算法设计与分析第三次作业参考答案

答案仅供参考，合理即酌情给分

整体评分思路：主要看**递推公式，标记函数**的正确性，函数的初值和条件是否写清，整体思路正确的情况下每处错误扣一分，累加不超过每部分评分规则的上限；非动规算法若思路正确，在复杂度不高于动规算法时也可酌情给分；

3.3

15%（递推公式 4%，标记函数 3%，伪码 5%，复杂度 3%）

3.3 类似于 0-1 背包问题，库房的长度相当于背包的重量限制，每个货柜的收益相当于物品的价值. 于是问题是：

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} l_i x_i \leqslant D, \quad x_i = 0,1$$

令 $C[k,y]$是只允许装前 k 个货柜，库房长度为 y 时的最大收益，那么有

$$C[k,y] = \begin{cases} C[k-1,y] & y < l_k \\ \max\{C[k-1,y], C[k-1,y-l_k]+v_k\} & D \geqslant y \geqslant l_k \end{cases}, \quad k > 1$$

$$C[1,y] = \begin{cases} v_1 & y \geqslant l_1 \\ 0 & y < l_1 \end{cases}$$

算法的伪码是：

```
Store
输入：数组 L[1..n],V[1..n],D      //L 和 V 是货柜长度和价值序列，D 为库房长度
输出：最大的收益 C[n,D]
1. for y←1 to D
2.     C[1,y]←V[1]
3. for k←2 to n
4.     for y←1 to D
5.         C[k,y]←C[k−1,y]
6.         i[k,y]←i[k−1,y]

7.         if y≥L[k] and C[k−1,y−L[k]]+V[k]>C[k−1,y]
8.         then C[k,y]←C[k−1,y−L[k]]+V[k]
9.             i[k,y]←k
```

算法在第 1 行时间为 $O(D)$，第 3 行和第 4 行的循环进行 $O(nD)$次，循环内部是常数时间的操作，于是算法最坏情况下的时间复杂度是 $O(nD)$.

3.5

20%（递推公式 4%，标记函数 3%，伪码 5%，复杂度 3%，具体实例分析 5%）

3.5 设 x_i 表示第 i 种硬币使用的个数，$i=1,2,\cdots,n$. 该问题的描述为

$$\min \sum_{i=1}^{n} w_i x_i$$

$$\sum_{i=1}^{n} v_i x_i = y \quad x_i \text{ 为非负整数}$$

令 $F_k(x)$ 表示只允许使用前 k 种钱，总付款为 x 时所使用零钱的最轻重量，则

$$F_k(x) = \min\{F_{k-1}(x), F_k(x-v_k)+w_k\}, \quad k>1,\ 0<x\leqslant y$$

$$F_1(x) = w_1\left\lfloor \frac{x}{v_1} \right\rfloor = w_1 x$$

$$F_k(0) = 0$$

$$F_k(x) = +\infty, \quad x<0$$

设立标记函数 $t_k(y)$ 记录 $F_k(y)$ 取得最小值时最大币值的标号是否为 k.

$$t_k(x) = \begin{cases} k & F_k(x-v_k)+w_k \leqslant F_{k-1}(x) \\ t_{k-1}(x) & \text{否则} \end{cases}, \quad k>1,\ 0<x\leqslant y$$

$$t_1(x) = 1, \quad 0<x\leqslant y$$

$$t_k(0) = 0, \quad k=1,2,\cdots,n$$

算法的伪码是：

```
Coin
输入：w[1..n],v[1..n],y      //w,v 分别为硬币的重量和币值数组，y 是付款数
输出：F[i,j],t[i,j] i=1,2,…,n,j=1,2,…,y
1. for j←1 to y do
2.     F[1,j]←j * w[1]
3.     t[1,j]←1
4. for i←2 to n do
5.     for   j←1 to y do
6.         F[i,j]←F[i−1,j]
7.         t[i,j]←t[i−1,j]
8.         if F[i,j−v[i]]+w[i]≤F[i−1,j]
9.         then F[i,j]←F[i,j−v[i]]+w[i]
10.                t[i,j]←i
11. return 二维数组 F,t
```

算法的时间复杂度主要取决于第 4 行和第 5 行的 for 循环，内部工作量是常数时间，算法的时间是 $O(ny)$. 通过数组 t 追踪解的过程比较简单，时间不超过 $O(ny)$. 于是算法最坏情况下的时间复杂度是 $O(ny)$.

针对给定实例，算法的备忘录如表 3.5 和表 3.6 所示.

表 3.5 $F_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	1	2	3	2	3	4	5	4	5	6	7	6
3	1	2	3	2	3	4	5	4	5	6	7	6
4	1	2	3	2	3	4	5	4	5	6	7	6

表 3.6 $t_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2	2	2	2	2	2
3	1	1	1	2	2	3	3	2	2	3	3	2
4	1	1	1	2	2	3	3	2	2	3	3	2

问题实例的解是：最轻重量是 6，由 $t_4(12)=2$ 知道 $x_4=0, x_3=0, x_2\geqslant 1$，再由

$$t_2(12-4) = t_2(8) = 2 \Rightarrow x_2 \geqslant 2$$

$$t_2(8-4) = t_2(4) = 2 \Rightarrow x_2 \geqslant 3$$

$$t_2(4-4) = t_2(0) = 0 \Rightarrow x_2 = 3, \quad x_1 = 0$$

从而得到 $x_1=x_3=x_4=0, x_2=3$，只用了 3 枚币值为 4 的硬币.

3.8

15%（算法正确性：递推公式 5%，标记函数 5%，复杂度 5%）

3.8 令 $B=\lfloor N/2 \rfloor$，那么 $\min\{\sum_{a_i \in A_1} a_i, \sum_{a_j \in A_2} a_j\} \leqslant B$. 不妨设 $\sum_{a_i \in A_1} a_i \leqslant \sum_{a_j \in A_2} a_j$. 那么问题变成求 A 的子集 A_1 使得

$$\max \sum_{a_i \in A_1} a_i$$

$$\sum_{a_i \in A_1} a_i \leqslant B$$

那么该问题等价于双机调度问题(习题 3.4). 通过动态规划算法求出最优解 A_1，如果 A_1 中的数之和 $\sum_{a_i \in A_1} a_i = B$，那么输出“Yes”，否则输出“No”. 该算法的时间复杂度为 $O(nN)$.

令 F[k, y]为从 A 中前 k 个元素中取若干元素，在不大于 y 的情况下能够得到的最大和，则 F[n,B]=B 则可以，否则不行

递推公式:

$$F[k,y] = max\{F[k-1,y], F[k-1,y-a_k] + a_k\}, k > 1, y \leq B$$
$$F[1,y] = a_1 \; if \; a_1 \leq y \; else \; 0.$$

$$F[\mathrm{k},0] = 0, \qquad F[\mathrm{k},\mathrm{y}] = -\infty$$

标记函数 i[k,y]:

$$i[k,y] = \begin{cases} i[k-1,y], & if \; F[k-1,y] > F[k-1,y-a_k] \\ k, & else \end{cases}, \quad k > 1$$

$$i[1,y] = \begin{cases} 0, & a_1 < y \\ 1, & a_1 \geq y \end{cases}$$

其余解法思路正确也可给分

3.10

20%（递推公式 10%，标记函数 5%，复杂度 5%，考虑成非 0-1 的情况若算法正确可酌情给分）

3.10 设 x_i 表示装入背包的第 i 种物品个数，$i=1,2,\cdots,n$，推广的 0-1 背包问题的描述是

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leqslant W$$

$$\sum_{i=1}^{n} c_i x_i \leqslant V$$

$$x_i = 0,1$$

设 $m[i,j,k]$表示使用前 i 种物品、背包重量限制为 j、容积为 k 时的最大价值，其中 $i=1,2,\cdots,n, j=1,2,\cdots,W, k=1,2,\cdots,V$，那么递推方程是：

$$m[i,j,k]=\begin{cases}\max\{m[i-1,j,k], m[i-1,j-w_i,k-c_i]+v_i\} & \text{若 } j\geqslant w_i \text{ 且 } k\geqslant c_i\\ m[i-1,j,k] & \text{否则}\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$m[1,j,k]=\begin{cases}v_1 & \text{如果 } j\geqslant w_1 \text{ 且 } k\geqslant c_1\\ 0 & \text{否则}\end{cases}$$

$$m[i,0,k]=m[i,j,0]=0$$

$$m[i,j,k]=-\infty \quad j<0 \text{ 或 } k<0$$

与前面的背包问题类似，可定义标记函数 $t[i,j,k]$用于追踪问题的解．其中

$$t[i,j,k]=\begin{cases}i & \text{如果 } m[i-1,j,k]\leqslant m[i-1,j-w_i,k-c_i]+v_i, j\geqslant w_i, k\geqslant c_i\\ t[i-1,j,k] & \text{否则}\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$t[1,j,k]=\begin{cases}1 & \text{如果 } j\geqslant w_1 \text{ 且 } k\geqslant c_1\\ 0 & \text{否则}\end{cases}$$

该算法最坏情况下的时间复杂度是 $O(nWV)$.

3.15

15%(算法设计 5%, 递推方程和标记函数 5%, 复杂度 5%; 算法复杂度不超过 O(n^3)即可给满分)

3.15 **方法一**：设 $F[j]$表示第 $1,2,\cdots,j$ 天加工任务的最大数目. 如果在第 i 天进行检修，那么从第 $i+1$ 天到第 j 天连续工作的加工量用 $w[i+1,j]$表示，则

$$w[i+1,j]=\sum_{k=i+1}^{j}\min\{x_k,s_{k-i}\},\quad 0\leqslant i<j\leqslant n$$

不难看出，$w[i+1,j]$满足如下递推方程：

$$w[i+1,j]=w[i+1,j-1]+\min\{x_j,s_{j-i}\},\quad 0\leqslant i<j\leqslant n,\ j>1$$

$$w[i+1,i+1]=\min\{x_{i+1},s_1\},\quad i=0,1,\cdots,n$$

假设最后一次检修机器在第 i 天，那么从第 $i+1$ 天到第 j 天连续加工任务总数为 $w[i+1,j]$，而检修前加工任务数至多是 $F[i-1]$. 于是当最后一次检修发生在第 i 天时，到第 j 天为止，加工任务数至多是 $F[i-1]+w[i+1,j]$. 考虑到所有可能的检修时间 $i(i=1,2,\cdots,j-1)$，或者在第 j 天之前根本不做检修的情况，可以得到如下递推关系：

$$F[j]=\max\begin{cases}\max\limits_{0<i<j}\{F[i-1]+w[i+1,j]\} & j>1\\ w[1,j] & j>0\end{cases}$$

$$F[0]=0$$

可以定义标记函数记录得到最优 $F[j]$时的检修时间 i. 如果每次都重新计算 $w[i+1,j]$，算法最坏情况下的时间复杂度是 $O(n^3)$. 如果在预处理时根据 $w[i.j]$的递推方程计算所有的 $w[i,j]$，$1\leqslant i\leqslant j\leqslant n$，并把计算结果存成备忘录，在计算 $F[j]$时直接查找相应的值，那么计算 $F[j]$的时间可以降为 $O(n^2)$，而预处理的时间也是 $O(n^2)$，因此算法最坏情况下的时间复杂度是 $O(n^2)$.

方法二：参照矩阵链相乘的方式划分子问题，该算法的时间复杂度高一些.

首先证明下述命题.

命题 3.2 在最优解中不会出现连续两天检修的情况.

证 假若不然，在一个最优解 T 的第 i 和 $i+1$ 天都进行检修. 那么，当去掉第 i 天的检修后，不会减少从第 $i+1$ 天到第 n 天的加工量；也不会减少第 1 天到第 $i-1$ 天的加工量；只有第 i 天的加工量由 0 增加到 $\min\{x_i,s_t\}$，其中 s_t 表示第 i 天机器具有的加工能力. 于是总加工量将大于原来的加工量，从而与 T 的最优性矛盾.

通过类似的分析还可以证明，检修也不会发生在子问题的最后一天.

利用上述性质，可以从检修的时间点 k 将原问题划分成两个子问题. 设 $G[i,j]$表示第 i 天到第 j 天的最大加工任务数. 如果在第 k 天进行检修，$i<k<j$，那么最大加工任务数等于 $G[i,k-1]+G[k+1,j]$. 如果在第 i 天到第 j 天的时间内没有发生检修，那么加工任务数是 $w[i,j]$，于是得到如下递推方程：

$$G[i,j]=\max\{w[i,j],\max_{i<k<j}\{G[i,k-1]+G[k+1,j]\},\quad 1\leqslant i<k<j\leqslant n$$

$$G[i,i]=w[i,i],\quad i=1,2,\cdots,n$$

标记函数的设定可参照矩阵链相乘问题的做法，只是当最优解的第 i 到第 j 天之间没有进行检修时将标记函数设为 0. 该算法最坏情况下的时间复杂度是 $O(n^3)$.

3.17

15%（递推方程 5%，标记函数 5%，复杂度 5%）

3.17 令 $F[i,j]$表示 a_{11}到 a_{ij}的路径上的数的最小和，则

$$F[i,j]=\min\{F[i-1,j],F[i-1,j-1]\}+a_{ij},\quad i=2,3,\cdots,n,\ j=2,3,\cdots,i-1$$

$$F[i,1]=F[i-1,1]+a_{i1},\quad i=2,3,\cdots,n$$
$$F[i,i]=F[i-1,i-1]+a_{ii},\quad i=2,3,\cdots,n$$
$$F[1,1]=a_{11}$$

问题的最优路径上的和是

$$\min\{F[n,j]\mid j=1,2,\cdots,n\}$$

可以定义标记函数 $k[i,j]$，记录得到最优 $F[i,j]$时的路径选择，即

$$k[i,j]=\begin{cases}j & \text{若 } F[i-1,j]<F[i-1,j-1]\\ j-1 & \text{否则}\end{cases}$$

算法的时间复杂度为 $O(n^2)$.