第五章动态规划练习参考答案

一、习题五

1． 解：

1) 第一层 for 循环 n，内层循环 n,n-1,…1, 所以 $T(n)=1=2+\ldots+n=n(n-1)/2=O(n^2)$

2) 分治算法：和最邻近点对的算法类似，我们可以在 k=n/2 的位置将 $A=<a_1,a_2,\ldots a_n>$划分为 A_1 和 A_2 两半，于是 A 的最大子段和可能是三种情况：出现在 A_1 部分、出现在 A_2 部分、出现在横跨两边的中间部分。前两种情况恰好对应两个规模减半的子问题，第三种情况,设最大子段为[p..q]，则一定 p≤k,q≥k+1，只需从 A[k]和 A[k+1]分别向前和向后求和，记下其最大的和 S1,S2，则 S1+S2 就是横跨中心的最大和。算法如下：

```
MaxSubSum(A,left,right){
输入：数组 A,left、right 分别是 A 的左右边界；
输出：A 的最长子段和 sum 及子段的前后边界。
if left=right then return max(A[left],0)
k:=(left+right)/2
leftsum:=MaxSubSum(A,left,k)
Rightsum:=MaxSubSum(A,k+1,right)
S1:=A1[k]    //从 k 向左的最大和,参考 1)内循环改为 for j=k to 1
S2:=A2[k+1] //从 k 向右的最大和,求 s1,s2 需扫描整个数组 O(n)
Sum:=s1+s2
```

If leftsum>sum then sum:=leftsum

If rightsum>sum then sum:=rightsum

}

算法复杂度：T(n)=2T(n/2)+O(n)=O(nlogn)(主定理之情形 2)

3)这个问题如何确定子问题是个有意思的事情，需要认真考虑。如我们可能很自然设 C[i]=A[1..i]的最大和，但它不满足最优子结构。因为使 A[1..i]达到最大和的子段未必包含 A[i]，计算后面的问题不能直接把该子段直接组合进来。如 A=<2,-5,8, 1,-9,4,6>，C[5]=8+1=9，在计算 C[6]时不能把<8,1>直接组合进来，因为后面有-9，对连续和也有影响。例中，C[7]=4+6，C[6]显然不是 4，而是 8+1。所以这样定义的 C 不满足最优子结构。

按提示，定义 b(j)= $\max\limits_{1\le i\le j}\{\sum\limits_{k=i}^{j} A[k]\}$ 为输入 A[1..j]中必须包含 A[j]的最大子段和，b[j]满足最优子结构性质：若 b[j+1]最大，则 b[j]必然最大，从 b[j+1]= $\max\limits_{1\le i\le j+1}\{\sum\limits_{k=i}^{j+1} A[k]\} = \max\limits_{1\le i\le j}\{\sum\limits_{k=i}^{j} A[k] + A[j+1]\}$ 得 b[j+1]=b[j]+A[j+1]即可很易证明。

据此可得递推公式：b[j]=max{b[j-1]+A[j],A[j]},j=1,2,..n；b[0]=0.

这里还有一个问题：b[j]并不是 A[1..j]的最大子段和，只是包含 A[j]的最大子段和。但我们已得到 b[1],b[2],…b[n]，恰好枚举了以任何元素为最后元素的的所有子段的最大和，A[1..n]的最大子段和一定在其中：对 n 个和比较取最大者即可。

算法描述：

```
MaxSum(A,n){
输入：数组 A；输出：组大字段和 sum，子段的最后位置 c
Sum:=0
b:=0
for i:=1 to n do
    if b>0 then b:=b+A[i]
    else b:=A[i]
    endif
    if b>sum then
      sum:=b
      c:=i
    endif
end {for}
return sum,c
}
```

尚有一点工作：从 sum，c 找到子段的左边界(略)。

算法复杂度：显然 T(n)=O(n)。

注意:很多同学,使用,$ta_i=max(t_a+a(i),t_b)$,$tb_i=max(t_a,t_b+b(i))$,即对第 i 个任务以谁加工总长最短来安排，if $Ta_i<Tb_i$ A 加工 else B 加工

这是贪心算法，不正确，第 10 章有多机调度近似算法、反例。

又如反例：n=3，(a1,a2,a3)=(2,5,6)，(b1,b2,b3)=(3,6,3),贪心调度为 $A_1B_2A_3$，

$max=8$；更优 $A_1A_2B_3,max=7$。

2. 解：

在完成前k个作业时，设机器A工作了x 时间，则机器B此时最小的工作时间是x的一个函数。

设F[k][x]表示完成前k个作业时，机器B最小的工作时间，则

$$F[k][x]=\min\{F[k-1][x]+b_k,F[k-1][x-a_k]\}$$

其中$F[k-1][x]+b_k$对应第 k个作业由机器 B 来处理（完成 k-1 个作业时机器A的工作时间仍是 x，则 B在 k-1 阶段用时F[k-1][x]；而$F[k-1][x-a_k]$对应第 k个作业由机器A处理（完成k-1 个作业，机器A 工作时间是x-a[k]，而B完成k阶段与完成k-1 阶段用时相同为$F[k-1][x-a_k]$。

则完成前 k 个作业所需的时间为：$\max\{x,F[k][x]\}$。

根据递推关系，很容易证明问题满足最优子结构性质。

解为：$\min(\max(x,f(N,x)),0<=x<=\Sigma a(i)$。

当处理第一个作业时，a[1]=2,b[1]=3;

机器A 所花费时间的所有可能值范围：$0\leqslant x \leqslant a[0]$.

$x<0$ 时，设$F[1][x]= \infty$，则$\max(x, \infty)= \infty$;

$0\leqslant x<2$ 时，$F[1][x]=3$，则$Max(0,3)=3$，

$x\geqslant 2$ 时， $F[1][x]= 0$，则$Max(2,0)=2$;

2）处理第二个作业时：x 的取值范围是：$0 <= x <= (a[0] + a[1])$，

当 $x<0$ 时，记 $F[2][x] = \infty$；以此类推下去(略)。

第二种解法：设（A，B）数对表示机器 A、B 的时间，则到第 i 个任务时，可能的安排为S_i，有上述数对集合组成。

$S_0=\{(0,0)\}$，$S_i=\{S_{i-1}+(a_i,0)\}\cup\{S_{i-1}+(0,b_i)\}$

递推求出S_n，求 min(max(A,B))即可。

可以通过贪心算法求得上届剔除部分数对，也可以通过比较数对间关系剔除部分数对。（实现略）

3. 解：

这是整数背包问题。类似0/1背包问题，易证满足最优子结构性质：设 $y_1,y_2,..,y_n$ 是原问题的最优解，则 $y_1,y_2,..,y_{n-1}$ 是下述子问题：

$$\max\sum_{i=1}^{n-1}c_i x_i \text{ , S.T. } \sum_{i=1}^{n-1}a_i x_i \le b-a_n y_n \text{ ,} x_i\in\{0,1,2\},1\leqq i\leqslant n-1$$

的最优解。如不然，$y'_1,y'_2,\cdots,y'_{n-1}$是子问题的最优解，则

$\sum_{i=1}^{n-1}c_i y'_i > \sum_{i=1}^{n-1}c_i y_i$,则$\sum_{i=1}^{n-1}a_i y'_i + a_n y_n \le b$, $y'_1,y'_2,\ldots,y'_{n-1},y_n$ 将是原问题的可行解解，且$\sum_{i=1}^{n-1}c_i y'_i + c_n y_n > \sum_{i=1}^{n}c_i y_i$，与 $y_1,y_2,\ldots,y_n$ 是最优解矛盾。

递推公式：设 m[k][x]表示容量约束 x，可装入 1,2,…k 件物品的最优解，则

$m[k][x]=\max\{m[k-1][x],m[k-1][x-a_k]+c_k,m[k-1][x-2a_k]+2c_k\}$,$0\leqslant x\leqslant b$

$m[0][x]=0$,if $x>=0$;　$m[0][x]=-\infty$,if$x<0$;　$m[k][<0]=-\infty$,$k=1,2,..n$

据此不难给出算法的描述。（略。仿照 0/1 背包问题，但那个程序是从后向前推的，上述递推是从前往后。本问题可以扩展到 x_i 可以取值 0,1,2，…m,，递推部分改为 $m[k][x]=\max\{m[k-1][x-x_i a_k]+x_i c_k\}$，对 0

≤xi≤m 取 max 即可）

4. 解：

问题可描述为：$\max\limits_{1\le i\le n}\Pi g_i(m_i)$， s.t . $\sum\limits_{i=1}^{n} m_i c_i \le C$ 。

下面证明问题满足最优子结构性质。设 $m_1,m_2,\ldots m_{n-1},m_n$ 是原问题的最优解，则其可靠性为 $\prod\limits_{1\le i\le n} g_i(m_i) = g_n(m_n)\prod\limits_{1\le i\le n-1} g_i(m_i)$，那么不难证明 $m_1,m_2,\ldots m_{n-1}$ 是下述问题的最优解：(仿照上述 3：如果不是，存在.....略)。 $\max\limits_{1\le i\le n-1}\Pi g_i(m_i)$，s.t. $\sum\limits_{i=1}^{n-1} m_i c_i \le C - \mathrm{m_n c_n}$ 。

递推关系：设 M[k][x]为成本 x，前 k 级设备串联所得最优可靠性值，则

$m[k][x]=\max\{m[k-1][x-m_k c_k]\times g_k(m_k)\}$，$1\le m_k\le x/c_k$

$m[0][x]=1$，$x>=0$; $m[k][c_k]=1$。

算法描述：

```
Safe(c[],g[],C){      //g[i]()是函数 gi()
Int m[][],p[][]
C=c-Σc(i)    //每级至少一个设备
for (j=0 to C) m[0][j]=1
for (i=1 to n){
   for j=0 to c {
      m[i][j]=0
      for (k=0 to j/c[i]){
         t= m[i-1][j-k*c[i]]g[i](k)
```

```
            if m[i][j]<t    then {
                m[i][j]=t
                p[i][j]=k        //此处可用 p[i][j]记录 k=m_i
        }
      }
  }
```

最优解是 m[n][C]。回溯求 m_i：

$p[n][C]=m_n$;

$C=C-m_n*c[n]$;

$P[n-1][C]=m_{n-1}$； …..

（注：由于 c(i)可能是实数，上述算法是示意，用类似 0/1 背包问题数对的递推方式可能更具一般性）

二、

1.解：

使用动态规划设计技术。对于 i=1,2,…n，考虑以 xi 为最后项的最长递增子序列的长度 C[i]。如果在 xi 前存在 $x_j<x_i$，则 C[i]=max{C[j]}+1,；否则 C[j]=1。显然，若 C[i]的子序列是 $i_1i_2…i_ki$，则 $C[i_k]$的子序列是 $i_1i_2…i_k$，满足最优子结构性质。因此 C[1]=1，对 i>1，

C[i]= C[i]=max{C[j]}+1,存在 j,1<=j<l,$x_j<x_i$；或 C[i]=1，任给 j，1<=j<l,$x_j>x_i$；所求最长子序列的长度是 C=max{C[i]|i=1,2…n}。

算法设计时用 K[i]记录 C[i]最大时的 j，C[i]=1 时，K[i]=0；则从 K[]

可追踪出解序列：K[i],K[K[i]]….。算法描述略，参考习题五、1,3）。

算法复杂度：对每个 i 需要检索比 i 小的所有 j，需 O(n)时间，i 取值 n 种，所以 $T(n)=O(n^2)$。

对于给定的实例 A=<2,8,4,-4,5,9,11> ，计算过程如下：

C[1]=1，K[1]=0

C[2]=max{C[1]+1)=2，K[2]=1

C[3]=max{C[1]+1}=2，K[3]=1

C[4]=1, K[4]=0

C[5]=max{C[1]+1,C[3]+1,C[4]+1}=3，K[5]=3

C[6]=max{ C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1}=4 ，K[6]=5

C[7]=max{ C[1]+1,C[2]+1,C[3]+1,C[4]+1,C[5]+1,C[6]+1}=5，K[7]=6

C[7]=5 是最大值。子序列追踪过程：11，A[K[7]]=A[6]=9,A[K[6]]=A[5]=5,A[K[5]]=A[3]=4,A[K[3]]=A[1]=2，得解为<2,4,5,9,11>。

注：本题可以有多解。

2．解：与 0/1 背包问题类似，使用动态规划算法。令 N(j,d)表示考虑作业集{1,2,…,j}、结束时间剩余 d 的最优调度的效益，那么

N(j,d)=max{N(j-1,d),N(j-1,d-t(j)+v(j)} , d≥t(j);

N(j,d)=N(j-1,d) , d<t(j) 初值：

N(1,d)=v(1) ，t(1)≤d； N(1,d)=0 ， t(1)>d

N(j,0)=0； N(j,d)=-∞,d<0

自底向上计算，用 B(j,d) 记录 N(j,d) 达到最大时是否

N(j-1,d-t(j))+v(j)>N(j-1,d)，如果是，B(j,d)=j，否则，B(j,d)=B(j-1)。

算法描述：

输入：加工时间 t[1..n]，效益 v[1..n]，结束时间 D

输出：最优效益 N[i,j]，标记函数 B[i,j]，i=1.2..n,j=1,2,..D。

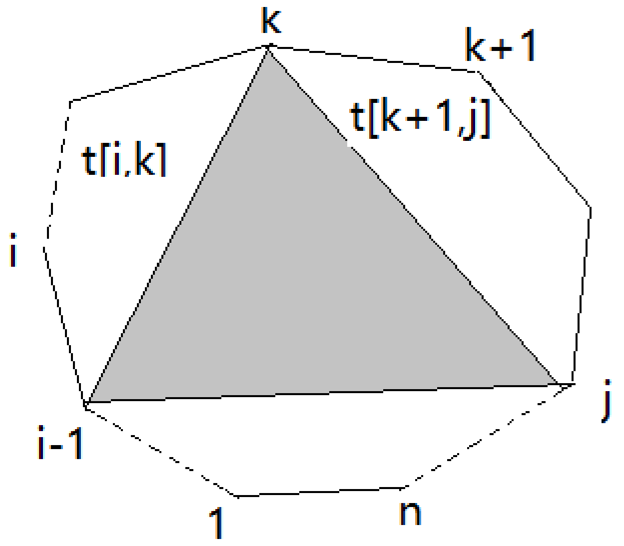
```
for d:=1 to t[1]-1
   N[1,d]:=0; B[1][d]:=0
For d:=t[1] to D
 N[1,d]:=v[1]; B[1][d]=1
For j:=2 to n {
 For d:=1 to D{
   N[j,d]:=N[j-1,d]
   B[j,d]:=B[j-1,d]
   If d>t[j] and N[j-1,d-t[j]]+v[j]>N[j-1,d] then {
    N[j,d]:=N[j-1,d-t[j]]+v[j]
    B[j,d]:=j
   }
 }(ford)
}(fo j)
```

得到最大效益 N[n,D]后，通过对 B[n,D]的追踪，可以得到问题的解。

算法的主要工作是 j、d 循环，需执行 O(nD)次，循环体内关键操作是常数，所以 T(n)=O(nD)。

3.解：

如图所示，n 边形的顶点是 1,2,..,n，顶点 i-1,i,...j 构成的凸多边形记做 A[i,j]，于是原始问题就是 A[2,n]。



考虑子问题 A[i,j]的划分。假设它的所有划分方案中的最小权值是 t[i,j]，从 i,i+1,...,j-1 中任选点 k，它与底边(i-1)j 构成一个三角形，如图中灰色三角形所示。这个三角形将 A[i,j]划分为两个凸多变形：A[i,k]和 A[k+1,j]，从而产生了两个子问题。这两个凸多边形的划分方案的最小权值分别是 t[i,k]和 t[k+1,j]。根据动态规划的思想，A[i,j]相对于这个 k 的划分方案的最小权值是：

$t[i,k]+t[k+1,j]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}$

其中 $d_{(i-1)k}+d_{kj}+d_{(i-1)j}$ 是三角形(i-1)kj 的周长。于是得到递推关系：

$t[i,j]=\min\{t[I,k]+t[k+1,j]+ d_{(i-1)k}+d_{kj}+d_{(i-1)j}\}$ for $i\leqslant k\leqslant j$；当 $i<j$ 时；

$t[i,j]=0$，当 $i=j$ 时。

有了这个递推关系，不难证明，若最优划分 t[i,j]与(i-1)j 相连的三角形第三顶是 k，则这个划分也是 A[i,k]和 A[k+1,j]的最优划分。不难

看出，这个递推关系与矩阵连乘算法的递推式非常相似，定义 s[I,j=k 记录得到最小权值的 k 的位置，算法描述参考矩阵连乘的迭代算法(略)。算法最坏复杂度 $O(n^3)$。