算法作业答案

—杨文建

作业一：

1（3.1-1） 假设$f(n)$与$g(n)$都是渐近非负函数。使用θ记号的基本定义来证明$\max\big(f(n),g(n)\big)=\theta\big(f(n)+g(n)\big)$。

证明：因为$f(n)$与$g(n)$都是渐近非负函数

根据定义有：存在N_f，N_g使得：当$n\ge N_f$时，$f(n)\ge 0$，同时，当$n\ge N_g$时，$g(n)\ge 0$。

所以，我们取$N_0=max\big(N_f,N_g\big)$，此时，当$n\ge N_0$时，同时有$f(n)\ge 0$，$g(n)\ge 0$。

取$C_1=1/2$，$C_2=1$，则当$n\ge N_0$时，有：

$\big(f(n)+g(n)\big)/2\le \max\big(f(n),g(n)\big)\le f(n)+g(n)$

得证

2（3.1-4） $2^{n+1}=O(2^n)$成立吗？$2^{2n}=O(2^n)$成立吗？

解：1）$2^{n+1}=O(2^n)$成立

取$\mathrm{c}\ge 2$，当$\mathrm{n}\ge 1$时，

有$2^{n+1}=2*2^n\le c*2^n$

因此，成立

2）$2^{2n}=O(2^n)$不成立

如果$2^{2n}=O(2^n)$成立，根据定义有：$2^{2n}\le c2^n$（c为常数）

得，$n\le \lg c$，则$2^{2n}\le c2^n$对于任意大的n不成立，与假设矛盾

因此，$2^{2n}=O(2^n)$不成立

3 求下列函数的渐近表达式：

$3n^2+10n$；$n^2/10+2^n$；$21+1/n$；$\log n^3$；$10\log 3^n$。

解：$3n^2+10n=O(n^2)$

$n^2/10+2^n=O(2^n)$

$21+1/n=O(1)$

$\log n^3=O(\log n)$

$10\log 3^n=O(n)$

4 试讨论 $O(1)$与$O(2)$的区别

解：根据符号O的定义易知$O(1)=O(2)$。用$O(1)$或$O(2)$表示用一个函数时，差别仅在于其中的常数因子。

5 （1）假设某算法在输入规模上为n时的计算时间为$T(n)=3*2^n$。在某台计算机上实现并完成该算法的时间为t秒。现有另一台计算机，其运行速度为第一台的 64 倍，那么在这台新机器上用同一算法在t秒内能输入规模为多大的问题？

（2）若上述算法的计算时间改进为$T(n)=n^2$，其余条件不变，则在新机器上用t秒时间能解输入规模为多大的问题？

（3）若上述算法的计算时间进一步改进为 8，其余条件不变，那么在新机器上用t秒时间能解输入规模为多大的问题？

解：（1）设新机器用同一算法在t秒内能解输入规模为n_1的问题。因此有，$t=3*2^n=3*$

$2^{n_1}/64$，解得$n_1 = n + 6$。

（2）同（1），$t = n^2 = n_1^2/64$，解得，$n_1 = 8n$。

（3）由于$T(n) = 8$为常数阶，因此算法可解任意规模的问题。

6 对于下列各组函数$f(n)$和$g(n)$，确定$f(n) = O\big(g(n)\big)$或$f(n) = \Omega\big(g(n)\big)$或$f(n) = \theta\big(g(n)\big)$，并简述理由。

（1）$f(n) = \log n^2$；$g(n) = \log n + 5$ （5）$f(n) = 10$；$g(n) = \log 10$

（2）$f(n) = \log n^2$；$g(n) = \sqrt{n}$ （6）$f(n) = \log^2 n$；$g(n) = \log n$

（3）$f(n) = n$；$g(n) = \log^2 n$ （7）$f(n) = 2^{\mathrm{n}}$；$g(n) = 100\mathrm{n}^2$

（4）$f(n) = n\log n + n$；$g(n) = \log n$ （8）$f(n) = 2^{\mathrm{n}}$；$g(n) = 3^{\mathrm{n}}$

解：（1）$\log n^2 = \theta(\log n + 5)$ （5）$10 = \theta(\log 10)$

（2）$\log n^2 = O\big(\sqrt{n}\big)$ （6）$\log^2 n = \Omega(\log n)$

（3）$n = \Omega(\log^2 n)$ （7）$2^{\mathrm{n}} = \Omega(100\mathrm{n}^2)$

（4）$n\log n + n = \Omega(\log n)$ （8）$2^{\mathrm{n}} = O(3^{\mathrm{n}})$

7 证明：$n! = \mathrm{o}(n^n)$

证明：由 stirling 公式得，$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^{\mathrm{n}}\left[1 + \theta\left(\frac{1}{n}\right)\right]$

$$\lim_{n\to\infty} n!/n^n = \lim_{n\to\infty} \frac{\sqrt{2\pi n}\left[1+\theta\left(\frac{1}{n}\right)\right]}{e^n} = 0$$

所以，$n! = \mathrm{o}(n^n)$。

证毕

8 证明：当$a_k > 0$时，任何多项式$P(n) = a_k n^k + a_{k-1}n^{k-1} + \cdots + a_0$属于集合$\theta(n^k)$。

证明：要证，$P(n)$属于集合$\theta(n^k)$

只需证，存在N_0，C_1，C_2，使得：当$n \ge N_0$时，有$0 \le C_1 n^k \le P(n) \le C_2 n^k$。

1）
$$\begin{aligned} P(n) &= a_k n^k + a_{k-1}n^{k-1} + \cdots + a_0 \\ &\ge a_k n^k - (|a_{k-1}| + \cdots + |a_0|)n^{k-1} \\ &\ge \left(a_k - \frac{(|a_{k-1}|+\cdots+|a_0|)}{n}\right)n^k \end{aligned}$$

令$a_k - \frac{(|a_{k-1}|+\cdots+|a_0|)}{n} \ge 0$，得$n \ge \frac{(|a_{k-1}|+\cdots+|a_0|)}{a_k}$

取$N_1 = \frac{(|a_{k-1}|+\cdots+|a_0|)}{a_k} + 1$，$C_1 = a_k - \frac{(|a_{k-1}|+\cdots+|a_0|)}{N_1} > 0$，当$n \ge N_1$时，有

$$0 \le C_1 n^k \le P(n)$$

2）
$$\begin{aligned} P(n) &= a_k n^k + a_{k-1}n^{k-1} + \cdots + a_0 \\ &\le (|a_k| + |a_{k-1}| + \cdots + |a_0|)n^k \end{aligned}$$

取$N_2 = 1$，$C_2 = |a_k| + |a_{k-1}| + \cdots + |a_0|$，当$n \ge N_2$时，有

$$P(n) \le C_2 n^k$$

综上 1)，2)，取$N_0 = \max(N_1, N_2)$，当$n \ge N_0$时，有

$$0 \le C_1 n^k \le P(n) \le C_2 n^k$$

所以，当$a_k > 0$时，任何多项式$P(n) = a_k n^k + a_{k-1}n^{k-1} + \cdots + a_0$属于集合$\theta(n^k)$。

证毕

作业 2：

1 假设有n段待分配的频谱 spectrum，m个频谱需求者。每段频谱有一个干扰半径r，每个频谱需求者有一个期望接入频谱的地理位置 Location。当两个频谱用户之间的距离(欧氏距离)小于r时，不能共享同一段频谱；否则，用户同时使用时会相互干扰。现需要找到一种频谱分配方式，在互不干扰的前提下，将频谱分配给尽可能多的频谱需求者。请对该问题进行规约。

2 证明 Partition Problem 可以在多项式时间内规约为 0-1 knapsack Problem 的判定版本。

作业 3：

1（4.3-1）证明：$T(n)=T(n-1)+n$的解为$O(n^2)$。

证明：假设对$\forall m<n$，$\exists c>0$，使得：

$$T(m)\le \mathrm{cm}^2$$

则有：

$$T(n-1)\le c(n-1)^2$$

带入迭代式可得：

$$\begin{aligned}T(n)&\le c(n-1)^2+n\\&=cn^2-2cn+c+n\end{aligned}$$

令$-2cn+c+n\le 0$，可得：

$$c\ge \frac{n}{2n+1}$$

故对于$\forall n>0$，可令$c=1$，使得

$$T(n)\le cn^2-2cn+c+n\le cn^2$$

故$T(n)=O(n^2)$。

2（4.3-6） 证明：$T(n)=2T(\lfloor n/2\rfloor+17)+n$的解为$\mathrm{O}(n\lg n)$。

往证：存在常数N_0，C_0，使得：当$n\ge N_0$时，$T(n)\le C_0 n\lg n$。

假设：$T(\lfloor n/2\rfloor+17)\le C_0(\lfloor n/2\rfloor+17)\lg(\lfloor n/2\rfloor+17)$

有：$T(n)\le 2C_0(\lfloor n/2\rfloor+17)\lg(\lfloor n/2\rfloor+17)+\mathrm{n}$

$$\begin{aligned}&\le 2C_0(n/2+17)\lg(n/2+17)+\mathrm{n}\\&=C_0(n+34)[\lg(n+34)-1]+\mathrm{n}\\&=C_0 n\lg(n+34)-C_0 n+34C_0\lg(n+34)-34C_0+\mathrm{n}\qquad(*1)\end{aligned}$$

取$C_0\ge 2$

则有$(*1)$ $\le C_0 n\lg(n+34)+34C_0\lg(n+34)-34C_0-\mathrm{n}$

$$\begin{aligned}&< C_0 n\lg(n+34)+34C_0\lg(n+34)-\mathrm{n}\\&=C_0 n\lg n+C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\qquad(*2)\end{aligned}$$

下面我们只需证明存在常数N_0，$C_0\ge 2$，使得当$n\ge N_0$时，

有 $C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\le 0$

即有： $(*2)\le C_0 n\lg n$

$$\begin{aligned}&C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\\&=C_0 n(\lg(n+34)-\lg n)+34C_0\lg(n+34)-\mathrm{n}\\&=C_0 n(\ln(1+34/n)/\ln 2)+34C_0\lg(n+34)-\mathrm{n}\\&\le C_0 n(34/(n\ln 2))+34C_0\lg(n+34)-\mathrm{n}\end{aligned}$$

此时只需取$C_0=3$，$N_0=100$，即可使当$n\le N_0$时，有

$C_0 n\lg(n+34)-C_0 n\lg n+34C_0\lg(n+34)-\mathrm{n}\le 0$

原题得证

3（4.4-4）对递归式$T(n)=T(n-1)+1$，利用递归树确定一个好的渐进上界，用代入法进行验证。

解：递归树如下，树高为$n-1$，叶节点数目为 1，整棵树的代价为：

$$T(n)=1+1+1+\cdots 1+\theta(1)=n-1+\theta(1)=O(n)$$

代入法验证：

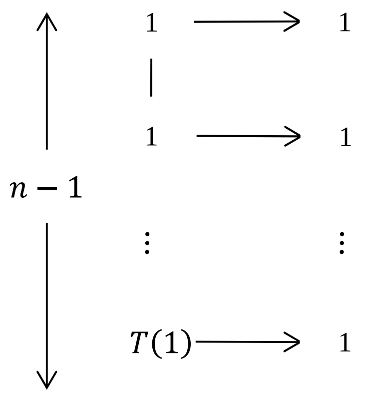
令对于$\forall m<n$，$\exists c>0$，使$T(m)\le cm$，有

$$
\begin{aligned}
T(n) &= T(n-1)+1 \\
&\le c(n-1)+1 \\
&= cn-c+1
\end{aligned}
$$

故对于$\forall n>0$，可令$c=1$，使得

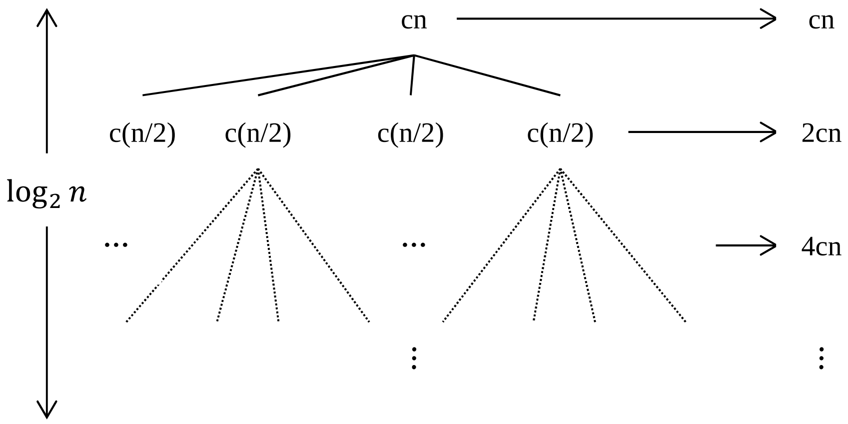
$$T(n) \le cn-c+1 \le cn$$

所以，$T(n)=O(n)$。



4（4.4-7） 对递归式$T(n)=4T(\lfloor n/2 \rfloor)+cn$（$c$为常数），画出递归树，并给出其解的一个渐近紧缺界。用代入法验证。

证明：



递归树的高度为$\log_2 n$，非叶子节点的度为 4，每一层的贡献为$4^i \lfloor cn/2^i \rfloor$。

则

$$
\begin{aligned}
T(n) &= 4T(\lfloor n/2 \rfloor)+cn \\
&= \sum_{i=0}^{\log_2 n} 4^i \lfloor cn/2^i \rfloor
\end{aligned}
$$

1)

$$
\begin{aligned}
T(n) &\le \sum_{i=0}^{\log_2 n} 4^i cn/2^i \\
&= cn\sum_{i=0}^{\log_2 n} 2^i \\
&= cn\frac{2^{\log_2 n+1}-1}{2-1}
\end{aligned}
$$

$= O(n^2)$

2) $$T(n) \geq \sum_{i=0}^{\log_2 n} 4^i\left(cn/2^i - 1\right)$$

$$= cn\sum_{i=0}^{\log_2 n} 2^i - \sum_{i=0}^{\log_2 n} 4^i$$

$$= cn\frac{2^{\log_2 n+1}-1}{2-1} - \frac{4^{\log_2 n+1}-1}{4-1}$$
$$= 2cn^2 - cn - 4/3n^2 + 1/3$$
$$= (2c - 4/3)n^2 - cn + 1/3$$
$$= \Omega(n^2)$$

综上 1）、2），得$T(n) = \theta(n^2)$

用代入法验证

a) 令$T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor^2 - c\lfloor n/2 \rfloor$

有
$$\begin{aligned} T(n) &= 4T(\lfloor n/2 \rfloor) + cn \\ &\leq 4(c\lfloor n/2 \rfloor^2 - c\lfloor n/2 \rfloor) + cn \\ &< 4c(n/2)^2 - 4c(n/2) + cn \\ &= cn^2 - 2cn + cn \\ &= cn^2 - cn \end{aligned}$$

b) 令$T(\lfloor n/2 \rfloor) \geq c\lfloor n/2 \rfloor^2 + 3c\lfloor n/2 \rfloor + 3c$

有
$$\begin{aligned} T(n) &= 4T(\lfloor n/2 \rfloor) + cn \\ &\geq 4(c\lfloor n/2 \rfloor^2 + 3c\lfloor n/2 \rfloor + c) + cn \\ &> 4c(n/2-1)^2 + 12c(n/2-1) + 4c + cn \\ &= cn^2 - 4cn + 4c + 6cn - 12c + 12c + cn \\ &= cn^2 + 3cn + 4c \\ &> cn^2 + 3cn + 3c \end{aligned}$$

5（4.2-1） 使用 Strassen 算法计算如下矩阵乘法：

$$\begin{bmatrix} 1 & 3 \\ 7 & 5 \end{bmatrix}\begin{bmatrix} 6 & 8 \\ 4 & 2 \end{bmatrix}$$

给出计算过程。

解： $A_{11} = 1$，$A_{12} = 3$，$A_{21} = 7$，$A_{22} = 5$

$B_{11} = 6$，$B_{12} = 8$，$B_{21} = 4$，$B_{22} = 2$

$S_1 = B_{12} - B_{22} = 8 - 2 = 6$

$S_2 = A_{11} + A_{12} = 1 + 3 = 4$

$S_3 = A_{21} + A_{22} = 7 + 5 = 12$

$S_4 = B_{21} - B_{11} = 4 - 6 = -2$

$S_5 = A_{11} + A_{22} = 1 + 5 = 6$

$S_6 = B_{11} + B_{22} = 6 + 2 = 8$

$S_7 = A_{12} - A_{22} = 3 - 5 = -2$

$S_8 = B_{21} + B_{22} = 4 + 2 = 6$

$S_9 = A_{11} - A_{21} = 1 - 7 = -6$

$S_{10} = B_{11} + B_{12} = 6 + 8 = 14$

$P_1 = A_{11}S_1 = 1 * 6 = 6$

$P_2 = S_2B_{22} = 4 * 2 = 8$

$P_3 = S_3 B_{11} = 12 * 6 = 72$
$P_4 = A_{22} S_4 = 5 * (-2) = -10$
$P_5 = S_5 S_6 = 6 * 8 = 48$
$P_6 = S_7 S_8 = -2 * 6 = -12$
$P_7 = S_9 S_{10} = -6 * 14 = -84$
$C_{11} = P_5 + P_4 - P_2 + P_6 = 48 + (-10) - 8 + (-12) = 18$
$C_{12} = P_1 + P_2 = 6 + 8 = 14$
$C_{21} = P_3 + P_4 = 62$
$C_{22} = P_5 + P_1 - P_3 - P_7 = 48 + 6 - 72 - (-84) = 66$

6（4.2-2） 为 Strassen 算法编写伪代码
解：

算法 1 Strassen 算法

Strassen 算法

输入：矩阵A，B
输出：A，B的乘积矩阵C

Strassen(A，B，C)
  if A 和 B 的维数足够小
    计算$C \leftarrow A * B$ //直接求解//
  else
    //将n维矩阵 A，B 划分为$n/2$维矩阵A_{11}，A_{12}，A_{21}，A_{22}，B_{11}，B_{12}，B_{21}，B_{22}//
    $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \leftarrow$ A
    $\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \leftarrow$ B
    //10 个加法//
    $S_1 \leftarrow B_{12} - B_{22}$
    $S_2 \leftarrow A_{11} + A_{12}$
    $S_3 \leftarrow A_{21} + A_{22}$
    $S_4 \leftarrow B_{21} - B_{11}$
    $S_5 \leftarrow A_{11} + A_{22}$
    $S_6 \leftarrow B_{11} + B_{22}$
    $S_7 \leftarrow A_{12} - A_{22}$
    $S_8 \leftarrow B_{21} + B_{22}$
    $S_9 \leftarrow A_{11} - A_{21}$
    $S_{10} \leftarrow B_{11} + B_{12}$
    //7 个乘法，递归调用 Strassen//
    Strassen(A_{11}，S_1，P_1)
    Strassen(S_2，B_{22}，P_2)
    Strassen(S_3，B_{11}，P_3)
    Strassen(A_{22}，S_4，P_4)
    Strassen(S_5，S_6，P_5)
    Strassen(S_7，S_8，P_6)

Strassen(S_9，S_{10}，P_7)
//计算$n/2$维矩阵C_{11}，C_{12}，C_{21}，C_{22}//
$C_{11} \leftarrow P_5 + P_4 - P_2 + P_6$
$C_{12} \leftarrow P_1 + P_2$
$C_{21} \leftarrow P_3 + P_4$
$C_{22} \leftarrow P_5 + P_1 - P_3 - P_7$
//将C_{11}，C_{12}，C_{21}，C_{22}合并为C//
$C \leftarrow \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$

7（4.5-3） 使用主方法证明：二分查找递归式$T(n) = T(n/2) + \theta(1)$的解是$T(n) = \theta(\lg n)$。
证明：$a = 1$，$b = 2$，$n^{\log_b a} = 1$

因此，$f(n) = \theta(1) = \theta\left(n^{\log_b a}\right)$

故，$T(n) = \theta\left(n^{\log_b a} \lg n\right) = \theta(\lg n)$

证毕

8（4.6-3） 证明：主定理中的情况 3 被过分强调了，从某种意义上来说，对某个常数$c < 1$，正则条件$af(n/b) \le cf(n)$成立本身就意味着存在常数$\varepsilon > 0$，使得$f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$。
证明：主函数成立有一个条件：$T(n)$是必须是单调增函数，并且$f(n)$必须是多项式函数。
由递推关系，得

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^k) \tag{1}$$

将$n = b^m$带入等式(1)，得

$$T(b^m) = aT\left(\frac{b^m}{b}\right) + O(b^m)^k$$
$$T(b^m) = aT(b^{m-1}) + O(b^k)^m \tag{2}$$

将等式(2)两边同除以a^m，有

$$\frac{T(b^m)}{a^m} = \frac{aT(b^{m-1})}{a^m} + \frac{O(b^k)^m}{a^m}$$
$$\frac{T(b^{m-1})}{a^{m-1}} = \frac{T(b^{m-2})}{a^{m-2}} + \left(\frac{b^k}{a}\right)^{m-1}$$
$$\cdots$$
$$\frac{T(1)}{1} = 1 \tag{3}$$

等式(3)中各式相加，得

$$T(b^m) = a^m \times \sum_{i=0}^{m} \left(\frac{b^k}{a}\right)^i \tag{4}$$

对于主定理中的情况 3，有
如果$a < b^k$

$$T(b^m) = a^m \times \sum_{i=0}^{m} \left(\frac{b^k}{a}\right)^i$$

$$\approx a^m \times \frac{\left(\frac{b^k}{a}\right)^{m+1}}{\frac{b^k}{a}}$$

$$= (b^k)^m$$

所以，$T(n) = O(n^k)$（$k > \log_b a$）

因此，总的来说，如果$f(n) = \Omega\left(n^{\log_b a+\varepsilon}\right)$（$\varepsilon > 0$），并且有对某个常数$c < 1$，正则条件$af(n/b) \le cf(n)$成立，则有，$T(n) = f(n)$。

9（7.2-1）利用代入法证明：正如 7.2 节开头提到的那样，递归式$T(n) = T(n-1) + \theta(n)$的解为$T(n) = \theta(n^2)$。

证明：1）假设对于$\forall m_1 < n$，$\exists c_1 > 0$，使得：$T(m_1) \le c_1 {m_1}^2$，有

$$T(n) = T(n-1) + \theta(n)$$
$$\le c_1(n-1)^2 + b_1 n (b_1 > 0)$$
$$= c_1 n^2 - 2c_1 n + c_1 + b_1 n$$

当$c_1 \ge b_1$时，$-2c_1 n + c_1 + b_1 n \le 0$，有

$$T(n) = c_1 n^2 - 2c_1 n + c_1 + b_1 n \le c_1 n^2$$

2）假设对于$\forall m_2 < n$，$\exists c_2 > 0$，使得：$T(m_2) \ge c_2 {m_2}^2 \ge 0$，有

$$T(n) = T(n-1) + \theta(n)$$
$$\ge c_2(n-1)^2 + b_2 n (b_2 > 0)$$
$$= c_2 n^2 - 2c_2 n + c_2 + b_2 n$$

当$c_2 \le \frac{b_2}{2}$时，$-2c_2 n + c_2 + b_2 n > 0$，有

$$T(n) = c_2 n^2 - 2c_2 n + c_2 + b_2 n \ge c_2 n^2$$

综上 1）、2），得递归式$T(n) = T(n-1) + \theta(n)$的解为$T(n) = \theta(n^2)$。

作业四：

1 动态规划和分治算法有什么共同点？这两种技术之间最主要的不同点是什么？

答：1）分治法与动态规划主要共同点：

二者都要求原问题具有最优子结构性质，都是将原问题分而治之，分解成若干个规模较小(小到很容易解决的程序)的子问题。然后将子问题的解合并，形成原问题的解。

2）分治法与动态规划主要区别：

分治法将分解后的子问题看成相互独立的。

动态规划将分解后的子问题理解为相互间有联系，有重叠部分。

2（15.2-1） 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：由题意得

矩阵	A_1	A_2	A_3	A_4	A_5	A_5
规模	5×10	10×3	3×12	12×5	5×50	50×6

其中$< p_0, p_1, p_2, p_3, p_4, p_5, p_6 > = < 5,10,3,12,5,50,6 >$

利用公式计算 m 表：

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

计算得下表：

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

相应的 s 表为：

$i\backslash j$	2	3	4	5	6
1	1	2	2	4	2
2		2	2	2	2
3			3	4	4
4				4	4
5					5

易得，矩阵链最优括号化方案为$\big((A_1A_2)\big((A_3A_4)(A_5A_6)\big)\big)$。

3（15.2-3） 用代入法证明递归公式(15.6)的结果为$\Omega(2^n)$。

证明：递归公式(15.6)如下：

$$P(n) = \begin{cases} 1 & \text{如果} n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{如果} n \ge 2 \end{cases}$$

即证，$P(n) = \Omega(2^n)$，亦即证，存在正常数c和n_0，当$n \ge n_0$时，使$P(n) \ge c2^n$成立。

当$n=1$时，有$P(n)=1$，成立

假设当$n<m$时，有$P(n)\geq c2^n$成立

则当$n=m$时，有$P(m)\geq \sum_{k=1}^{m-1}P(k)P(m-k)$

$$\geq \sum_{k=1}^{m-1}c2^k c2^{m-k}$$

$$\geq c^2(m-1)2^m$$

取$n_0=\frac{1}{c}+1$，当$n\geq n_0$时，有$P(m)\geq c2^m$

证毕。

4（15.3-3） 考虑矩阵链乘法问题的一个变形：目标改为最大化矩阵序列括号化方案的标量乘法运算次数，而非最小化。此问题具有最优子结构性质吗？

解：这个问题具有最优子结构。

分析如下：

采用动态规划法的第一步就是寻找最优子结构，然后利用这一子结构，就可以根据子问题的最优解构造出原问题的一个最优解。

用记号$\boldsymbol{A_{i\ldots j}}$表示对乘积$\boldsymbol{A_i A_{i+1}\cdots A_j}$的求积结果，其中$\boldsymbol{i<j}$。

这个问题的最优子结构如下：假设$\boldsymbol{A_i A_{i+1}\cdots A_j}$的一个最优括号化方案把$\boldsymbol{A_k}$与$\boldsymbol{A_{k+1}}$之间分开，则对$\boldsymbol{A_i A_{i+1}\cdots A_j}$最优括号化方案的“前缀”子链$\boldsymbol{A_i A_{i+1}\cdots A_k}$的括号化方案必须是$\boldsymbol{A_i A_{i+1}\cdots A_k}$的一个最优括号化方案，因为如果对括号化方案$\boldsymbol{A_i A_{i+1}\cdots A_k}$有一个代价更大的括号化方案，那么把它替换到$\boldsymbol{A_i A_{i+1}\cdots A_j}$的最优括号化方案中就会产生$\boldsymbol{A_i A_{i+1}\cdots A_j}$的另一种括号化方案，而它会具有更大的代价，产生矛盾。

所以，判断这个问题具有最优子结构。

5（15.4-3） 设计 LCS-LENGTH 的带备忘的版本，运行时间为$O(mn)$。

解：假设 X、Y 为需要求解最长公共子序列的字符串，$c[i,j]$为 X 中前 i 个字符，Y 中前 j 个字符的最长公共子序列的长度

利用如下公式设计算法：

$$c[i,j]=\begin{cases}0 & if\ \ i=0\ \ or\ \ j=0\\ c[i-1,j-1]+1 & if\ \ i,j>0\ \ and\ \ x_i=y_i\\ \max\{c[i,j-1],c[i-1,j]\} & if\ \ i,j>0\ \ and\ \ x_i\neq y_i\end{cases}$$

```
int c[1005][1005];//存储 X 中前 i 个字符，Y 中前 j 个字符的最长公共子序列的长度

int LOOKUP_LENGTH(char *X, char *Y, int i, int j)
//求解 c[i][j]
{
    if (c[i][j] > -1)
    {
        return c[i][j];
    }

    if (i == 0 || j == 0)
    {
```

```
            c[i][j] = 0;
        }
        else
        {
            if (X[i - 1] == Y[j - 1])//X 中第 i 个字符与 Y 中第 j 个字符相等
            {
                c[i][j] = LOOKUP_LENGTH(X, Y, i - 1, j - 1) + 1;
            }
            else                            //X 中第 i 个字符与 Y 中第 j 个字符不相等
            {
                LOOKUP_LENGTH(X, Y, i, j - 1);
                LOOKUP_LENGTH(X, Y, i - 1, j);
                c[i][j] = c[i][j - 1] > c[i - 1][j] ? c[i][j - 1] : c[i - 1][j];
            }
        }

        return c[i][j];
}

int LCS_LENGTH(char *X, char *Y, int m, int n)
//求解 X、Y 的最长公共子序列的长度
{
        int i, j;

        for (i = 0; i <= m; i++)
        {
            for (j = 0; j <= n; j++)
            {
                c[i][j] = -1;    //初始化
            }
        }

        return LOOKUP_LENGTH(X, Y, m, n);
}
```

6（15.4-5） 设计一个$O(n^2)$时间的算法，求一个n个数的序列的最长单调递增子序列。

解：假设序列为x，$c[i]$为以$x[i]$结尾的单调递增子序列的长度，$pre[i]$为以$x[i]$结尾的单调递增子序列中$x[i]$前一个字符的位置

利用如下公式设计算法：

$$c[i] = max\left\{\max_{1\leq j<i\&x[i]>x[j]}\{c[j]\}+1,1\right\}$$

$$pre[i] = \begin{cases} 0 \quad if \;\; c[i]=1 \\ j \quad if \;\; c[i]>1 and \;\; x[j] is \;\; the \;\; prevous \;\; number \;\; of \;\; x[i] \end{cases}$$

```
int c[1005], pre[1005], longest_subsequence[1005];//longest_subsequence 存储最长单调递增子序列

void longest_monotonically_increasing_subsequence(int *x, int n)
//求解最长单调递增子序列
{
	int i, j;
	for (i = 1; i <= n; i++)
	{
		c[i] = 1;
		pre[i] = 0;
		for (j = 1; j < i; j++)
		{
			if (x[i] > x[j] && c[i] < c[j] + 1)
			{
				c[i] = c[j] + 1;
				pre[i] = j;
			}
		}
	}

	int max = c[1];
	int max_index = 1;
	for (i = 2; i <= n; i++)
	{
		if (max < c[i])
		{
			max = c[i];
			max_index = i;
		}
	}
	j = 0;
	while (pre[max_index])
	{
		longest_subsequence[j++] = max_index;
		max_index = pre[max_index];
	}
	longest_subsequence[j++] = max_index;

	printf("最长单调递增子序列的长度为：%d\n", j);
	printf("最长单调递增子序列为：\n");
	for (i = j - 1; i >= 0; i--)
	{
		printf("%d ", longest_subsequence[i]);
```

```
	}
	printf("\n");
}
```

7 A 和 B 是两个字符串，用最少的字符操作将 A 转换为 B，操作包括：删除 1 个字符、插入 1 个字符、替换 1 个字符。

解：假设$c[i,j]$为 A 中前 i 个字符，B 中前 j 个字符的最少的字符操作。

利用如下公式设计算法：

$$c[i,j] = \begin{cases} 0 & if\ \ i=0 \\ c[i-1,j-1] & if\ \ i,j>0\ \ an \\ \min\{c[i,j-1],c[i-1,j],c[i-1,j-1]\}+1 & if\ \ i,j>0\ \ and\ \ x_i \neq y_i \end{cases}$$

```
#include <iostream>
#include <cstdio>
#include <cstring>

using namespace std;

#define N 1005

char input_str[N];
int len_input_str;
char output_str[N];
int len_output_str;
int c[N][N];
int b[N][N];

void input()
{
	gets(&input_str[1]);
	len_input_str = strlen(&input_str[1]);
	gets(&output_str[1]);
	len_output_str = strlen(&output_str[1]);
}

int Min(int x, int y, int z, int i, int j)
{
	if (x <= y && x <= z)
	{
		b[i][j] = 1;
		return x;
	}
	else if (y < z)
	{
```

```
        b[i][j] = 2;
        return y;
    }
    else
    {
        b[i][j] = 3;
        return z;
    }
}

void solve()
{
    int i, j;

    for (j = 0; j <= len_output_str; j++)
    {
        c[0][j] = j;
        b[0][j] = 2;
    }
    for (i = 0; i <= len_input_str; i++)
    {
        c[i][0] = i;
        b[i][0] = 3;
    }

    for (i = 1; i <= len_input_str; i++)
    {
        for (j = 1; j <= len_output_str; j++)
        {
            if (input_str[i] == output_str[j])
            {
                c[i][j] = c[i - 1][j - 1];
                b[i][j] = 0;
            }
            else
            {
                c[i][j] = Min(c[i - 1][j - 1], c[i][j - 1], c[i - 1][j], i, j) + 1;
            }
        }
    }
}

void dfs(int i, int j)
{
```

```
	if (!i && !j)
	{
		return;
	}

	if (!b[i][j])
	{
		dfs(i - 1, j - 1);
	}
	else if (b[i][j] == 1)
	{
		dfs(i - 1, j - 1);
		printf("将X中第%d位置的字符%c替换为Y中第%d位置的字符%c\n", i, input_str[i], j, output_str[j]);
	}
	else if (b[i][j] == 2)
	{
		dfs(i, j - 1);
		printf("将字符%c 插入到为 Y 中第%d 位置\n", output_str[j], j);
	}
	else
	{
		dfs(i - 1, j);
		printf("将 X 中第%d 位置的字符%c 删除\n", i, input_str[i]);
	}
}

void output()
{
	int i, j;
	for (i = 0; i <= len_input_str; i++)
	{
		for (j = 0; j <= len_output_str; j++)
		{
			printf("%d ", c[i][j]);
		}
		printf("\n");
	}
	for (i = 0; i <= len_input_str; i++)
	{
		for (j = 0; j <= len_output_str; j++)
		{
			printf("%d ", b[i][j]);
		}
```

```
        printf("\n");
    }
    printf("最少的操作次数为：%d\n", c[len_input_str][len_output_str]);

    dfs(len_input_str, len_output_str);
}

int main()
{
    input();

    solve();

    output();

    return 0;
}
```