数据结构与算法 II 作业 (10.27)

中国人民大学　信息学院　崔冠宇　2018202147

T15.2-1　对矩阵规模序列 $<5, 10, 3, 12, 5, 50, 6>$，求矩阵链最优括号化方案。

解: 迭代公式：$m[i][j] = \min\{m[i][k] + m[k+1][j] + p_{i-1}p_kp_j\}(i < j)$。

① 链长为 1 的情况，初始化为 0：

$$\text{数组}m[i][j]: \begin{pmatrix} 0 & * & * & * & * & * \\ & 0 & * & * & * & * \\ & & 0 & * & * & * \\ & & & 0 & * & * \\ & & & & 0 & * \\ & & & & & 0 \end{pmatrix}$$

$$\text{数组}s[i][j]: \begin{pmatrix} 0 & * & * & * & * & * \\ & 0 & * & * & * & * \\ & & 0 & * & * & * \\ & & & 0 & * & * \\ & & & & 0 & * \\ & & & & & 0 \end{pmatrix}$$

② 链长为 2 的情况：

$m[1][2] = p_0p_1p_2 = 150$，$k = 1$

$m[2][3] = p_1p_2p_3 = 360$，$k = 2$

$m[3][4] = p_2p_3p_4 = 180$，$k = 3$

$m[4][5] = p_3p_4p_5 = 3000$，$k = 4$

$m[5][6] = p_4p_5p_6 = 1500$，$k = 5$

$$\text{数组}m[i][j]: \begin{pmatrix} 0 & 150 & * & * & * & * \\ & 0 & 360 & * & * & * \\ & & 0 & 180 & * & * \\ & & & 0 & 3000 & * \\ & & & & 0 & 1500 \\ & & & & & 0 \end{pmatrix}$$

$$\text{数组}s[i][j]: \begin{pmatrix} 0 & 1 & * & * & * & * \\ & 0 & 2 & * & * & * \\ & & 0 & 3 & * & * \\ & & & 0 & 4 & * \\ & & & & 0 & 5 \\ & & & & & 0 \end{pmatrix}$$

③ 链长为 3 的情况：

$$m[1][3] = \min\begin{cases} m[1][1] + m[2][3] + p_0p_1p_3 = 960 \\ m[1][2] + m[3][3] + p_0p_2p_3 = 330 \end{cases} = 330\text{，}k = 2$$

$$m[2][4] = \min\begin{cases} m[2][2] + m[3][4] + p_1p_2p_4 = 330 \\ m[2][3] + m[4][4] + p_1p_3p_4 = 960 \end{cases} = 330\text{，}k = 2$$

$$m[3][5] = \min\begin{cases} m[3][3] + m[4][5] + p_2p_3p_5 = 4800 \\ m[3][4] + m[5][5] + p_2p_4p_5 = 930 \end{cases} = 930\text{，}k = 4$$

$$m[4][6] = \min\begin{cases} m[4][4] + m[5][6] + p_3p_4p_6 = 1860 \\ m[4][5] + m[6][6] + p_3p_5p_6 = 6600 \end{cases} = 1860\text{，}k = 4$$

$$\text{数组}m[i][j]: \begin{pmatrix} 0 & 150 & 330 & * & * & * \\ & 0 & 360 & 330 & * & * \\ & & 0 & 180 & 930 & * \\ & & & 0 & 3000 & 1860 \\ & & & & 0 & 1500 \\ & & & & & 0 \end{pmatrix}$$

$$\text{数组}s[i][j]: \begin{pmatrix} 0 & 1 & 2 & * & * & * \\ & 0 & 2 & 2 & * & * \\ & & 0 & 3 & 4 & * \\ & & & 0 & 4 & 4 \\ & & & & 0 & 5 \\ & & & & & 0 \end{pmatrix}$$

④ 链长为 4 的情况：

$$m[1][4]=\min\begin{cases} m[1][1]+m[2][4]+p_0p_1p_4=580 \\ m[1][2]+m[3][4]+p_0p_2p_4=405 \\ m[1][3]+m[4][4]+p_0p_3p_4=630 \end{cases}=405，k=2$$

$$m[2][5]=\min\begin{cases} m[2][2]+m[3][5]+p_1p_2p_5=2430 \\ m[2][3]+m[4][5]+p_1p_3p_5=9360 \\ m[2][4]+m[5][5]+p_1p_4p_5=2830 \end{cases}=2430，k=2$$

$$m[3][6]=\min\begin{cases} m[3][3]+m[4][6]+p_2p_3p_6=2076 \\ m[3][4]+m[5][6]+p_2p_4p_6=1770 \\ m[3][5]+m[6][6]+p_2p_5p_6=1830 \end{cases}=1770，k=4$$

$$数组m[i][j]:\begin{pmatrix} 0 & 150 & 330 & 405 & * & * \\ & 0 & 360 & 330 & 2430 & * \\ & & 0 & 180 & 930 & 1770 \\ & & & 0 & 3000 & 1860 \\ & & & & 0 & 1500 \\ & & & & & 0 \end{pmatrix}$$

$$数组s[i][j]:\begin{pmatrix} 0 & 1 & 2 & 2 & * & * \\ & 0 & 2 & 2 & 2 & * \\ & & 0 & 3 & 4 & 4 \\ & & & 0 & 4 & 4 \\ & & & & 0 & 5 \\ & & & & & 0 \end{pmatrix}$$

⑤ 链长为 5 的情况：

$$m[1][5]=\min\begin{cases} m[1][1]+m[2][5]+p_0p_1p_5=4930 \\ m[1][2]+m[3][5]+p_0p_2p_5=1830 \\ m[1][3]+m[4][5]+p_0p_3p_5=6330 \\ m[1][4]+m[5][5]+p_0p_4p_5=1655 \end{cases}=1655，k=4$$

$$m[2][6]=\min\begin{cases} m[2][2]+m[3][6]+p_1p_2p_6=1950 \\ m[2][3]+m[4][6]+p_1p_3p_6=2940 \\ m[2][4]+m[5][6]+p_1p_4p_6=2130 \\ m[2][5]+m[6][6]+p_1p_5p_6=5430 \end{cases}=1950，k=2$$

$$数组m[i][j]:\begin{pmatrix} 0 & 150 & 330 & 405 & 1655 & * \\ & 0 & 360 & 330 & 2430 & 1950 \\ & & 0 & 180 & 930 & 1770 \\ & & & 0 & 3000 & 1860 \\ & & & & 0 & 1500 \\ & & & & & 0 \end{pmatrix}$$

$$数组s[i][j]:\begin{pmatrix} 0 & 1 & 2 & 2 & 4 & * \\ & 0 & 2 & 2 & 2 & 2 \\ & & 0 & 3 & 4 & 4 \\ & & & 0 & 4 & 4 \\ & & & & 0 & 5 \\ & & & & & 0 \end{pmatrix}$$

⑥ 链长为 6 的情况：

$$m[1][6] = \min \begin{cases} m[1][1] + m[2][6] + p_0p_1p_6 = 2250 \\ m[1][2] + m[3][6] + p_0p_2p_6 = 2010 \\ m[1][3] + m[4][6] + p_0p_3p_6 = 2550 \\ m[1][4] + m[5][6] + p_0p_4p_6 = 2055 \\ m[1][5] + m[6][6] + p_0p_5p_6 = 3155 \end{cases} = 2010，k = 2$$

$$\text{数组} m[i][j] : \begin{pmatrix} 0 & 150 & 330 & 405 & 1655 & 2010 \\ & 0 & 360 & 330 & 2430 & 1950 \\ & & 0 & 180 & 930 & 1770 \\ & & & 0 & 3000 & 1860 \\ & & & & 0 & 1500 \\ & & & & & 0 \end{pmatrix}$$

$$\text{数组} s[i][j] : \begin{pmatrix} 0 & 1 & 2 & 2 & 4 & 2 \\ & 0 & 2 & 2 & 2 & 2 \\ & & 0 & 3 & 4 & 4 \\ & & & 0 & 4 & 4 \\ & & & & 0 & 5 \\ & & & & & 0 \end{pmatrix}$$

所以最小代价为 2010 次乘法，序列为 $((A_1A_2)((A_3A_4)(A_5A_6)))$

T15-2 （最长回文子序列）回文是正序与逆序相同的非空字符串。（例子略）设计高效算法，求给定输入字符串的最长回文子序列。（例子略）算法的运行时间是怎样的?

解: 本题类似于矩阵链最优括号化方案问题。

问题描述：给定一个字符串 S，求它的最长回文子序列。

输入：字符串 S。

输出：最长回文子序列 $S[k_1, k_2, \cdots, k_l]$。

分析：设 $l[i][j](i \leq j)$ 为 $S[i...j]$ 中（含边界）最长回文子序列的长度，$s[i][j](i \leq j)$ 为 $S[i...j]$ 中"最宽"的相同字符下标对 (m, n)（即满足 $i \leq m \leq n \leq j$，$S[m] == S[n]$，并最大化 $n - m$）。以下分析各子问题之间的联系：

1. 当 $i = j$ 时，仅有一个字符，显然 $l[i][j] = 1$，$s[i][j] = (i, i)$；
2. 当 $i < j$ 时，有两种情况：
   (a) 若 $S[i] == S[j]$，前后字符相同，最长回文子串就是去头尾的子问题再加上头尾字符，即
   $l[i][j] = l[i+1][j-1] + 2$，$s[i][j] = (i, j)$；
   (b) 若 $S[i]! = S[j]$，前后字符不同，最长回文子串就是去头的子问题和去尾的子问题中较长的一个，即

$$l[i][j] = \max\{l[i+1][j], l[i][j-1]\}$$

，

$$s[i][j] = \begin{cases} s[i+1][j], & l[i+1][j] \geq l[i][j-1] \\ s[i][j-1], & l[i+1][j] < l[i][j-1] \end{cases}$$

所以，**状态转移方程**：

$$l[i][j] = \begin{cases} 1, & i = j \\ l[i+1][j-1] + 2, & i < j 且 S[i] = S[j] \\ \max\{l[i+1][j], l[i][j-1]\}, & i < j 且 S[i] \neq S[j] \end{cases}$$

$$s[i][j] = \begin{cases} (i, i), & i = j \\ (i, j), & i < j 且 S[i] = S[j] \\ s[i+1][j], & i < j, S[i] \neq S[j], 且 l[i+1][j] \geq l[i][j-1] \\ s[i][j-1], & i < j, S[i] \neq S[j], 且 l[i+1][j] < l[i][j-1] \end{cases}$$

重构解 `PrintSolution(i, j)`：如果 $i = j$，打印 $S[i]$ 即可；否则，只需要打印 $S[s[i][j][0]]$，再调用 PrintSolution($i + 1, j - 1$)，最后打印 $S[s[i][j][1]]$ 即可。

算法伪代码：

```
// 最长回文子序列
// 输入：
//      S: 字符串
// 输出：
//      l[][]: S[i...j] 中最长回文子序列长度
//      s[][]: S[i...j] 中最宽相等字符对下标
//
LONGEST-PALINDROME-SUBSEQUENCE(S):
    n = length(S)
    l[1...n][1...n], s[1...n][1...n]
    // 初始化, O(n)
    for i = 1 to n:
        l[i][i] = 1
        s[i][i] = (i,i)
    // 链长为2~n, O(n)
    for l = 2 to n:
    // 沿主对角线方向填表, O(n^2)
        for i = 1 to n - l + 1:
            j = i + l - 1
            if(S[i] == S[j])
                l[i][j] = l[i + 1][j - 1] + 2
                s[i][j] = (i, j)
```

```
            else if(l[i + 1][j] >= l[i][j - 1])
                l[i][j] = l[i + 1][j]
                s[i][j] = s[i + 1][j]
            else
                l[i][j] = l[i][j - 1]
                s[i][j] = s[i][j - 1]
```

```
// 打印解
// 输入:
//      S[]: 字符串
//      s[][]: S[i...j] 中最宽相等字符对下标
// 输出:
//      回文子序列
PrintSolution(S, s, i, j):
    if(i == j)
        print(S[i])
    else
        print(S[s[i][j][0]])
        PrintSolution(S, s, i + 1, j - 1)
        print(S[s[i][j][1]])
```

LONGEST-PALINDROME-SUBSEQUENCE(S) 的**复杂度分析**:
问题规模 n 为输入字符串的长度。

- 时间复杂度：根据注释，运算次数为 $O(n) + O(n) \times O(n)$，所以 $T(n) = O(n^2)$；
- 空间复杂度：显然有两个 $n \times n$ 辅助矩阵，$S(n) = O(n^2)$。

T15-4 （整齐打印）考虑整齐打印问题，即在打印机上用等宽字符打印一段文本。输入文本为 n 个单词的序列，单词长度分别为 l_1，l_2，$\cdots$，l_n 个字符。我们希望将此段文本整齐打印在若干行上，每行最多 M 个字符。"整齐"的标准是这样的：如果某行包含第 i 到第 j（$i \le j$）个单词，且单词间隔为一个空格符，则行尾的额外空格符数量为 $M - j + i - \sum_{k=i}^{j} l_k$，此值必须为非负的，否则一行内无法容纳这些单词。我们希望能最小化所有行的（除最后一行外）额外空格数的立方之和。设计一个动态规划算法，在打印机上整齐打印一段 n 个单词的文本。分析算法的时间和空间复杂性。

解:

问题描述：给定单词序列和一行文本的字符数上限，确定一种整齐打印的分行方案，使得除最后一行外，行尾的额外空格数的立方和最小。

输入：单词序列 $< t_1, t_2, \cdots, t_n >$，它们的长度分别为 $< l_1, l_2, \cdots, l_n >$，再给定每行字符数上限 M。

输出：分行打印文本，即 $<< t_1, t_2, \cdots, t_{l_1} >, \texttt{'\textbackslash n'}, < t_{l_1+1}, t_{l_1+2}, \cdots, t_{l_2} >, \texttt{'\textbackslash n'}, \cdots >$。这种打印方式使得除最后一

行外，行尾额外空格数的立方和最小。

分析：

首先必须假定 $M \geq \max\{l_i\}(i = 1, 2, \cdots, n)$，保证最长的单词能被一行容纳。对于某一行，假定它是由第 i 至第 j 个单词构成的。

创建二维数组 $le[i][j]$ 来保存每行（由单词 $t_i \sim t_j$ 构成）额外空格数，则

$$le[i][j] = M - j + i + \sum_{k=i}^{j} l_k$$

创建二维数组 $lc[i][j]$ 保存修正后的每行（由单词 $t_i \sim t_j$ 构成）的额外空格数的立方，则对应于 $le[i][j]$ 的不同，$lc[i][j]$ 的取值分为三种情况：

1. 若 $le[i][j] < 0$，表示单词 $t_i \sim t_j$ 放在一行会超出最大字符数。这种情况不允许出现，所以将其惩罚为 $+\infty$；
2. 若 $j = n$，表示最后一行，此时不计算最后一行额外空格数，所以设为 0；
3. 其它情况，设为 $(le[i][j])^3$ 即可。

所以，

$$lc[i][j] \begin{cases} +\infty, & le[i][j] < 0 \\ 0, & le[i][j] \geq 0 \text{且} j = n \\ (le[i][j])^3, & le[i][j] \geq 0 \text{且} j < n \end{cases}$$

接下来创建 $c[j]$ 保存前 j 个单词 $t_1 \sim t_j$ 的最优额外空格数立方和 [1]。假设将之前的单词从 $t_i(1 \leq i \leq j)$ 分行，则根据 $c[j]$ 的含义，$c[j]$ 应该是诸 $c[i-1] + lc[i][j](1 \leq i \leq j)$ 的最小值，即：

$$c[j] = \begin{cases} 0, & j = 0 \\ \min\limits_{1 \leq i \leq j} \{c[i-1] + lc[i][j]\}, & j > 0 \end{cases}$$

为了重构解，还需要定义 $p[j]$ 来记录上述取得最小 $c[j]$ 时的 i，即最后一行的起始单词下标。

重构解 `PrintSolution(t, p, j)`：如果 $j = 1$，打印 $t[1]$ 即可；否则，先递归地打印 $t[1...p[j]-1]$，接着打印最后一行 $t[p[j]...j]$ 即可。

算法伪代码：

```
// 整齐打印
// 输入:
//        l: 单词长度数组
//        M: 行最大字符数
// 输出:
```

[1]注：当 $j < n$ 时，则需要包含最后一行的空格数立方；当 $j = n$ 则不包含。

```
//      c[]: 打印前 j 个单词最优额外空格数立方和
//      p[]: 打印前 j 个单词最优方案最后一行起始单词下标
//
PRINT-NEATLY(l, M):
    n = l.length
    le[1...n][1...n], lc[1...n][1...n]
    c[0...n], p[1...n]
    // 根据 l, 计算 le 数组, O(n^2)
    for i = 1 to n:
        le[i][i] = M - l[i]
        for j = i + 1 to n:
            le[i][j] = le[i][j - 1] - l[j] - 1
    // 根据 le, 计算 lc, O(n^2)
    for i = 1 to n:
        for j = i to n:
            if le[i][j] < 0:
                lc[i][j] = +INFINITY
            else if j == n:
                lc[i][j] = 0
            else:
                lc[i][j] = (le[i][j])^3
    // 计算c, O(n^2)
    c[0] = 0
    for j = 1 to n:
        c[j] = +INFINITY
        for i = 1 to j:
            if c[i - 1] + lc[i][j] < c[j]:
                c[j] = c[i - 1] + lc[i][j]
                p[j] = i
    return c and p

// 打印解
// 输入:
//      t[]: 单词数组
//      p[]: 打印前 j 个单词最优方案最后一行起始单词下标
//      j: 打印单词数
// 输出:
```

```
//        最佳打印方案
//
PrintSolution(t, p, j):
    // 一个单词，直接打印
    if j == 1:
        print(t[1])
    // 递归打印前面的内容
    PrintSolution(t, p, p[j] - 1)
    // 打印最后一行
    print(t[p[j]...j])
```

`PRINT-NEATLY(l, M)` 的**复杂度分析**：
问题规模 n 为单词的个数。

- 时间复杂度：`PRINT-NEATLY` 中有三个双重循环，其中任何一个双重循环都是 $O(n^2)$ 的，故总时间复杂度为 $T(n) = O(n^2)$；
- 空间复杂度：`PRINT-NEATLY` 内部定义了 $le[1...n][1...n]$、$le[1...n][1...n]$、$c[0...n]$ 以及 $p[1...n]$ 四个数组，共有 $n \times n + n \times n + (n+1) + n = 2n^2 + 2n + 1 = O(n^2)$ 个辅助空间，所以总空间复杂度 $S(n) = O(n^2)$。