=stations[i-1][0]){ // 第 i 站不加油
                F[i][j] = F[i-1][j];
            }
            if (F[i-1][j-1]>=stations[i-1][0]){ // 第 i 站加油
                if (F[i][j] < F[i-1][j-1]+stations[i-1][1]){ // 更大值
                    F[i][j] = F[i-1][j-1]+stations[i-1][1];
                }
            }
        }
    }
    for (j=0;j<=n;j++){ // 从小到大遍历，找到最小的 j
        if (F[n][j] >= target){
            return j;
        }
    }
    return -1;
}
```

四、运行结果

如图 4.1，将动态规划算法代码提交至 LeetCode，得其执行用时为 76ms，内存消耗 8.9MB，通过全部 198 个测试用例。当测试输入为{100 1 [[10,100]]}时，此问题无解，程序输出为-1，与正确结果一致，如图 4.2 所示。采用自测试实例，若输入的数据为{100 10 [[10,60],[20,30],[30,30],[60,40]]}，则输出为 2，正确；若输入数据为{1 1 []}，得输出为 0，与理论一致，如图 4.3 所示。对于更多的输入可能，无

法一一列举，但由 LeetCode 测试结果可知，算法设计正确。

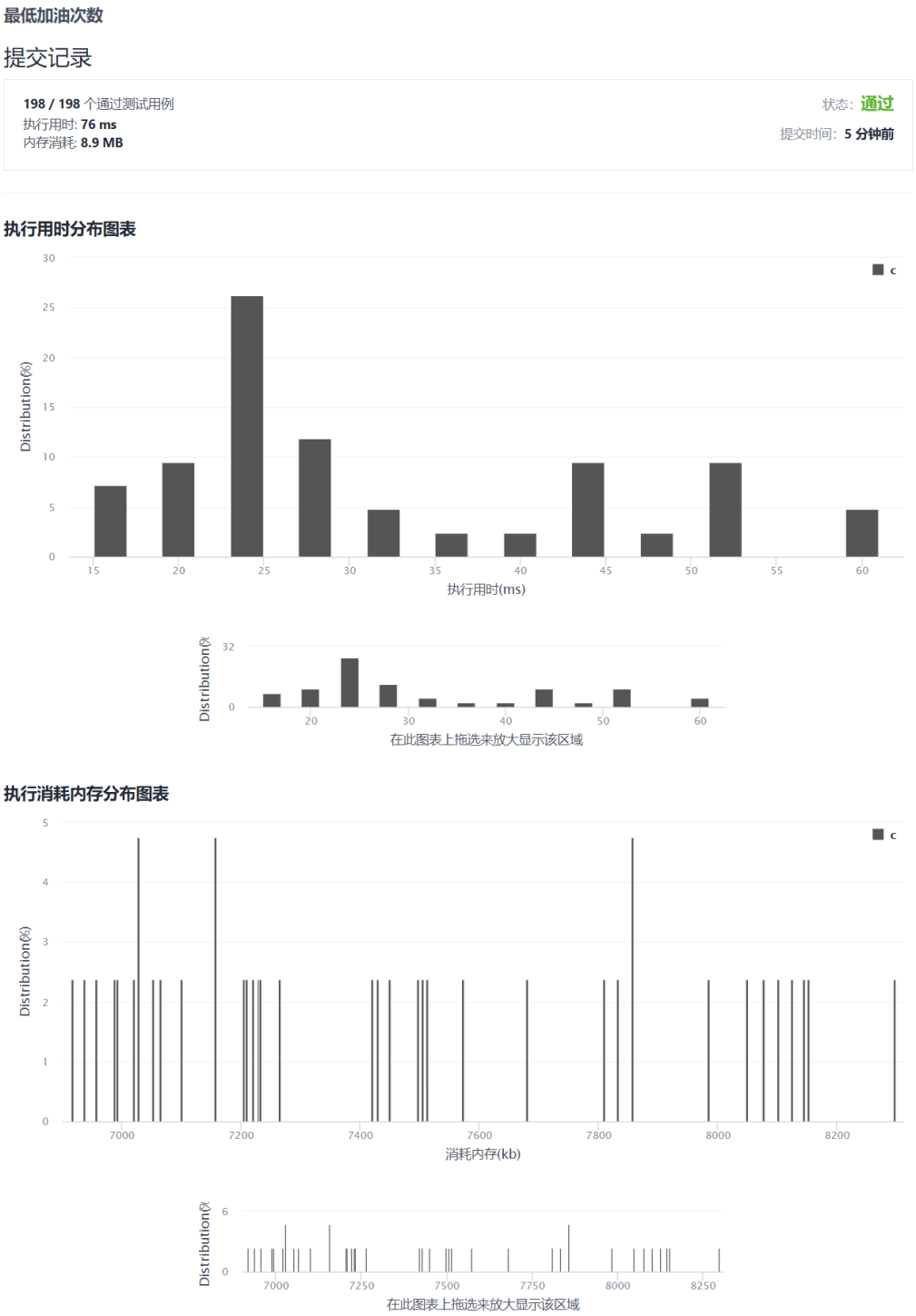

图 4.1 LeetCode 算法运行结果

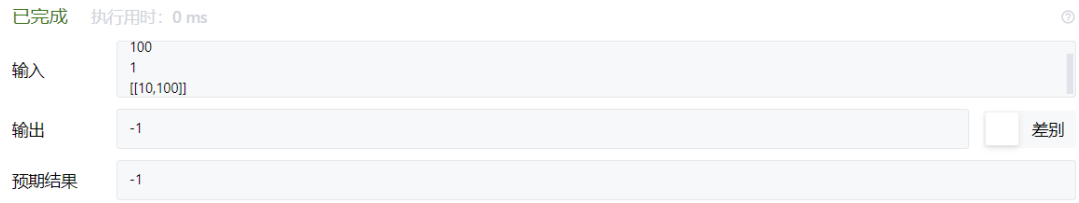

图 4.2 LeetCode 测试实例

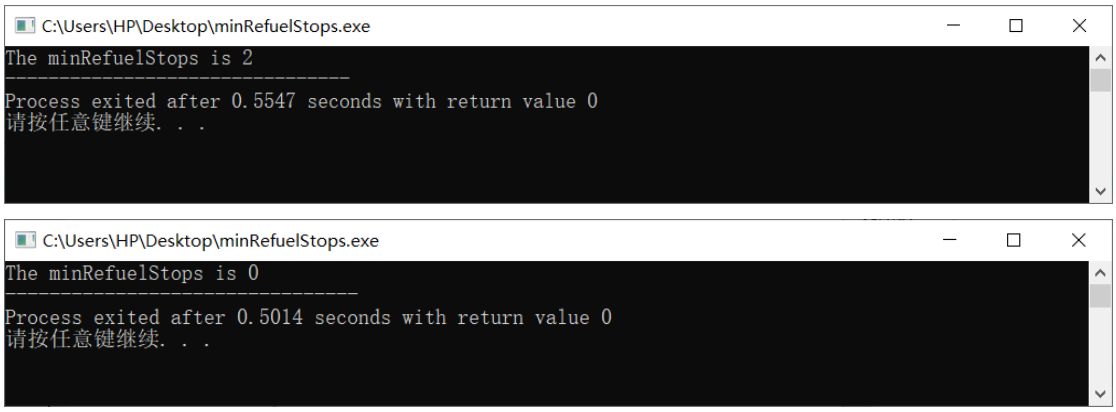

图 4.3 自测试实例

五、效率分析

由代码，设对输入规模为 n 的数组，其基本操作(比较)执行次数为 C(n)，则有：

$$C(n)=\sum_{i=0}^{n}\sum_{j=0}^{i}1=\sum_{i=0}^{n}(i+1)=\frac{(n+1)(n+2)}{2}\in O(n^2)$$

即动态规划算法的时间效率为 $O(n^2)$。

显然，由于在动态规划算法中，我们只需要对最大距离数组 F[n+1][n+1]进行填充，故所需的额外空间为：$V(n)=(n+1)^2\in O(n^2)$，即算法的空间效率为 $O(n^2)$。

六、总结

通过此题可以发现，对于看起来较为复杂的问题，特别是求解最值的问题，若能够合理转换、寻找递推关系式，我们便可以将其分解成相互独立、相对简单的小规模问题，逐步求解，并将结果记录在表中，最终通过处理表中数据得到原始问题的解。总的来说，动态规划算法具有较好的时间效率和空间效率，对一些特定题型具有“奇效”，需要我们能够放宽思路、灵活掌握。