习题参考答案

6.1

$$F(n) = F(n-1) + F(n-2)$$
$$\le 2F(n-1) \le 2^2 F(n-2)$$
$$\le \cdots \le 2^n F(0) = O(2^n)$$

6.2

```
PrintStations(l, j)
1    i← l*
2    l* ← li[j]
3    PrintStations(l, j-1)
4    print “line ” i“, station ” j
```

6.3

分析：

可以不记录 *fi*[*j*]，而用两个变量 *f1* 和 *f2* 来保存原程序中的 *f1*[*j-1*] 和 *f2*[*j-1*]，并在一轮循环中用 *f1*、*f2* 来计算 *f1*[*j*] 和 *f2*[*j*]即新的 *f1*、*f2*。因而可以节省 *2n-2* 个空间，在最后一轮循环后即 j=n 后，使之仍然能够计算出 *f** ，并且仍然能够构造出最快装配路线。 *f1*、*f2* 保存的为 *f1*[*n*] 和 *f2*[*n*]。根据这两个值，就能计算出 *f** 。

6.4

	1	2	3	4	5	6
6	2010	1950	1770	1860	1500	0
5	1655	2430	930	3000	0	
4	405	330	180	0		
3	330	360	0			
2	150	0				
1	0					

	1	2	3	4	5	6
6	2	2	4	4	5	-
5	4	2	4	4	-	
4	2	2	3	-		
3	2	2	-			
2	1	-				
1						

((A1A2)((A3A4)(A5A6)))

6.5

```
MatrixChainMultiply(Ai…j, s, i, j)
1    if i=j then
2        return Ai
3    else
4        q←s[i,j]
5        A1 ← MatrixChainMultiply(Ai…q, s, i, q)
6            A2 ← MatrixChainMultiply(Aq…j, s, q+1, j)
7        return    MatrixMultiply(A1, A2 )
```

6.6

6.7

备忘录方法是一种自顶向下的高效动态规划方法，它可能包含递归过程，并在第一次解决一个子问题时将结果记录到一个表中，在下一次遇见该子问题时，只需查表而无需再解决一次，因而当某个算法有重叠子问题时，能很高效地解决问题。而 MergeSort 算法的每个子问题都是相互独立的，不存在重叠子问题，因而使用备忘录方法并不能提高效率。

6.8

参考课件

6.9

```
PrintLCS(c, X, Y, i, j)
1       if i = 0 or j = 0 then
2            return 0
3       if X[i]=Y[j] then
4            PrintLCS(c, X, Y, i -1, j -1)
5            print x[i]
6       else if c[i-1, j] >= c[i, j-1] then
7                    PrintLCS(c, X, Y, i-1, j )
8            else
9                    PrintLCS(c, X, Y, i, j -1)
```

6.10

```
MemoizedLCSLength(X, Y, m, n)
1            for i←0   to   m   do
2                    for j← 0   to   n do
3                            if i=0 or j=0 then
4                                    c[i, j] ← 0
5                            else
 6                                   c[i, j] ← -1
 7             LookupLCSLength(X, Y, m, n)
 8             return       c and b
LookupLCSLength(X, Y, i, j)
1       if   c[i, j]≥o then
2            return   c[i, j]
3       if x[i]=y[j] then
4            q← LookupLCSLength(X, Y, i-1, j-1)
5            c[i, j] ← q+1
6            b[i, j] ← "↖"
7     else
8            q1← LookupLCSLength(X, Y, i-1, j-1)
9            q2←LookupLCSLength(X, Y, i-1, j-1)
10           if   q1≤q2 then
11                   c[i, j] ← q1
12                   b[i, j] ← "↑"
13           else
```

```
14                      c[i, j] ← q2
15                      b[i, j] ← "←"
```

6.11

反证法。

假设存在一个不以 A 开始的最长公共子序列，设为 Zk=z1z2…z3 则 Zk 不包括 Xm 和 Yn 的开头字符 A。现在将 A 加入到 Zk 的头部，得到的序列 A Zk 必定为 Xm 和 Yn 的公共子序列，且 A Zk 的长度为 k+1，这就与题设中 Zk 为最长公共子序列相矛盾了。

6.12

```
X=(x1, x2, ……xn)
b[i]= max 1≤k≤i-1 {b[k]}          xk≥xi
        max 1≤k≤i-1 {b[k]}+1      xk≤xi
LSC(X)
1       b[1] ←1
2       for   i←2 to n do
3             k←0
4             for j←1 to i-1 do
5                   if    xj≤xi and k<b[j]   then
6                         k←b[j]
7             b[i] ←k+1
8     return   MAX(b)
```

6.13

6.14

p[*i*, *w*]：当背包容量为 *w* 时，可以从物品 1 到 *i* 获得的最大价值

(1)不装入物品 *i*　　p[*i*, *w*]=p[*i*-1, *w*]

(2)装入 1 个物品 i　　p[*i*, *w*]=p[*i*-1, *w*-w_i]+v_i

(3)装入 2 个物品 i　　p[*i*, *w*]=p[*i*-1, *w*-2w_i]+2v_i

递归方程为：

p[*i*, *w*]= max{p[*i*-1, *w*], p[*i*-1, *w*-w_i]+v_i , p[*i*-1, *w*-2w_i]+2v_i}

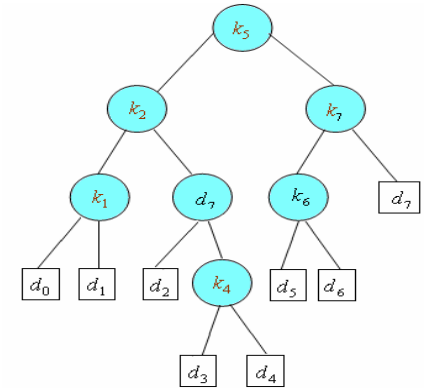
```
012knapsack(W, v, w)
1   for   i←2 to n do
2     for   w← 0 to W do
3       if   2wi<=w then
4           p[i,w] ←max{p[i-1,w],p[i-1,w-wi]+vi, p[i-1, w-2wi]+2vi}
5       else   if   wi<=w   then
6                  p[i, w] ←max{p[i-1, w], p[i-1, w-wi]+vi }
7            else
8                  p[i, w] ←p[i-1, w]
```

6.15

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.1	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

最优代价为 $E(T)$=2.96

最优二叉树：



6.16

当用直接公式计算时，则求 $w[i, j]$ 需要的时间为 $O(n)$，OptimalBST 算法中有两个 **for** 嵌套的循环和嵌套在这两个外循环的两个独立的内循环。所以算法的时间复杂度为：

$$T(n) = O(nn(n+n)) = O(n^3)$$

6.17

```
OptimalBST(p, q, n)
1   for i←1 to n + 1 do
2               e[i, i– 1] ← qi-1
3               w[i, i– 1] ← qi-1
4   for i←1 to n do
5               root[i,i] ←i
6   for l← 1 to n do
7               for i ← 1 to n–l + 1 do
8                       j ←i + l–1
9                       e[i, j ]←∞
10                      w[i, j ] ← w[i, j–1] + pj+qj
11                      for r←root[i,j-1] to root[i+1,j] do
12                              t ← e[i, r–1] + e[r + 1, j ] + w[i, j ]
13                              if t < e[i, j] then
14                                      e[i, j] ← t
15                                      root[i, j ] ←r
16  return e and root
```

6.18

不妨设 $a_1 \le a_2 \le \cdots \le a_n$，假设用 $a_1, a_2, \cdots, a_i$ 硬币找钱数 j 的最少硬币个数为 $c[i, j]$，

设找钱序列为 $x_1, x_2, \cdots, x_i$，其中 x_k 表示面值 a_k 需要的个数。则最优子结构性质为：

假设 $x_1, x_2, \cdots, x_i$ 是最优找钱序列，则 $x_1, x_2, \cdots, x_{i-1}$ 是只用面值 $a_1, a_2, \cdots, a_{i-1}$ 找钱 $j - x_i a_i$ 的一个最优找钱序列。

由最优子结构性质可得如下递归方程

$$c[i, j] = \min_{0 \le k \le (j/a_i)} (k + c[i-1, j - k a_i])$$

初始条件为 $c[i,0] = 0, i = 1,...,n$

$$c[1, j] = \begin{cases} \infty & j \bmod a_1 \neq 0 \\ j / a_1 & otherwise \end{cases}$$

```
FindMoney()
1   for i←1 to n do
2       c[i,0]←0
3   for i←1 to j do
4       if i≥a[1] then
5           if i%a[1]=0 then c[1,i]←i/a[1] else c[1,i]←Max
6       else
7           c[1,i]←Max
8   for i←2 to n do
9       for k←1 to j do
10          if k≥a[i] then
11              if c[i,k-a[i]]+1 < c[i-1,k] then c[i,k]←c[i, k-a[i]]+1
12              else c[i,k]←c[i-1,k]
13              if c[i,k]≥Max then c[i,k]←Max
14          else
15              c[i,k]←c[i-1,k]
16  k←j
17  s←n
18  if c[s, k]<Max then
19      while k>0 do
20              if c[s,k]=c[s-1,k] then s←s-1
21              else
22                  x[s]←x[s]+1;
23                  k←k-a[s]
```

6.19

问题是需要求出将字符串 A 转换为字符串 B，所需要的最少的字符运算数。请设计一个有效算法，对任给的 2 个字符串 A 和 B，输出将字符串 A 转换为字符串 B 所需的最少字符运算数。

设两个字符串为 $A_i =< a_1, a_2, \cdots, a_i >$ 和 $B_j =< b_1, b_2, \cdots, b_j >$，令 $c[i, j]$ 表示将 A_i 转换

为 B_i 所需要的最小的字符运算数，则有下面三种情况：

1)删除 A_i 一个字符 a_i，则 $c[i, j]= c[i\text{-}1, j]+1$；

2)插入一个字符 b_j，则 $c[i, j]= c[i, j\text{-}1]+1$；

3)将一个字符改写为另一个字符，如果 $a_i = b_j$，则需要字符运算 $c = 0$，否则需要 $c = 1$，因此 $c[i, j]= c[i\text{-}1, j\text{-}1]+c$

综上所述，递归方程为 $c[i, j]=\min\{ c[i\text{-}1, j]+1, c[i, j\text{-}1]+1, c[i\text{-}1, j\text{-}1]+c \}$

详细的算法过程如下

```
TransformString(A, B)
1    for i←0 to m do
2          c[i,0] = i
3    for j←0 to n do
4          c[0,j] = j
5    for i←1 to m do
6        for j = 1 to n do
7            c[i,j]←Max
8            c[i,j]←min{c[i,j], c[i – 1,j] + 1}
9            c[i,j]←min{c[i,j], c[i,j - 1] + 1}
10           if  a_i = b_j  then c←0
11           else c←1
12           c[i,j]←  min{c[i,j], c[i – 1,j - 1] + c}
```

6.20

设 n 堆货物从左到右编号为 1,2,⋯,n，每堆货物的货物数分别为 A1,A2,⋯,An，n 堆货物的合并可以有许多不同的方式，每一种合并方式都对应于 A1,A2,⋯,An 的一种完全加括号方式。这样类似矩阵链的乘积问题求解，令合并子问题 $A_i, A_{i+1}, \cdots, A_j$ 所需要的最小总代价为 $m[i, j]$，则有如下递归方程：

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min\limits_{i \le k \le j}\{m[i,k] + m[k+1, j] + \sum\limits_{r=i}^{j} A_r\} & \text{if } i < j \end{cases}$$

详细的伪代码略。

6.21

算法思想：从三角形顶至三角形底的最优路径包含了该路径上各点至三角形底的最优路径。令 a[i,j]表示数字三角形某点的数字，设 f[i,j]表示数字 a[i, j]至三角形底的最优路径，则

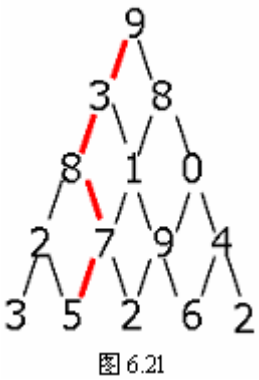


图 6.21

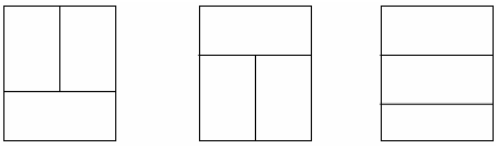
$$f(i,j)=\begin{cases} a[i,j] & i=n \\ \max\{f[i+1,j],f[i+1,j+1]\}+a[i,j] & i<n \end{cases}$$

详细的伪代码略。

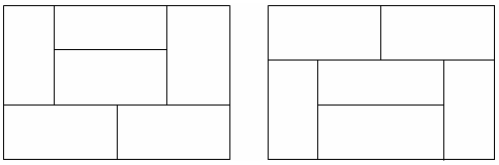
6.22

由题可求出切割布条的方法有两种：

一种是占用的大小，这种切割法有三种方案（该算法中有考虑上下次序），如下图所示：



另外一种是占用的大小，这种切割法有两种方案，如下图所示：



对整条布的切割其实可看为是先用的方法切割或先用的方法切割，余下的布条用同样的方法切割。令 $F(n)$表示将 $3\times n$ 的布条按照要求切割的种数，则递归方程为：

$$F(n)=\begin{cases} 1 & n=1 \\ 3 & n=2 \\ 3F(n-2)+2F(n-4) & n>2 \end{cases}$$

详细的伪代码略。

6.23

对整条路的走法与对部分路的走法是一样的，设 $f(n)$表示为走 n 米的走法总数，则递归方程为：

$$f(n)=\begin{cases} 1 & n=1 \\ 2 & n=2 \\ 4 & n=3 \\ f(n-1)+f(n-2)+f(n-3) & n>3 \end{cases}$$

6.24

算法思想：先将这 n 个人分成两组(当 n 为偶数时，两组人数相等，当 n 为奇数时，两组人数相差为 1)，接下来按以下的递归过程调换两边的人直到两边的重量差为最小：

递归过程：1 先计算出两边的重量差 m；

2 查询两边中重量差最接近 m/2 的两个人，并把这两个人对调；

3 当 m=0 或原 m 的绝对值小于新 m 的绝对值，则说明两组的重量差达到最小。可推出递归。

实验题

6.25 分别实现矩阵链乘法的递归算法、动态规划算法及其备忘录算法，并用实验分析方法分析比较三种算法的效率。

6.26 完成 XOJ 如下题目：1010，1022，1023，1028，1029，1030，1031，1032，1033，1038，1041，1056，1059，1086。

完成 POJ 如下题目：1018，1050，1080，1088，1159，1160，1179，1276，1678，1742，2593，2614，2411，2778，320