算法设计与分析第三次作业

4.1

（1）贪心策略：按照服务时间 Ti 从小到大顺序服务。
（2）算法描述：对 Ti 进行从小到大排序，依次服务。

```
sort(T[],1,n)
time:=0
wait_time:=0
For i:=1 TO n:
    serve(S[i]);
    time:=time+T[i]
    wait_time:=wait_time+(n-i)*T[i]
return wait_time//总等待时间
```

（3）T[]={1,3,2,15,10,6,12},服务顺序{1,2,3,6,10,12,15},等待时间依次是{1*6,2*5,3*4,6*3,10*2,12*1}，总等待时间 78.

4.2

（1）可行的贪心策略：按照文件大小 Pi 从小到大顺序存储。

证明：

不妨令最终能够存储 M 个文件。只需证明按照贪心策略得到的“T={Pi|Pi 为前 M 小的文件}为最终装入文件的集合”为可行解即可。

假设最终装入的文件集合为 S，且 S≠T。

∵Count（S）=Count（T）=M ∴ ∃Pk∈T，Pk ∉ S；且 ∃ Pj∈S，Pj ∉ T。

∵Pk ∉ S， ∴Pk 不是前 M 小的文件∴Pk 一定大于等于 S 中任一元素，也大于等于 Pj∴将 Pk 换成 Pj，∑P 不会增大，也不会超过 C∴S-Pk+Pj 也是可行解。

持续进行上述过程，直到¬∃Pk∈T，Pk ∉ S，此时显然有 S=T。

∴T 也是可行解。

命题得证。

（2）采用动态规划。

```
F[i][j]:从前 i 个文件选取文件能够占据 j 存储空间则为 true。
状态转移方程：F[i][j]=F[i-1][j]||F[i-1][j-Pi]
//init F[]
for j:=C DOWNTO 0
    F[j]:=false
F[0]:=true
for I:=1 TO n
    for j:=C DOWNTO 0
            if(F[j-Pi])
                    F[j]:=true
//
```

```
for j:=C DOWNTO 0
    if(F[j])
            return C-j
```

时间复杂度 O(n*C)

4.9

贪心策略：按照结束时间从小到大排序，每次取最小结束时间 d[i]插入观测点，更新尚未测试的任务集合。

伪码：

```
Num:=0
//按照结束时间从小到大顺序排序 p[]
sort(p[])
//初始化
for i:=1 TO n
    Tested[i]=false
//线性扫描
for i:=1 TO n
    if !Tested[i]
            Num:=Num+1 //测试次数增加
            For  j:=i TO n
                    If s[j]<=d[i]
                            Tested[j]=true
return Num
```

时间复杂度：O（n*n）

归纳证明：任一时刻 t 采用上述算法所需要的测试次数都是最少的。

- 0 时刻，如果存在任务 p[i]且 d[i]=0,则测试次数为 1；否则为 0。显然不存在更少的测试次数。
- 假设 t1 时刻上述算法所用测试次数最少。
- 则 t1+1 时刻，按照上述算法，当且仅当未测试任务集合中存在任务 p[i]的结束时间 d[i]=t1+1，才需要测试次数加 1。

  如果存在这样的任务 p[i]，如果不增加一次测试，显然会错过任务 p[i]，不可行。

  如果不存在这样的任务 p，则不需要增加一次测试。

  综合这两种情况，采用这种算法在 t1+1 时刻所用的测试次数依然是最少的。

4.15

（1）贪心策略：对任务按照 b[i]时间从大到小在 A 机器上加工，然后选择新机器 B 继续加工。
（2）最坏时间的时间复杂度为 O（n*lgn），主要为 b[i]排序的时间。
证明：

- 引理 1：假设存在在 A 上加工的任务 i(在 A 上开始时间为 t）和下一个任务 j,满足 b[i]<b[j]。交换他们在 A 上加工顺序不会使最终总时间变长：

```
在 Q=∑a[i]时刻,所有任务刚好都在 A 机器完成
则最终完成时间为 Max(Q,Min(e[i]+b[i])),其中 e[i]为任务 i 在机器 A 上完成时的时间。
```

对于在机器A上相邻的任务i、j：
e[j]=e[i]+a[j]=t+a[i]+a[j].
b[j]+e[j]=b[j]+t+a[i]+a[j]>b[i]+e[i]=b[i]+t+a[i]
如果将任务i和j在机器A上调换执行顺序，则有
b[j]+e[j]=b[j]+t+a[j]<b[j]+t+a[i]+a[j].b[i]+e[i]=b[i]+t+a[i]+a[j]<b[j]+t+a[i]+a[j].显然Min(e[i]+b[i])减小，从而最终完成时间Max(Q,Min(e[i]+b[i]))不会变长。

- 引理2：假设机器A执行序列上存在任务i和后续某任务j，满足b[i]<b[j]，则交换他们两执行顺序不会使得最终完成时间变长。

∵交换任务i、j，可以通过有限次交换相邻任务的步骤来完成，且每一步的最终完成时间都不会变长。（根据引理1）
∴交换任务i、j，最终完成时间不会变长。

- 反证结论

假设存在一个解：机器A的任务执行序列为S[]，S不是b[i]递减的，且最终完成时间t1小于沿b[i]递减的A执行序列的完成时间t2。
根据引理2，只要S[]上存在逆序的任务b[i]<b[j]，就可以交换顺序，将任务j放到任务i之前，并且最终完成时间t<=t1。
反复进行交换，直到得到沿b[i]递减的A执行序列，此时最终完成时间t2<=t1。与题设(t1<t2)矛盾。

5.1

```
for x2:=0 TO MAX :
    if 4*x2>12 break
    for x1:=0 TO  MAX :
        if 4*x2+3*x1>12 break
        for x3:=0 TO MAX :
            if 4*x2+3*x1+2*x3>12 break
            print x1,x2,x3
```

解：

(0,0,0), (0,0,1), (0,0,2), (0,0,3), (0,0,4), (0,0,5), (0,0,6), (1,0,0), (1,0,1), (1,0,2), (1,0,3), (1,0,4), (2,0,0), (2,0,1), (2,0,2), (2,0,3), (3,0,0), (3,0,1), (4,0,0), (0,1,0), (0,1,1), (0,1,2), (0,1,3), (0,1,4), (1,1,0), (1,1,1), (1,1,2), (2,1,0), (2,1,1), (0,2,0), (0,2,1), (0,2,2), (1,2,0), (0,3,0)

5.2

有3^4种组合。采用回溯剪枝快速穷举所有可能组合。伪码如下：

```
MinW:=MAX //保存最小重量
func(i,v,w)://处理第i个配件之前总重量为w，总价值v
    if w>MinW||v>120 //剪枝
```

```
        return
    if i>4  //到达根节点
        if w<MinW
            MinW=w //更新结果
        return
    for j:=1 TO 3
        func(i+1,v+value[i][j],w+weight[i][j])
            //value[i][j]:配件 i 在供应商 j 的价值,weight[i][j]:配件 i 在 j 的重量
func(1,0,0)
```

最小重量 31,配件分别对应供货商 3，1，2，3。