数据结构 Lab-6 实验报告

林子开 21307110161

2023 年 11 月 10 日

目录

1 算法描述

首先，对工作按照最晚结束时间重新排序，使得对于工作 $a_1, a_2, \cdots, a_n$，满足

$$d_1 \leq d_2 \leq \cdots \leq d_n$$

然后，定义 $G(i, j)$ 为在时间 j 之内，从工作 $a_1, a_2, \cdots, a_i$ 中选取**满足约束条件**的工作，所能得到的**最大的收益**。注意到每一项工作最多工作的时间为 n，完成 n 项工作最多需要的时间是 n^2，因此最终要求的总收益即为 $G(n, n^2)$。根据题意，$G(i, j)$ 满足以下的递推关系：

$$y = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ G(i, d_i) & \text{if } j > d_i \\ \max\{G(i-1, j), G(i-1, j-t_i) + p_i\} & \text{if } j \leq d_i \text{ and } j - t_i \geq 0 \\ G(i-1, j) & \text{if } j \leq d_i \text{ and } j - t_i < 0 \end{cases}$$

对上述递推关系的解释如下：

- 若 $i = 0$，则说明没有任何工作可选，此时收益显然为 0。若 $j = 0$，则说明此时没有任何时间可以使用，则收益也显然为 0。
- 若 $j > d_i$，注意到 d_i 是 $a_1, a_2, \cdots, a_n$ 中允许最晚结束的工作的结束时间，当 $j > d_i$ 时，其实在时间区间 $(d_i, j]$ 中，已经不可能分配任何工作，因此能够得到的最大收益等于在时间区间 $[0, d_i]$ 中安排工作所能得到的最大收益。
- 若 $j \leq d_i, j - t_i \geq 0$，此时总的时间 j 大于进行允许最晚结束的工作的耗时 t_i，此时有两种选择：第一，可以选择做结束时间（d_i）最晚的工作 a_i，此时得到的最大收益为

$G(i-1, j-t_i)+p_i$，其中 p_i 是做工作 a_i 所得到的收益，$G(i-1, j-t_i)$ 是让 a_i 压线完成后（即 a_i 恰好在 j 时刻完成），把剩余的时间区间 $[0, j-t_i]$ 分配给剩下的 $a_1, a_2, \cdots, a_{i-1}$ 任务所能得到的最大收益。第二，也可以选择不做结束时间（d_i）最晚的工作 a_i，直接把时间区间 $[0, j]$ 分配给前 $i-1$ 项任务，此时的最大收益就是 $G(i-1, j)$。从以上两种中选择收益较大者，即为 $G(i, j)$。

- 若 $j \leq d_i, j - t_i < 0$，此时总的时间 j 小于进行允许最晚结束的工作的耗时 t_i，则仍然只能从 $a_1, a_2, \cdots, a_{i-1}$ 之中选择，因此有 $G(i, j) = G(i-1, j)$。

注意到递推关系中存在大量的重复计算，因此可以用**动态规划**进行求解。构建两张 $(n+1) \times (n^2+1)$ 的辅助表，分别为 `G` 和 `Action`，其中 `G` 存储最优收益，第 i 行第 j 列恰为上面所定义的 $G(i, j)$。`Action` 存储转移关系，在第 i 行，第 j 列，存储三元组 (u, v, a)，其中，若在计算 $G(i, j)$ 时，选择了执行工作 a_i，则把三元组中的 a 设置为 a_i，否则设置为 NULL；而计算 $G(i, j)$ 时，如果引用了第 u_0 行第 v_0 列的 $G(u_0, v_0)$，则将三元组的前两个元素设置为 u_0 和 v_0。

在最后，只需要从表格 `Action` 的最后一个三元组出发，根据转移函数走到表格的边界，并且一路记录所有的不等于 NULL 的 a，即可得到最优的安排方案。而最大收益即为表格 `G` 中的 $G(n, n^2)$。

2 复杂度分析

空间复杂度 由于需要额外建立两张 $(n+1) \times (n^2+1)$ 的辅助表格，分别用于存储局部的最优值和转移关系，因此，空间复杂度为 $\mathcal{O}(n^3)$。

时间复杂度 在算法执行的过程中，首先要对 n 项工作按照 deadline 进行升序排序，该步骤复杂度为 $\mathcal{O}(n \log(n))$；然后，要向两张复杂度为 $\mathcal{O}(n^3)$ 的表格填入有关数据，对于表格中每一个位置，运算复杂度为 $\mathcal{O}(1)$，则填写表格时间复杂度也为 $\mathcal{O}(n^3)$。综合考虑，该算法的时间复杂度为 $\mathcal{O}(n^3)$。

3 测试用例的实验结果

测试用例的实验结果如下：

1. 对于工作 $[(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4)]$，最大的收益为 **160.0**，获得最大收益的工作安排方式为 $[(1, 100, 2), (2, 60, 3)]$
2. 对于工作 $[(3, 100, 4), (1, 80, 1), (2, 70, 2), (1, 10, 3)]$，最大的收益为 **180.0**，获得最大收益的工作安排方式为 $[(1, 80, 1), (3, 100, 4)]$
3. 对于工作 $[(4, 100, 4), (2, 75, 3), (3, 50, 3), (1, 25, 1)]$，最大的收益为 **100.0**，获得最大收益的工作安排方式为 $[(1, 25, 1), (2, 75, 3)]$
4. 对于工作 $[(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (2, 50, 3)]$，最大的收益为 **160.0**，获得最大收益的工作安排方式为 $[(1, 100, 2), (2, 60, 3)]$
5. 对于工作 $[(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4), (2, 50, 3), (1, 80, 2)]$，最大的收益为 **220.0**，获得最大收益的工作安排方式为 $[(1, 100, 2), (1, 80, 2), (2, 40, 4)]$

6. 对于工作 $[(2,60,3),(1,100,2),(3,20,3),(2,40,2),(2,50,3),(1,80,2),(4,90,4)]$，最大的收益为 **180.0**，获得最大收益的工作安排方式为 $[(1,100,2),(1,80,2)]$

7. 对于工作 $[(3,60,3),(2,100,2),(1,20,2),(2,40,4),(4,50,4)]$，最大的收益为 **140.0**，获得最大收益的工作安排方式为 $[(2,100,2),(2,40,4)]$

8. 对于工作 $[(2,60,3),(1,100,2),(3,20,3),(2,40,2),(4,50,4),(1,80,2),(4,90,4)]$，最大的收益为 **180.0**，获得最大收益的工作安排方式为 $[(1,100,2),(1,80,2)]$

9. 对于工作 $[(3,60,3),(2,100,2),(1,20,2),(2,40,2),(4,50,4),(5,70,5)]$，最大的收益为 **100.0**，获得最大收益的工作安排方式为 $[(2,100,2)]$

10. 对于工作 $[(2,60,3),(1,100,2),(3,20,3),(2,40,2),(4,50,4),(5,70,5),(3,90,4)]$，最大的收益为 **190.0**，获得最大收益的工作安排方式为 $[(1,100,2),(3,90,4)]$

4 附录：本次实验所使用的 Python 代码

本次使用的 Python 代码如下所示

Listing 1: Python codes for scheduling

```python
import numpy as np

def get_time(job):
    """返回job的消耗时间"""
    return job[0]

def get_profit(job):
    """返回job的收益"""
    return job[1]

def get_ddl(job):
    """返回允许结束的最晚时间"""
    return job[2]


def find_best_schedule(jobs: list):
    """输入可以选择的工作jobs，返回最大的收益，以及最优的工作安排方案"""
    # 输入样例的格式为:
    # jobs = [(t1,p1,d1),(t2,p2,d2),(t3,p3,d3)....]
    jobs_by_ddl =  sorted(jobs,key = lambda job:job[2])  # 按照ddl升序排列
    n =  len(jobs_by_ddl)
    jobs_by_ddl.insert(0,None) # 使得job的下标从1开始
    # G 是辅助矩阵，用于存储收益
    G = np.zeros((n+1, n**2+1))  # 初始值全部设为零。G的上方和左边的值始终为0
    # Action 用于存储转移关系
    Action = G.copy()
    Action =  list(Action)
    for i in  range( len(Action)):
        Action[i] =  list(Action[i])
    for i in  range(1,n+1):
        t_i = get_time(jobs_by_ddl[i])
```

```
        p_i = get_profit(jobs_by_ddl[i])
        d_i = get_ddl(jobs_by_ddl[i])
        for j in  range(1,n**2+1):
            if j <= d_i and j - t_i >= 0: # 有时间做最后一项工作
                p1 = G[i-1,j] # 最后一项工作不做
                p2 = G[i-1,j-t_i] + p_i # 最后一项工作要做
                if p1 >= p2: # 最后一项工作不做
                    G[i,j] = p1
                    Action[i][j] = (i-1,j,None)
                elif p1 < p2: #要做最后一项工作
                     G[i,j] = p2
                     Action[i][j] = (i-1,j-t_i,i)
            elif j <= d_i and j - t_i < 0: #没有时间做最后一项工作
                G[i,j] = G[i-1,j]
                Action[i][j] = (i-1,j,None)
            elif j > d_i: # 时间太多了，超过最后一项工作的ddl
                G[i,j] = G[i,d_i]
                Action[i][j] = (i,d_i,None)

    best_jobs = []
    act = Action[-1][-1] # act是一个三元组
    while True:
        if act[-1] is not None: # 此时选择了某个job
            best_jobs.insert(0,jobs_by_ddl[act[-1]])
            if act[0] == 0 or act[1] == 0:
                break
            act = Action[act[0]][act[1]] # 转移关系
        else: # act[-1] is None:
            if act[0] == 0 or act[1] == 0:
                break
            act = Action[act[0]][act[1]] # 转移关系

    return G[-1,-1], best_jobs  # 最优收益，最佳安排方案


def main():
    sample_jobs = [
    [(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4)],
    [(3, 100, 4), (1, 80, 1), (2, 70, 2), (1, 10, 3)],
    [(4, 100, 4), (2, 75, 3), (3, 50, 3), (1, 25, 1)],
    [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (2, 50, 3)],
    [(2, 60, 3), (1, 100, 2), (3, 20, 4), (2, 40, 4), (2, 50, 3), (1, 80, 2)],
    [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (2, 50, 3), (1, 80, 2), (4, 90, 4)],
    [(3, 60, 3), (2, 100, 2), (1, 20, 2), (2, 40, 4), (4, 50, 4)],
    [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (4, 50, 4), (1, 80, 2), (4, 90, 4)],
    [(3, 60, 3), (2, 100, 2), (1, 20, 2), (2, 40, 2), (4, 50, 4), (5, 70, 5)],
    [(2, 60, 3), (1, 100, 2), (3, 20, 3), (2, 40, 2), (4, 50, 4), (5, 70, 5), (3, 90, 4)]
]

    for jobs in sample_jobs:
        max_profit, schedule = find_best_schedule(jobs)
        s = "\n \item 对于工作${}$，最大的收益为{}，获得最大收益的工作安排方式为${}$".
```

```
            format(jobs,max_profit,schedule)
        print(s)

if __name__ == '__main__':
    main()
```