# 算法设计大作业（三）

毛圆鑫 - 201721012271

2019 年 12 月 7 日

## 说明

这是算法设计课的第三个大作业，主题是**贪心算法的分析**，我完成的题目是**第三道**：明明忙得焦头烂额地交作业。

在**题目描述**中有完整的题目介绍，之后便是我的分析报告，整篇文档是使用 LaTeX 语言设计的，源码见附件或 github.com/punk-boy/algorithm/greedalgo.tex 和 greedalgo.cpp

## 1 问题的描述

明明的周日作业任务非常繁重，足足有 $N$ 个任务要做且周日当天上交，每个作业都需要做 1 个小时才能完成，为此，他在周日的凌晨 0 点就起床了，感觉自己已经到了焦头烂额、风吹就倒的地步。每位老师都给他设置了交作业的截止时间 $Deadline[i]$（一个 $1 \sim 24$ 之间的整数）。如果明明在截止时间后交作业，老师就会将他的作业分数相应减少 $Reduce[i]$ 分（一个 $1 \sim 99$ 之间的整数）。要求：已知第 $i$ 个作业的 $Deadline[i]$ 和 $Reduce[i]$，请你设计一个算法，帮助明明合理选择做作业的先后顺序，从而把被减掉的总分数降到最低，并将被减总分也告诉明明，让他有个心理准备。

我们可以称这个问题是**单位时间作业安排问题**。

## 2 问题的形式化

给定 $w = (w_1, w_2, \ldots, w_n)$，其中 $w_i > 0, i \in [1, n]$ 和 $d = (d_1, d_2, \ldots, d_n)$，其中 $d_i \in \{1, 2, \ldots, 24\}$，要求找到一个 $n$ 元 $0-1$ 向量 $x = \{x_1, x_2, \ldots, x_n\}$ 和互不相等的 $n$ 元向量 $t = (t_1, t_2, \ldots, t_n)$，$t$ 和 $x$ 存在约束：当 $x_i = 1$ 时，$t_i \in \{1, 2, \ldots, 24\}$ 且 $t_i < d_i$，当 $x_i = 0$ 时，$t_i = 0$。找到的 $x$ 使得 $\sum_{i=1}^{n} w_i x_i$ 最大。

$$max \sum_{i=1}^{N} w_i x_i \qquad \text{s.t.} \begin{cases} t_i = f(i) \\ t_i = t_j \leftrightarrow i = j \\ t_i \in \{1, 2, \ldots, 24\} \\ x_i \in \{0, 1\}, \quad 1 \leqslant i \leqslant n \end{cases}$$

# 3 求解过程分析

贪心算法要求我们每一步都通过局部最优的选择，能产生全局最优选择。每一个阶段，我们都选择当前看起来最优的决策，所有的阶段的决策完成之后，最终由些局部最优解构成全局最优解。这就需要我们给出**贪心选择性**和**最优子结构性质**的证明，前者保证该问题中选择局部最优值是可以得到全局最优解的，后者保证每个子问题都和原问题形似，所以也就可以采用相同的策略，选择当前子问题的局部最优值。这样通过归纳就可证明具有**贪心选择性**和**最优子结构性质**的问题可以采用贪心算法得到最优解。

我们按照**贪心规则：选择 Reduce 值大的那些作业先完成，如果截止时间相同，则根据 Deadline 的值判断谁更优先，尽量将对应作业安排在 Deadline 时间上完成，否则放在 Deadline 前空的一天，如果在 Deadline 前都有作业安排了，则舍弃这个作业**来求解该问题。

所以我们还需要一个数组用来标识当前时间是否以及有任务安排了以及对应的任务序号，我们这里定义 $finished$[24]，当 $finished_i = 0$ 表示当前时间 $i$ 没有任务安排，而 $finished_i = m$ 表示当前时间 $i$ 有任务 $m$ 需要完成。

## 3.1 贪心选择性质

**存在一个最优解选择了 Reduce 值最大的作业**

假设作业已经按照其 Reduce 值降序排列，且 $x = \{x_1, x_2, \ldots, x_n\}$ 是该问题的一个最优解，令 $k = \min\{i | x_i = 1\}$，

若 $k = 1$，则 $x$ 是**单位时间作业安排问题**的一个最优解；

若 $k > 1$，则构造 $n$ 元 $0-1$ 向量 $Y = \{y_1, y_2, \ldots, y_n\}$，其中 $y_k = 0$，$y_1 = 1$，$y_i = x_i (i \neq 1 \text{ 且 } i \neq k)$，有

$$\left.\begin{array}{r} \sum_{i=1}^{n} w_i y_i = \sum_{i=1}^{n} w_i x_i - w_k + w_1 \\ w_1 \geqslant w_k \end{array}\right\} \quad \sum_{i=1}^{n} w_i y_i \geqslant \sum_{i=1}^{n} w_i x_i$$

所以 $Y$ 是一个满足贪心选择的最优解

综上所述，存在一个最优解选择了 Reduce 值最大的作业

## 3.2 最优子结构性质

若 $x = \{x_1, x_2, x_3, \ldots, x_n\}$ 是**单位时间作业安排问题**的一个最优解，则 $\{x_2, x_3, \ldots, x_n\}$ 就是下述问题的一个最优解：

$$\max \sum_{i=2}^{n} w_i x_i \qquad \text{s.t.} \begin{cases} t_i = f(i) \\ t_i = t_j \leftrightarrow i = j \\ t_i \in \{2, 3, \ldots, 24\} \\ x_i \in \{0, 1\}, \quad 2 \leqslant i \leqslant n \end{cases}$$

证明：假设 $\{x_2, x_3, \ldots, x_n\}$ 不是上述子问题的最优解

则存在最优解 $\{y_2, y_3, \ldots, y_n\}$ 使得 $\sum_{i=2}^{n} y_i > \sum_{i=2}^{n} x_i$

令 $y = \{x_1, y_2, y_3, \ldots, y_n\}$ 则有

$$\sum_{i=2}^{n} y_i + x_1 > \sum_{i=2}^{n} x_i + x_1 = \sum_{i=1}^{n} x_i$$

这说明 $y$ 是比 $x$ 更优的解，与已知矛盾，假设不成立，所以 $\{x_2, x_3, \ldots, x_n\}$ 是上述子问题的最优解。

# 4 算法步骤

## 4.1 伪代码

**算法 1** 求解**单位时间作业安排问题**

**输入:** 作业的截止时间数组 $d[]$，作业对应的惩罚值数组 $w[]$，作业的个数 $n$

**输出:** 单位时间作业安排后的最小损失值

```
1: function TaskArrange(d[], w[], n)
2:     sortd[], w[]bythew[]
3:     finished[n] ← 0
4:     totalReduce = 0
5:     for i = 0 → n do
6:         for j = d[i] → 0 do
7:             if finished[j] = 0 then
8:                 finished[j] ← 1
9:                 break;
10:            end if
11:        end for
12:        if j = 0 then
13:            totalReduce+ = w[i]
14:        end if
15:    end for
16:    return allReduce − totalReduce
17: end function
```

## 4.2 代码实现

在代码实现的时候，我们为了能打印更多的信息，引入了 $no[]$ 数组，记录了对应排序完的任务原先的编号。这样我们就可以通过对 $no[]$ 数组的查询找回到被排序的任务数组原先的值。

为了简化篇幅，这里只给出对应方法功能的实现，案例和对结果的打印以及更加详细的源代码信息，请查看附件中的.cpp 文件或是通过 GitHub 找到对应的.cpp 文件。

```
1 int greed_algo(int * d, int * w, const int n)
2 {
3     int tmp, totalReduce = 0, i, j;
4     int finished[24+1];
5     int no[N];
6     memset(finished, 0, sizeof(finished));
```

```
7       for(i=0;i<n;i++) no[i] = i+1;
8       /*--------------- sort array w and d by w ---------------*/
9       for(i=0;i<n-1;i++)
10      {
11          for(j=0;j<n-i-1;j++)
12          {
13              if(w[j] < w[j+1])
14              {
15                  tmp = w[j];
16                  w[j] = w[j+1];
17                  w[j+1] = tmp;
18
19                  tmp = d[j];
20                  d[j] = d[j+1];
21                  d[j+1] = tmp;
22
23                  tmp = no[j];
24                  no[j] = no[j+1];
25                  no[j+1] = tmp;
26              }
27          }
28      }
29      /*--------------- arrange work ---------------*/
30      for(i=0;i<n;i++) // the rank of the task
31      {
32          for(j=d[i];j>0;j--) // the time
33          {
34              if(finished[j] == 0)
35              {
36                  finished[j] = no[i];
37                  break;
38              }
39          }
40          if(j == 0) totalReduce += w[i];
41      }
42      return totalReduce;
43  }
```

# 5 实验结果及复杂性分析

这里给出两个测试案例，以及对应的运行的结果：

```
int n = 8;
int d[N] = {10, 8, 9, 3, 2, 5, 3, 8};
int w[N] = {5, 10, 10, 3, 2, 5, 8, 9};
```

```
PS E:\Users\Lenovo\Desktop\algorithm design and analysis\3> g++ .\greedalgo.cpp
PS E:\Users\Lenovo\Desktop\algorithm design and analysis\3> .\a.exe
no=2    t=8     * d=8  w=10
no=3    t=9     * d=9  w=10
no=8    t=7     * d=8  w=9
no=7    t=3     * d=3  w=8
no=1    t=10    * d=10 w=5
no=6    t=5     * d=5  w=5
no=4    t=2     * d=3  w=3
no=5    t=1     * d=2  w=2
Table of (Task No)/(The Time to solve It)-----------
Task No:        2       3       8       7       1       6       4       5
Time    :       8       9       7       3       10      5       2       1
Table of (Time)/(The Task we need to solve in that time)-----------
Time:   1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19
20      21      22      23      24
T No:   5       4       7       0       6       0       8       2       3       1       0       0       0       0       0       0       0       0       0
0       0       0       0       0
----------------
totalReduce:0
```

图 1: 单位时间作业安排问题的测试案例 1

```
int d[N] = {
    2, 10, 15, 6, 7, 19, 14, 17, 12, 16,
    24, 3, 18, 18, 3, 12, 9, 21, 21, 21,
    15, 16, 9, 2, 17, 18, 19, 20, 9, 20,
};
int w[N] = {
    21, 22, 14, 17, 20, 19, 20, 15, 21, 30,
    20, 23, 12, 11, 19, 13, 16, 10, 14, 13,
    12, 14, 11, 18, 18, 19, 15, 14, 13, 10,
};
```

```
PS E:\Users\Lenovo\Desktop\algorithm design and analysis\3> g++ .\greedalgo.cpp
PS E:\Users\Lenovo\Desktop\algorithm design and analysis\3> .\a.exe
no=10   t=16    * d=16 w=30
no=12   t=3     * d=3  w=23
no=2    t=10    * d=10 w=22
no=1    t=2     * d=2  w=21
no=9    t=12    * d=12 w=21
no=5    t=7     * d=7  w=20
no=7    t=14    * d=14 w=20
no=11   t=24    * d=24 w=20
no=6    t=19    * d=19 w=19
no=15   t=1     * d=3  w=19
no=26   t=18    * d=18 w=19
no=25   t=17    * d=17 w=18
no=4    t=6     * d=6  w=17
no=17   t=9     * d=9  w=16
no=8    t=15    * d=17 w=15
no=27   t=13    * d=19 w=15
no=3    t=11    * d=15 w=14
no=19   t=21    * d=21 w=14
no=22   t=8     * d=16 w=14
no=28   t=20    * d=20 w=14
no=16   t=5     * d=12 w=13
no=20   t=4     * d=21 w=13
Table of (Task No)/(The Time to solve It)-----------
Task No:        10      12      2       1       9       5       7       11      6       15      26      24      25      4       17      8       27      3
19      22      28      16      20      29      13      21      14      23      18      30
Time    :       16      3       10      2       12      7       14      24      19      1       18      0       17      6       9       15      13      11
21      8       20      5       4       0       0       0       0       0       0       0
Table of (Time)/(The Task we need to solve in that time)-----------
Time:   1       2       3       4       5       6       7       8       9       10      11      12      13      14      15      16      17      18      19
20      21      22      23      24
T No:   15      1       12      20      16      4       5       22      17      2       3       9       27      7       8       10      25      26      6
28      19      0       0       11
----------------
totalReduce:97
```

图 2: 单位时间作业安排问题的测试案例 2

这两个案例前者是一个比较简单的例子，用于说明算法的正确性的，后者是更加符合题意的一个案例，具有足够多（30 个）的作业需要考虑，这是明明肯定做不完的。所以他必须采用策略选择出一个可以使得总损失值最小的安排。

时间复杂度：$O(n^2)$，主要的时间都消耗在排序上了，这里我们采用的是冒泡排序，时间复杂度会较高一点，我们也可以采用具有归并排序或快速排序使得算法的时间复杂度下降为 $O(nlog_n)$。

空间复杂度：$O(1)$，我们只采用了一个 $finished[24]$ 来记录当前的选择情况，而 $no[]$ 则是为了我们打印结果更加方便而额外准备的，所以只从结果上我们可以不考虑 $O(n)$ 的 $no[]$。

## 6 讨论

这个问题在求解过程中我们就可以感受到，这个问题也是可以采用动态规划的思想求解的，这里给出动态规划求解的递推表达式：

设 $dp[i,d]$ 为当截止时间是 $d$ 时，对于任务 $\{1,2,\ldots,i\}$ 具有的最小的总损失值。

$$dp[i,d] = min\{dp(i-1,d) + w[i], dp(i-1, min\{d,d_i\} - t_i)\}$$

当然，这里对于动态规划的解法，其最优子结构的证明和贪心算法的证明一样，而其重叠子问题可以通过递推表达式显而易见。这里也就不给出具体证明和算法的实现了。