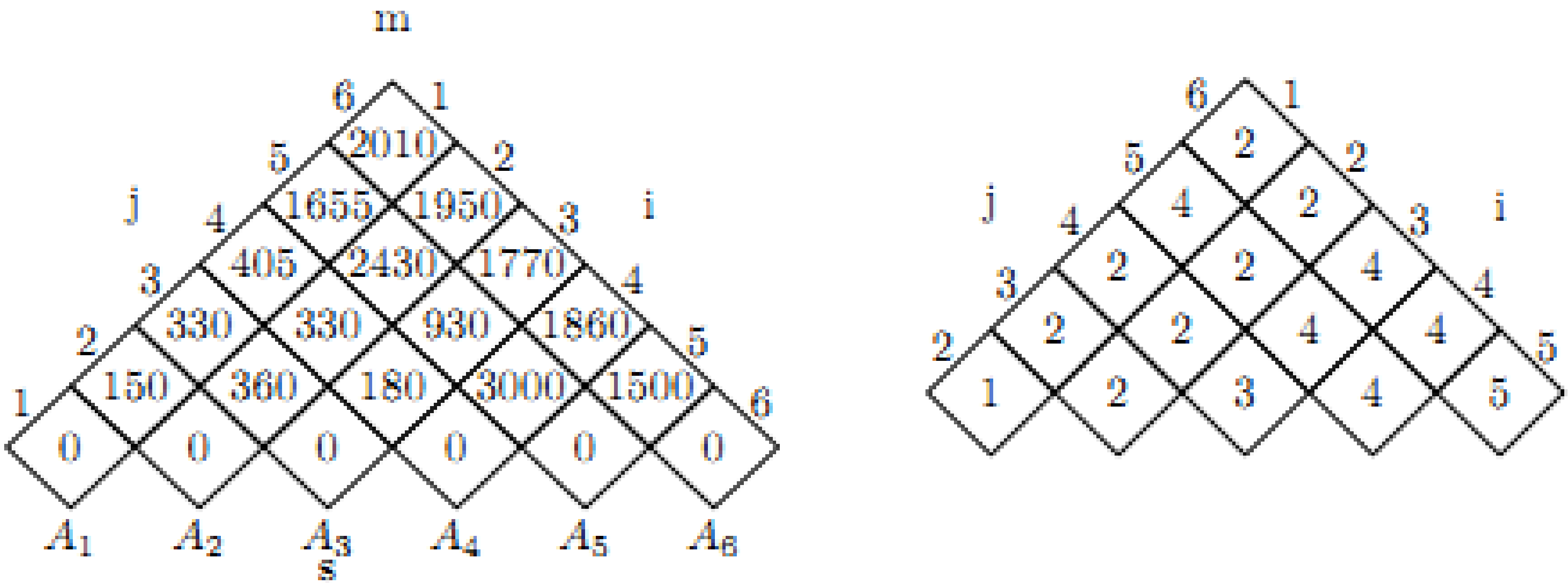
第15~16章作业

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。

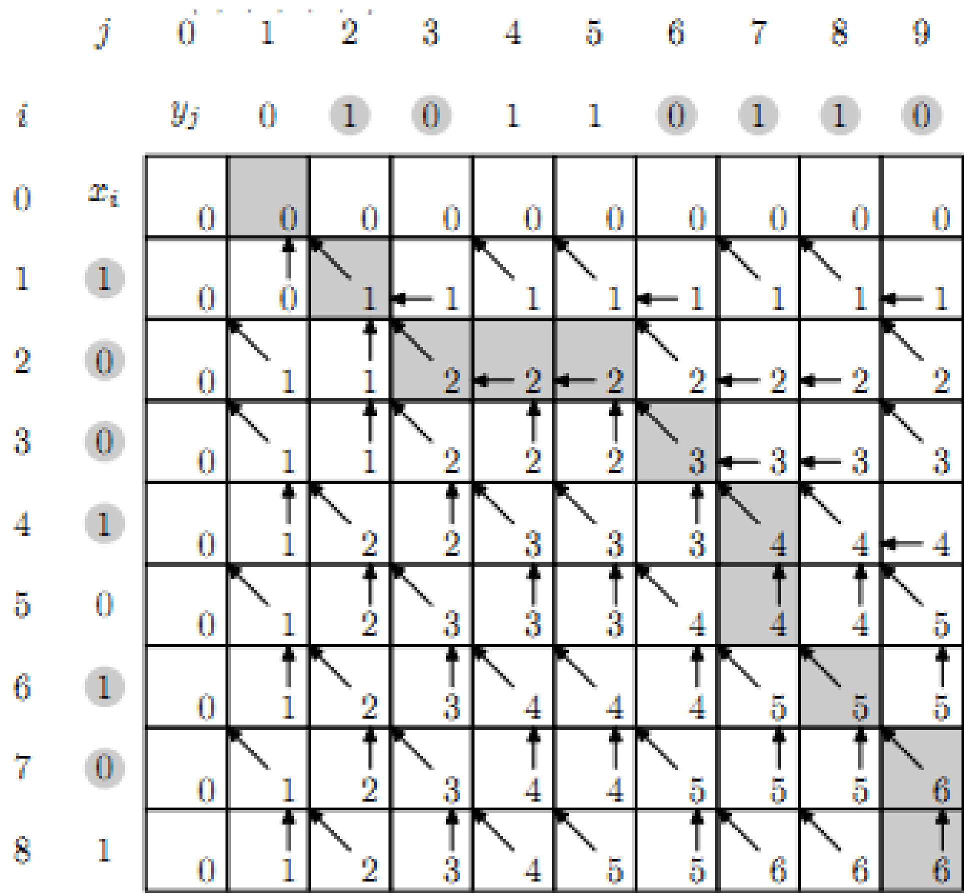


最终结果为$(A_1A_2)((A_3A_4)(A_5A_6))$

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

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

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



LCS为<1,0,0,1,1,0>

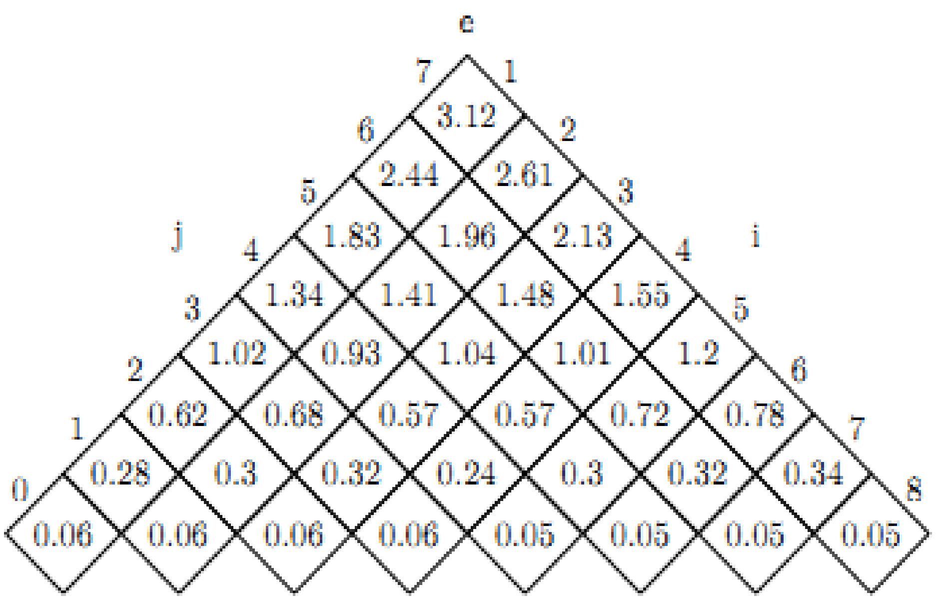
其他的LCS（如101010、101011等，只要公共子序列长度为6均为正确的结果）

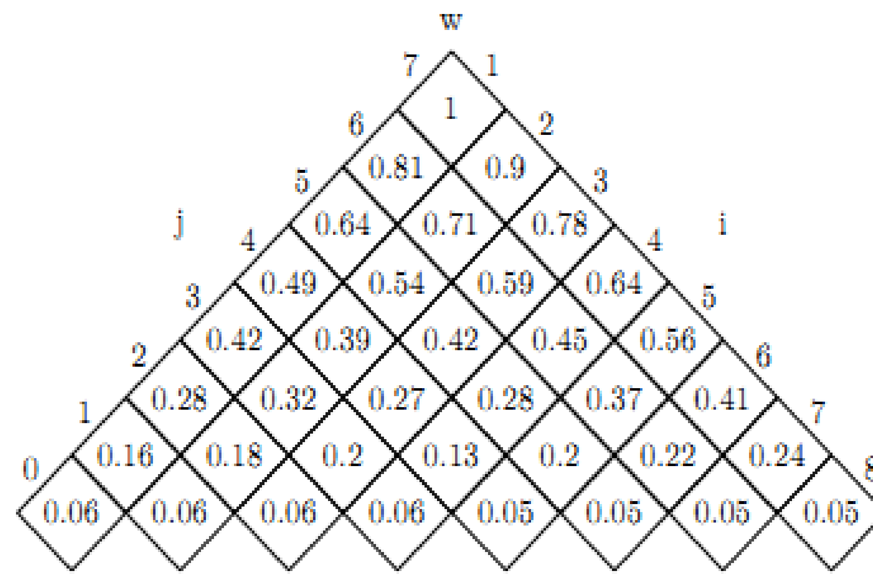
c[i,j]为前缀序列X_i和Y_j的一个LCS的长度。则有

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

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

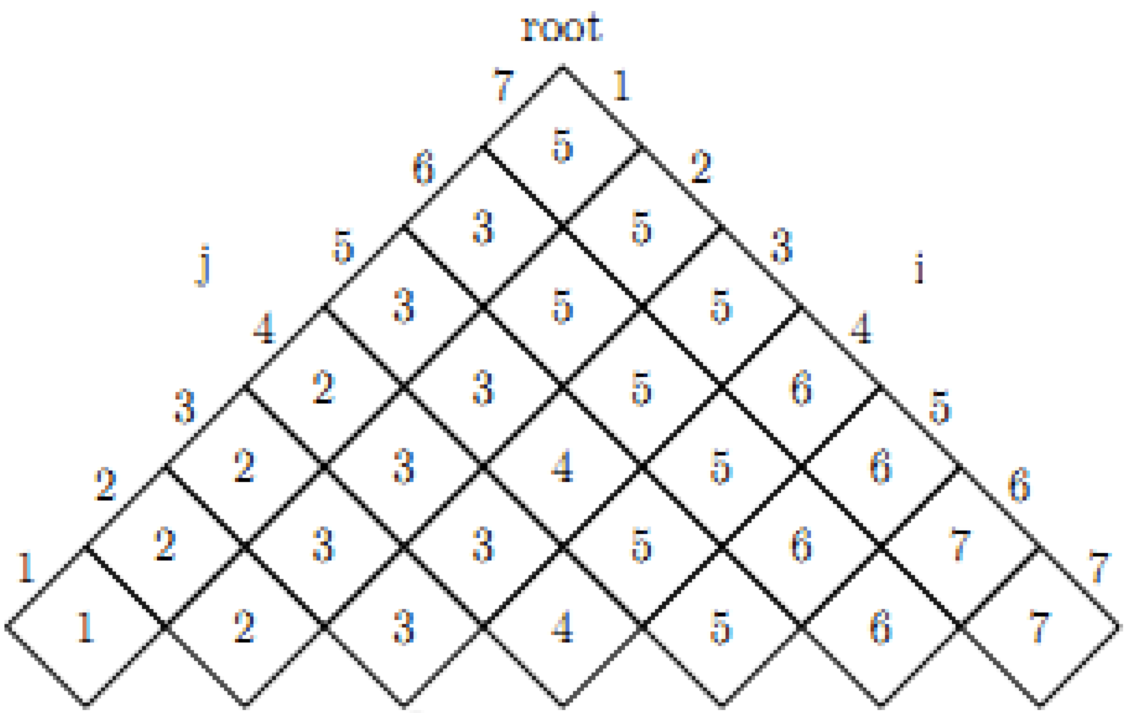


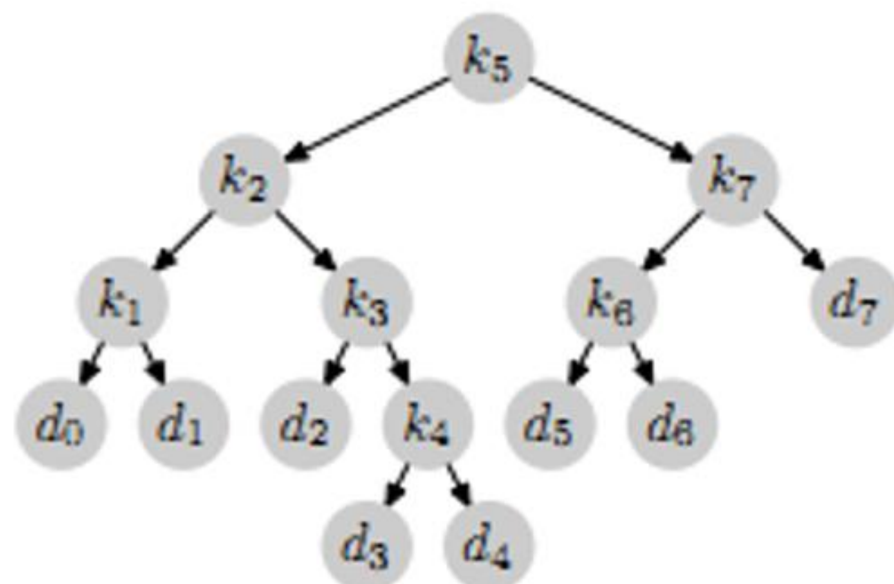


e[i,j]为包含关键字$k_i,\ldots,k_j$的最优二叉搜索树的期望搜索代价

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

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


root
j
i


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)
  let r[0 . . n] be a new array
  r[0] = 0
  for j = 1 to n
      q = -∞
      for i = 1 to j
          q = max(q, p[i] + r[j - i])
      r[j] = q
  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：

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

```
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(m³)

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

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：

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

令 $D = \sum_{i=1}^{n} d_i$

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

```
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 · 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] = ∞
          for f = max(d_k - s, 0) to U - s
              val = cost[k + 1, s + f - d_k]
                        + c · 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：

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i,j] = ∞
        for k = i to j − 1
            q = m[i,k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k
return m and s
```

- 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；
- 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

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

16.1-4 假定有一组活动，我们需要将它们安排到一些教室，任意活动都可以在任意教室进行。我们希望使用最少的教室完成所有活动。设计一个高效的贪心算法求每个活动应该在哪个教室进行。

（这个问题称为**区间图着色问题**(interval-graph color problem)。我们可以构造一个区间图，顶点表示给定的活动，边连接不兼容的活动。要求用最少的颜色对顶点进行着色，使得所有相邻顶点颜色均不相同——这与使用最少的教室完成所有活动的问题是对应的。）

1）将所有活动按照开始时间的非降次序排序（开始时间相同的按结束时间非降顺序排列）

2）为第一个活动分配一间教室，记录其结束时间

3）对剩下的活动依次遍历

对当前活动i，依次检查已经安排过活动的教室，若某间教室最后一个活动的结束时间早于活动i，则将i安排在该教室进行，并修改该教室的活动结束时间。

否则，若所有教室最后一个活动与i冲突，则为i安排一间新的教室，并记录该教室的活动结束时间。

4）时间复杂度

朴素的算法：$O(n^2)$

改进的算法：O(n+排序时间)

- 维护两个教室列表：

  第一个列表P包含当前活动时间t正在被使用的教室（即存在活动i，使得$s_i \le t < f_i$，活动起止时间是半开的）；另一个列表Q含时间t时空闲的教室。

  当t是一个活动的开始时间，活动进入一个空闲的教室，此教室会从空闲列表移到忙碌列表；当t是一个活动的结束时间，将活动的教室从忙碌列表移到空闲列表。

- 将n个活动的开始时间和结束时间（总共2n个“时间”，记录每个时间对应哪个活动（活动指针），是开始时间还是结束时间，活动所占用的教室，开始时候所有活动都不占用教室）一起排序，这会耗费O(nlgn)的时间复杂度。
- 对排好序的2n个时间依次进行扫描，若当前时间是一个活动的开始时间，则从Q表里摘除一个教室r、并把该教室加到P表中，该活动记录在r中进行。若当前时间是一个活动的结束时间，则将其所在教室从P表中摘除并加入Q表中（链表，双向）。
- 为了避免使用更多的教室，总是尽量取已经被某活动用过的教室（即在列表中总把最近从P表中摘除的教室置于最前面，这是最早有空闲时间的教室，可以尽可能多地安排活动，而尽量少的使用教室）。

16.2-6　设计算法，在 $O(n)$ 时间内求解分数背包问题。

- 分数背包问题：

已知n种物品，各具有重量 $(w_1, w_2, \cdots, w_n)$ 和效益值 $(p_1, p_2, \cdots, p_n)$，及一个可容纳M重量的背包。

问：怎样装包才能使在不超过背包容量的前提下，装入背包的物品的总效益最大？

线性时间求解算法：

1. 首先用线性时间求中位数的算法计算v_i/w_i的中位数m；

2. 然后按中位数将数组划分成三部分

$$\mathrm{G}=\left\{\mathrm{i}: \frac{v_i}{w_i}>m\right\}, \mathrm{E}=\left\{\mathrm{i}: \frac{v_i}{w_i}=m\right\}, \mathrm{L}=\left\{\mathrm{i}: \frac{v_i}{w_i}<m\right\},$$

这个步骤也会花线性时间。

3. 再然后，比较G和E的元素权重和

$$W_G=\sum_{i \in G} w_i 、 W_E=\sum_{i \in E} w_i ,$$

（1）当W_G>M（背包容量）,就无需取G中任何元素，而是在G中递归地执行1步骤。

（2）否则，当$W_G \leq M$，取G中所有元素，并且尽可能多的取E中元素来满足剩余重量$M-W_G$的缺口。

（3）当$W_G+W_E \geq M$，此情况包含在（2）中，故无需再操作。

（4）当$W_G+W_E<M$，取G和E中所有元素，并且尽可能多的取L中元素来满足剩余重量$M-W_G$。

- **时间复杂度**：

　　此算法中，求中位数花线性时间，问题为n规模时，子问题为n/2的规模，故有

$$\mathrm{T}(\mathrm{n})=\mathrm{T}\left(\frac{\mathrm{n}}{2}\right)+\Theta(\mathrm{n}),$$

故时间复杂度$\mathrm{T}(\mathrm{n})=O(n)$。

16.2-7 给定两个集合 A 和 B，各包含 n 个正整数。你可以按需要任意重排每个集合。重排后，令 a_i 为集合 A 的第 i 个元素，b_i 为集合 B 的第 i 个元素。于是你得到回报 $\prod_{i=1}^{n} a_i^{b_i}$。设计算法最大化你的回报。证明你的算法是正确的，并分析运行时间。

设计：

对A和B分别逆序排序时，回报$\prod_{i=1}^{n} {a_i}^{b_i}$最大。

证明：

要回报最大，对每个i < j，必须有${a_i}^{b_i}{a_j}^{b_j} \geq {a_i}^{b_j}{a_j}^{b_i}$。

而当A和B分别逆序排序时，$a_i \geq a_j, b_i \geq b_j$，此时，$a_i$、$a_j$均为正数，而$b_i - b_j$为非负数，则有${a_i}^{b_i - b_j} \geq {a_j}^{b_i - b_j}$，即满足

$${a_i}^{b_i}{a_j}^{b_j} \geq {a_i}^{b_j}{a_j}^{b_i}$$

所以可使得$\prod_{i=1}^{n} {a_i}^{b_i}$最大。

- **运行时间：**

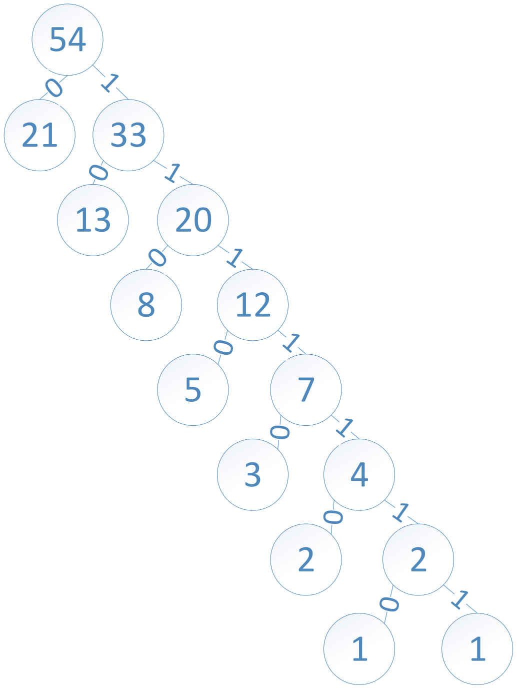
　　先对A、B进行排序，故时间复杂度可以为O(nlogn)，然后直接计算$\prod_{i=1}^{n} {a_i}^{b_i}$即可，计算步骤时间复杂度为O(n)，故时间复杂度为O(nlogn)依赖于排序的时间复杂度。

16.3-3 如下所示，8 个字符对应的出现频率是斐波那契数列的前 8 个数，此频率集合的赫夫曼编码是怎样的？

a：1　b：1　c：2　d：3　e：5　f：8　g：13　h：21

你能否推广你的结论，求频率集为前 n 个斐波那契数的最优前缀码？

解，根据哈夫曼编码的步骤，可得如下**树**：



最优前缀码：

字符	a	b	c	d	e	f	g	h
数值	1	1	2	3	5	8	13	21
最优前缀码	1111111	1111110	111110	11110	1110	110	10	0

可以推广此结论：

求n个斐波那契数$(a_1, a_2, \ldots, a_n)$的最优前缀码。

数学归纳法：当n>=3时，若已知斐波那契数列的前n-1个数，其前缀编码为：111...1(n-1个1),111...0(n-2个1,1个0),111..0(n-3个1,1个0),...,10,0

这是因为：由于$a_n = a_{n-1} + a_{n-2}$,显然$a_n > a_{n-1}$且$a_n > \sum_{i=1}^{n-2} a_i$(可用数学归纳法证得)，故在第n-2次时会选$a_{n-1}$、$\sum_{i=1}^{n-2} a_i$作为哈夫曼树中子节点，合并出父节点后再与$a_n$进行结合（根据哈夫曼树生成的步骤）。

16-1 （找零问题） 考虑用最少的硬币找 n 美分零钱的问题。假定每种硬币的面额都是整数。

a. 设计贪心算法求解找零问题，假定有 25 美分、10 美分、5 美分和 1 美分 4 种面额的硬币。证明你的算法能找到最优解。

b. 假定硬币面额是 c 的幂，即面额为 c^0，c^1，…，c^k，c 和 k 为整数，$c>1$，$k\geqslant 1$。证明：贪心算法总能得到最优解。

c. 设计一组硬币面额，使得贪心算法不能保证得到最优解。这组硬币面额中应该包含 1 美分，使得对每个零钱值都存在找零方案。

d. 设计一个 $O(nk)$时间的找零算法，适用于任何 k 种不同面额的硬币，假定总是包含 1 美分硬币。

解：

a. **贪心算法：**

找$\mathrm{q} = \lfloor n/25 \rfloor$个25美分硬币，剩余$n_q = n \ mod\ 25$美分待找。

找$\mathrm{d} = \lfloor n_q/25 \rfloor$个10美分硬币，剩余$n_d = n_q \ mod\ 10$美分待找

找$\mathrm{k} = \lfloor n_d/25 \rfloor$个5美分硬币，剩余$n_k = n_d mod\ 5$美分待找

找$\mathrm{p} = n_k$个1美分硬币

证明最优解性质：

问题描述：找n美分的硬币，若n=0，最优解就是不找硬币；n>0，找接近n的最大面额。假设这最大面额为c，则问题转化为找n-c美分的硬币的子问题。

最优性证明：首先，说明贪婪选择性质，找n美分，选最接近c的最大面额c的硬币(c≤n)，如果最优解包含c，则此步骤是最优解的步骤。否则，这个最优解不含c，故有以下四种情况考虑：

1） 1 ≤ n<5,则c=1，解只含1美分，故最优解肯定包含贪婪选择特性。

2） 5 ≤ n<=10,则c=5，根据假设，最优解不含5美分的硬币，故只包含1美分硬币，而选n个1美分的硬币明显比选一个5美分多个1美分的硬币数更多，这与最优解矛盾，故最优解肯定包含贪婪选择特性。

3） 10 ≤ n<=25,则c=10，根据假设，最优解不含10美分硬币，故最优解只含5美分和1美分硬币，一些5美分和多个1美分加起来等价于1个10美分的硬币，用10美分硬币替代这部分，则最优解有更多的金币数，矛盾，故最优解肯定包含贪婪选择特性。

4） 25 ≤ n，则c=25，根据假设，最优解不含25美分硬币，只含1,5,10美分硬币，多个1美分，5美分和10美分可以凑成1个25美分，故可以用一枚25美分替代这部分，此时最优解有更多金币数，矛盾，故最优解肯定包含贪婪选择特性。

故最优解包含贪婪选择，将贪婪选择用于对每个子问题求最优解，最终可以得到问题的最优解。

b. 贪心算法:

硬币面额包含$c^0, c^1, \dots, c^k$，贪婪算法找n美分的金币，可以通过找面额c^j，$\mathrm{j} = \max\{0 \le \mathrm{i} \le \mathrm{k}: c^i \le n\}$，给定一个金币面额$c^j$，子问题找n-$c^j$即可递归的进行找零钱。

最优性证明：

引理：对于i=0,1,...,k，假设a_i为找n美分时使用的c^i面额硬币的硬币数目，故有$a_i < c$

引理证明：若对于部分$0 \le \mathrm{i} < \mathrm{k}, a_i \ge c$，故可以用一个$c^{i+1}$和$a_i - c$个$c^i$来达到$a_i c^i$，此时用的金币数更少，故最优解需满足$a_i < c$

下证贪婪选择是最优的：

让$j = \max\{0 \le i \le k: c^i \le n\}$，故贪婪选择用至少一个$c^j$面额的硬币。而对于一个非贪婪解，则必然不含$c^j$面额的金币。

非贪婪时，假设a_i(i=0,1,...,j-1)为找n美分时使用的c^i面额硬币的硬币数目。则有$\sum_{i=0}^{j-1} a_i c^i = n$。

因$n \ge c^j$，故有$\sum_{i=0}^{j-1} a_i c^i \ge c^j$。

根据引理，$a_i < c$，即 $a_i \le c - 1$，故有

$$n = \sum_{i=0}^{j-1} a_i c^i \le \sum_{i=0}^{j-1} (c-1) c^i = (c-1) \sum_{i=0}^{j-1} c^i = c^j - 1 < c^j$$

这与 $n \ge c^j$ 矛盾，故非贪婪解不是最优的。只有贪婪解是最优的。

c. 假设只含面额1,10和25美分的金币，当需要找三十美分，贪婪解会找1个25美分硬币和5个1美分硬币共6个硬币，而3个10美分也为30美分，只需3个硬币，故贪婪选择不是最优解。

d. 对于具有最优子结构的问题，可以使用动态规划来解决。

定义c[j]为找j美分需要的最少硬币数。

设硬币面额有$d_1, d_2, \ldots, d_k$，有一个面额为1，故有一个方法可以全找1美分面额硬币。

由于具有最优子结构特性，如果有一个最优解针对问题：找j美分零钱，使用一个面额硬币d_i，则c[j]=1+c[j-d_i]，假设对所有j≤0, c[j]=0

故可以得到问题的表达式：

$$\mathrm{c[j]} = \begin{cases} 0, & if \quad j \le 0 \\ 1 + \min\limits_{1 \le i \le k} \{c[j - d_i]\} & if \quad j > 1 \end{cases}$$

- 利用以下算法解此动态规划问题求最优解：

```
COMPUTE-CHANGE(n, d, k)
  let c[1..n] and denom[1..n] be new arrays
  for j = 1 to n
      c[j] = ∞
      for i = 1 to k
          if j ≥ d_i and 1 + c[j − d_i] < c[j]
              c[j] = 1 + c[j − d_i]
              denom[j] = d_i
  return c and denom
```

- 这个过程的时间花费：O(nk)
- 输出换零钱的方案：

```
GIVE-CHANGE(j, denom)
  if j > 0
      give one coin of denomination denom[j]
      GIVE-CHANGE(j − denom[j], denom)
```