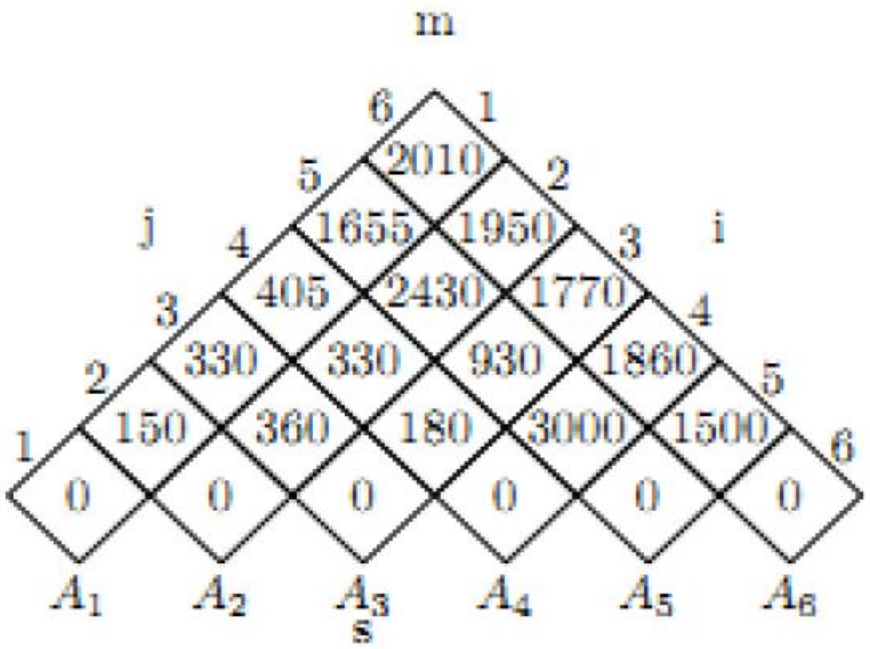
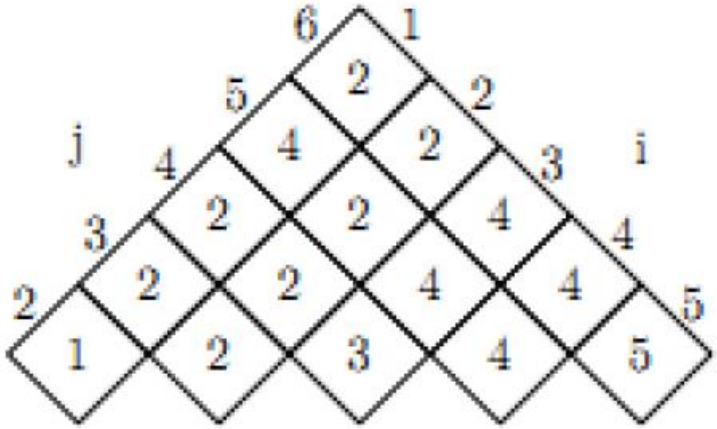
第15~16章作业

15.2-1：对于矩阵规模序列⟨5,10,3,12,5,50,6⟩，求其矩阵链最优括号化方案。





最终结果为$(A_1A_2)((A_3A_4)(A_5A_6))$

令 m[i,j] 为计算矩阵链$A_{i,j}$所需的标量乘法运算次数的最小值。则

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\ , \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j\ . \end{cases}$$

15.4-1求⟨1,0,0,1,0,1,0,1⟩和⟨0,1,0,1,1,0,1,1,0⟩的一个LCS

i		j	0	1	2	3	4	5	6	7	8	9
			y_j	0	1	0	1	1	0	1	1	0
0	x_i		0	0	0	0	0	0	0	0	0	0
1	1		0	↑ 0	↖ 1	← 1	↖ 1	↖ 1	← 1	↖ 1	↖ 1	← 1
2	0		0	↖ 1	↑ 1	↖ 2	← 2	← 2	↖ 2	← 2	← 2	↖ 2
3	0		0	↖ 1	↑ 1	↖ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4	1		0	↑ 1	↖ 2	↑ 2	↖ 3	↖ 3	↑ 3	↖ 4	↖ 4	← 4
5	0		0	↖ 1	↑ 2	↖ 3	↑ 3	↑ 3	↖ 4	↑ 4	↑ 4	↖ 5
6	1		0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 4	↑ 4	↖ 5	↖ 5	↑ 5
7	0		0	↖ 1	↑ 2	↖ 3	↑ 4	↑ 4	↖ 5	↑ 5	↑ 5	↖ 6
8	1		0	↑ 1	↖ 2	↑ 3	↖ 4	↖ 5	↑ 5	↖ 6	↖ 6	↑ 6

LCS为<1,0,0,1,1,0>

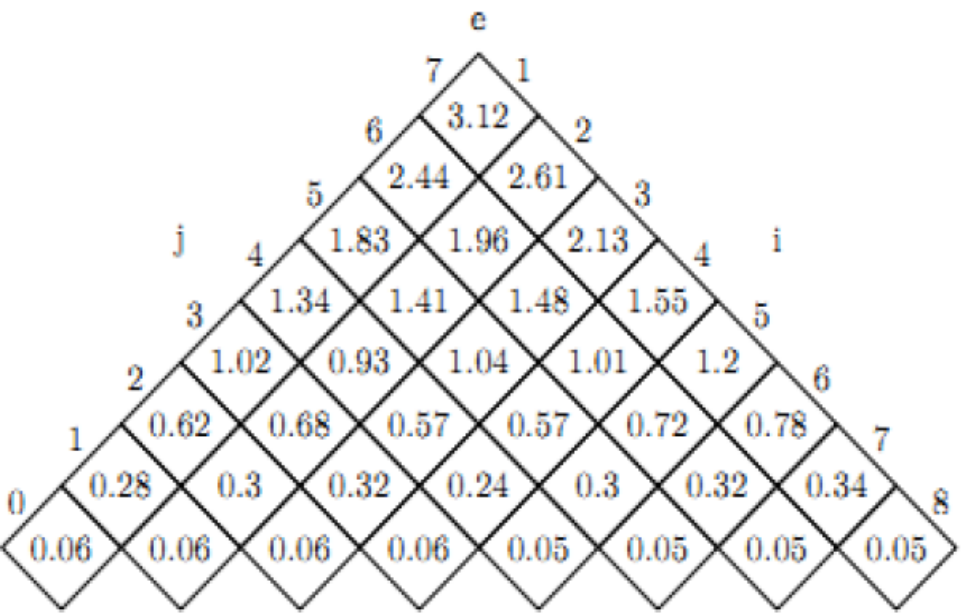
其他的LCS（如101010、101011等，只要公共子序列长度为6均为正确的结果）

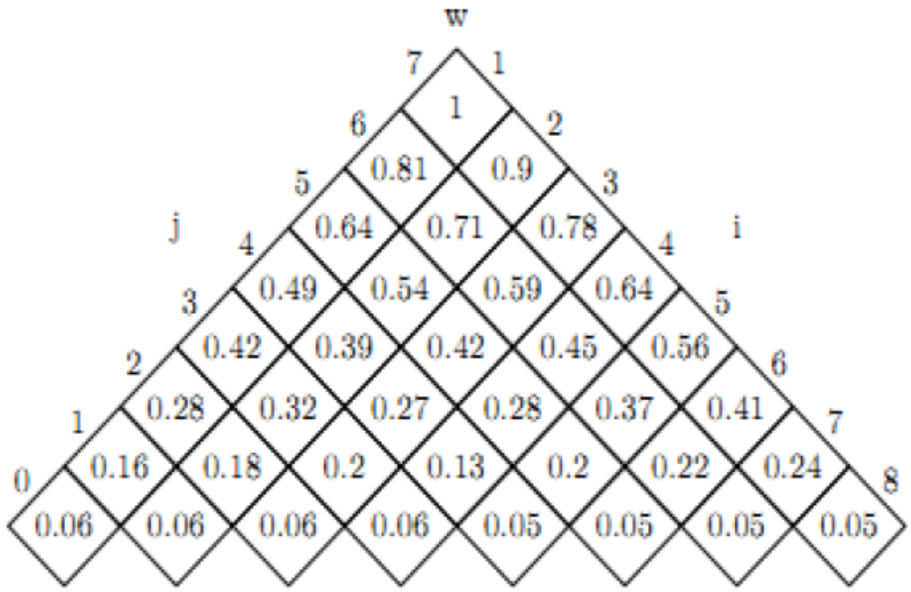
c[i,j]为前缀序列X_i和Y_j的一个LCS的长度。则有

$$c[i,j]=\begin{cases}0 & \text{如果}i=0\text{或}j=0\\ c[i-1,j-1]+1 & \text{如果}i,j>0\text{且}x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果}i,j>0\text{且}x_i\neq y_j\end{cases}$$

15.5-2：若7个关键字的概率如下所示，求其最优二叉搜索树的结构和代价。

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

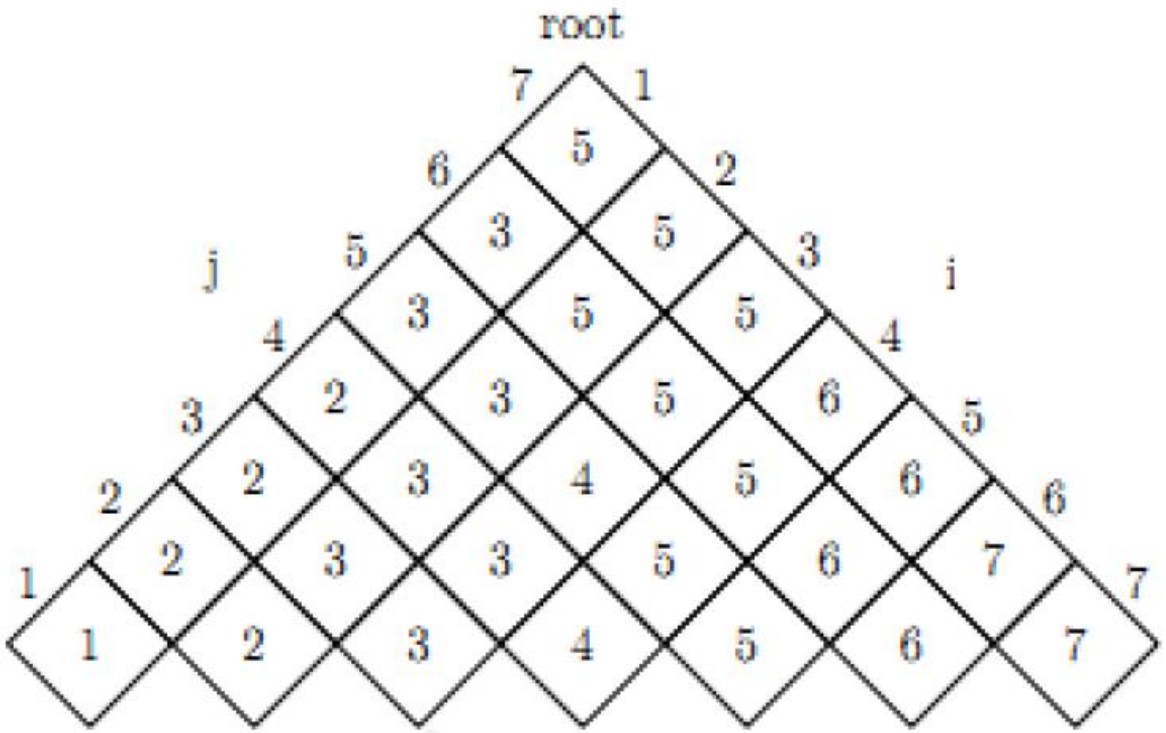


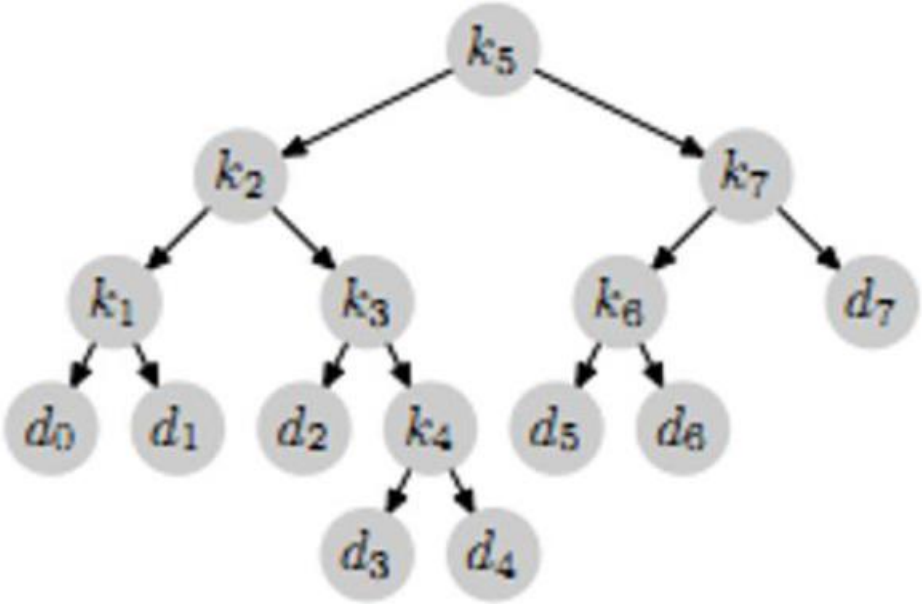


e[i,j]为包含关键字$k_i,\ldots,k_j$的最优二叉搜索树的期望搜索代价

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1, \\ \min\limits_{i \le r \le j} \{e[i,r-1] + e[r+1,j] + w(i,j)\} & \text{if } i \le j. \end{cases}$$

$$w(i,j) = w(i,r-1) + p_r + w(r+1,j)$$


root
j
i
7 1
6 2
5 3
4 4
3 5
2 6
1 7
5
3 5
3 5 5
2 3 5 6
2 3 4 5 6
2 3 3 5 6 7
1 2 3 4 5 6 7


k_5
k_2 k_7
k_1 k_3 k_6 d_7
d_0 d_1 d_2 k_4 d_5 d_6
d_3 d_4

15.1-3：我们修改一下钢条切割问题，除了切下的钢条段具有不同价格pi 外，每次切割还要付出固定成本c。这样，切割方案的收益就等于钢条段价格之和减去切割的成本。设计一个动态规划算法解决修改后的钢条切割问题。

```
MODIFIED-CUT-ROD(p, n, c)
  let r[0..n] be a new array
  r[0] = 0
  for j = 1 to n
      q = p[j]
      for i = 1 to j - 1
          q = max(q, p[i] + r[j - i] - c)
      r[j] = q
  return r[n]
```

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

15-9 （字符串拆分） 某种字符串处理语言允许程序员将一个字符串拆分为两段。由于此操作需要复制字符串，因此要花费 n 个时间单位来将一个 n 个字符的字符串拆为两段。假定一个程序员希望将一个字符串拆分为多段，拆分的顺序会影响所花费的总时间。例如，假定这个程序员希望将一个 20 个字符的字符串在第 2 个、第 8 个以及第 10 个字符后进行拆分（字符由左至右，从 1 开始升序编号）。如果她按由左至右的顺序进行拆分，则第一次拆分花费 20 个时间单位，第二次拆分花费 18 个时间单位（在第 8 个字符处拆分 3～20 间的字符串），而第三次拆分花费 12 个时间单位，共花费 50 个时间单位。但如果她按由右至左的顺序进行拆分，第一次拆分花费 20 个时间单位，第二次拆分花费 10 个时间单位，而第三次拆分花费 8 个时间单位，共花费 38 个时间单位。还可以按其他顺序，比如，她可以首先在第 8 个字符处进行拆分（时间 20），接着在左边一段第 2 个字符处进行拆分（时间 8），最后在右边一段第 10 个字符处进行拆分（时间 12），总时间为 40。

设计算法，对给定的拆分位置，确定最小代价的拆分顺序。更形式化地，给定一个 n 个字符的字符串 S 和一个保存 m 个拆分点的数组 $L[1..m]$，计算拆分的最小代价，以及最优拆分序列。

1）先确定满足最优子结构性质的子问题

2）再对输入做一些处理

（1）对拆分点数组L做个排序，拆分点按升序（从左向右）排列

（2）在L起点追加下标0，在L末尾追加下标n

例如：L=<20,17,14,11,25>和n=30，排序，追加，得到

L=<0,11,14,17,20,25,30>

定义 L[i...j]为L的下标从i到j的子数组，

定义 子问题(i,j)为 “拆分子序列S[L[i]+1...L[j]]的最小花费序列”

而该子序列里的拆分点是子数组L[i+1...j-1]。

如：L=<0,11,14,17,20,25,30>

子问题<2,6>为使得L[2]=11,L[6]=25的拆分S后获得的子序列，

考虑子串S[12..25]的最小拆分花费，内部只需用考虑L[3..5]。

定义 cost[i,j]表示子问题(i,j)的最小拆分花费，则有：

$$cost[i,j] = \begin{cases} 0 & \text{if } j - i \le 1, \\ \min\limits_{i<k<j} \{cost[i,k] + cost[k,j] + (L[j] - L[i])\} & \text{if } j - i > 1. \end{cases}$$

```
BREAK-STRING(n, L)
  prepend 0 to the start of L and append n to the end of L
  m = L.length
  sort L into increasing order
  let cost[1..m, 1..m] and break[1..m, 1..m] be new tables
  for i = 1 to m − 1
      cost[i, i] = cost[i, i + 1] = 0
  cost[m, m] = 0
  for len = 3 to m
      for i = 1 to m − len + 1
          j = i + len − 1
          cost[i, j] = ∞
          for k = i + 1 to j − 1
              if cost[i, k] + cost[k, j] < cost[i, j]
                  cost[i, j] = cost[i, k] + cost[k, j]
                  break[i, j] = k
          cost[i, j] = cost[i, j] + L[j] − L[i]
  print "The minimum cost of breaking the string is " cost[1, m]
  PRINT-BREAKS(L, break, 1, m)
```

```
PRINT-BREAKS(L, break, i, j)
  if j − i ≥ 2
      k = break[i, j]
      print "Break at " L[k]
      PRINT-BREAKS(L, break, i, k)
      PRINT-BREAKS(L, break, k, j)
```

时间复杂度：

- 三重循环
- $O(m^3)$

$$
\begin{aligned}
\sum_{i=1}^{m-2}\sum_{j=i+2}^{m}(j-i-1) &= \sum_{i=1}^{m-2}\sum_{d=1}^{m-i-1} d && (d = j-i-1)\\
&= \sum_{i=1}^{m-2}\Theta((m-i)^2) && \text{(equation (A.2))}\\
&= \sum_{h=2}^{m-1}\Theta(h^2) && (h = m-i)\\
&= \Theta(m^3) && \text{(equation (A.3))}\,.
\end{aligned}
$$

15-11 （*库存规划*） Rinky Dink 公司是一家制造溜冰场冰面修整设备的公司。这种设备每个月的需求量都在变化，因此公司希望设计一种策略来规划生产，需求是给定的，即它虽然是波动的，但是可预测的。公司希望设计接下来 n 个月的生产计划。对第 i 个月，公司知道需求 d_i，即该月能够销售出去的设备的数量。令 $D=\sum_{i=1}^{n} d_i$ 为后 n 个月的总需求。公司雇用的全职员工，可以提供一个月制造 m 台设备的劳动力。如果公司希望一个月内制造多于 m 台设备，可以雇用额外的兼职劳动力，雇用成本为每制造一台机器付出 c 美元。而且，如果在月末有设备尚未售出，公司还要付出库存成本。保存 j 台设备的成本可描述为一个函数 $h(j)$，$j=1, 2, \cdots, D$，其中对所有 $1 \leqslant j \leqslant D$，$h(j) \geqslant 0$，对 $1 \leqslant j \leqslant D-1$，$h(j) \leqslant h(j+1)$。

设计库存规划算法，在满足所有需求的前提下最小化成本。算法运行时间应为 n 和 D 的多项式函数。

1）最优子结构性

令(k,s)表示到第k个月之前（不含第k个月）时库存s台机器，并且自第k个月往后生产至第n个月的成本最低子问题。

设P是(k,s)的最优计划。令f是P计划第k个月生产的机器数。则到第k+1月时，库存为$s'=s+f-d_k$，d_k为第k个月的需求，则相对于(k+1, s')，P的子计划P'必须是最优的。否则可用更有的子计划P''代替P'，与P是最优的相矛盾。

令cost[k,s]表示子问题(k,s)最优计划的成本，令f是第k个月生产的机器数。

则f的下限是：$L(k,s)=\max(d_k-s,0)$

f的上限是：$U(k,s)=\sum_{t=k}^{n} d_t - s$

1) 最后一个月：

只要生产可交货的数量即可，不用多生产，所以有：

$$cost[n,s]=c\cdot\max(L(n,s)-m,0)+h(s+L(n,s)-d_n)$$

h是库存成本

2）其它各月：

$$\begin{aligned} cost[k,s] = \min_{L(k,s)\le f\le U(k,s)} \{ & cost[k+1, s+f-d_k] \\ & + c \bullet \max(f-m, 0) \\ & + h(s+f-d_k) \} \end{aligned}$$

令 $D = \sum_{i=1}^{n} d_i$

$$\cos t[k, s] = \min_{L(k,s) \le f \le U(k,s)} \{\cos t[k+1, s+f-d_k] + c \bullet \max(f-m, 0) + h(s+f-d_k)\}$$

INVENTORY-PLANNING(n, m, c, D, d, h)

let $cost[1..n, 0..D]$ and $make[1..n, 0..D]$ be new tables
// Compute $cost[n, 0..D]$ and $make[n, 0..D]$.
for $s = 0$ **to** D
    $f = \max(d_n - s, 0)$
    $cost[n, s] = c \cdot \max(f - m, 0) + h(s + f - d_n)$
    $make[n, s] = f$
// Compute $cost[1..n-1, 0..D]$ and $make[1..n-1, 0..D]$.
$U = d_n$
for $k = n - 1$ **downto** 1
    $U = U + d_k$
    for $s = 0$ **to** D
        $cost[k, s] = \infty$
        for $f = \max(d_k - s, 0)$ **to** $U - s$
            $val = cost[k+1, s+f-d_k] + c \cdot \max(f - m, 0) + h(s + f - d_k)$
            if $val < cost[k, s]$
                $cost[k, s] = val$
                $make[k, s] = f$
print $cost[1,0]$
PRINT-PLAN$(make, n, d)$

```
PRINT-PLAN(make, n, d)
  s = 0
  for k = 1 to n
      print "For month " k " manufacture " make[k, s] " machines"
      s = s + make[k, s] − d_k
```

- 时间复杂度：$O(nD^2)$

15.2-5：令R(i,j) 表示在一次调用MATRIX-CHAIN-ORDER过程中，计算其他表项时访问表项m[i,j] 的次数。证明：

$$\sum_{i=1}^{n}\sum_{j=1}^{n} R(i,j) = \frac{n^3 - n}{3}$$

证：

- l从2到n，循环n-1次
- 每次l循环，i循环执行n-l+1次；
- 每次i循环，k循环执行j-i=l-1次；
- 每次k循环，对m引用两次；

```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

- 所以整个计算过程对m中表项的引用次数为

$$\sum_{l=2}^{n}(n-l+1)(l-1)2$$

- 亦即

$$
\begin{aligned}
\sum_{i=1}^{n}\sum_{j=i}^{n} R(i,j) &= \sum_{l=2}^{n}(n-l+1)(l-1)2 \\
&= 2\sum_{l=1}^{n-1}(n-l)l \\
&= 2\sum_{l=1}^{n-1} nl - 2\sum_{l=1}^{n-1} l^2 \\
&= 2\frac{n(n-1)n}{2} - 2\frac{(n-1)n(2n-1)}{6} \\
&= n^3 - n^2 - \frac{2n^3 - 3n^2 + n}{3} \\
&= \frac{n^3 - n}{3}\,.
\end{aligned}
$$

15.3-6

Let $c_1 = 2$ and $c_2 = c_3 = 3$. Note that this example is not too badly contrived, in that $r_{ji} = 1/r_{ij}$ for all i and j.

To see how this example does not exhibit optimal substructure, let's examine an optimal solution for exchanging currency 1 for currency 4. There are five possible exchange sequences, with the following costs:

$$
\begin{array}{lcll}
\langle 1,4\rangle & : & 6-2 & = 4\,, \\
\langle 1,2,4\rangle & : & 2\cdot 3-3 & = 3\,, \\
\langle 1,3,4\rangle & : & 5/2\cdot 3-3 & = 9/2\,, \\
\mathbf{\langle 1,2,3,4\rangle} & : & \mathbf{2\cdot 3/2\cdot 3-3} & = \mathbf{6} \\
\langle 1,3,2,4\rangle & : & 5/2\cdot 2/3\cdot 3-3 & = 2
\end{array}
$$

The optimal exchange sequence, $\langle 1,2,3,4\rangle$, appears in boldface.

Let's examine the subproblem of exchanging currency 1 for currency 3. Allowing currency 4 to be part of the exchange sequence, there are again five possible exchange sequences with the following costs and the optimal one in boldface:

$$
\begin{array}{lcll}
\mathbf{\langle 1,3\rangle} & : & \mathbf{5/2-2} & = \mathbf{1/2} \\
\langle 1,2,3\rangle & : & 2\cdot 3/2-3 & = 0 \\
\langle 1,4,3\rangle & : & 6\cdot 1/3-3 & = -1 \\
\langle 1,2,4,3\rangle & : & 2\cdot 3\cdot 1/3-3 & = -1 \\
\langle 1,4,2,3\rangle & : & 6\cdot 1/3\cdot 3/2=3 & = 0
\end{array}
$$

We see that the solution to the original problem includes the subproblem of exchanging currency 1 for currency 3, yet the solution $\langle 1, 2, 3\rangle$ to the subproblem used in the optimal solution to the original problem is not the optimal solution $\langle 1,3\rangle$ to the subproblem on its own.