算法设计与分析（第三章习题）

3.1 设计一个$O(n^2)$时间的算法，找出由n个数组长的序列的最长单调递增子序列。

分析：

问题的输入：由n个数组成的序列；

问题的输出：序列中最长单调递增子序列；

问题的规模：序列长度n。

解法一：转化为求最长公共子序列

（求最长递增子序列问题 转化为 求最长公共子序列的问题）

将序列a 按非递减顺序排列，形成新序列b，问题就转变成求解a 和b的最长公共子序列。假设使用快速排序，则排序过程的时间复杂度为$O(n\log n)$， 而求两个序列的最长公共子序列的时间复杂度为$O(n^2)$（a,b长度相等，都为n），该解法的时间复杂度为$O(n^2)$。

解法二：

用数组$b[0:n-1]$记录以$a[i]$ $i\in[0,n-1]$为结尾元素的最长单调递增子序列的长度。（先不求解最长的单调递增子序列，而是先求其长度）

$b[i]$表示当以$a[i]$为单调递增子序列最后一个元素时，所得最长单调递增子序列的长度。然后找出数组b中的最大元素，就是序列a的最长单调递增子序列。我们可以得到递推式：

$$b[i]=\begin{cases}1 & i=0\\ \max\limits_{0\le k<i}(b[k]+1) & a[k]\le a[i]\end{cases}$$

降规模的体现：若找到某一个元素是单调递增子序列中的结尾元素，则其后的元素便不再考虑。

最优子结构验证：$b[i]$表示当以$a[i]$为单调递增子序列最后一个元素时，所得最长单调递增子序列的长度。很明显，满足最优子结构性质。

复杂度分析：遍历n个元素，计算b值；遍历n次，找出b的最大值。

算法复杂度$O(n^2)$。

3.2 复杂度为$O(n\log n)$

分析：

假设序列a，其最长单调递增子序列的长度为m

考虑其长度为i的单调子列$(1<=i<=m)$，这样的子列可能有多个，选取这些子列的结尾元素的最小值， 用L_i表示。易知：$L_1<=L_2<=L_3<=...<=L_m$。

现在，需要寻找序列a 对应的L 序列，如果找到的最大的L_i是L_m，那么m就是最大单调子列的长度。

从左至右扫描a，对于每一个a_i

1）$a_i<L_1$,则$L_1=a_i$

2）$a_i>L_k$,则$L_k+1=a_i, k=k+1$(其中 k 是当前见到的最大的 L 下标)

3）$L_s<=a_i<L_s+1$,则$L_s+1=a_i$

扫描完成后，就得到了最长递增子序列的长度。

从上述方法可知，对于每一个元素，我们需要对L 进行查找操作，由于L是有序的，使用二分查找，复杂度为$\log n$，于是总的复杂度为$O(n\log n)$。

详细的博客：

https://www.felix021.com/blog/read.php?1587

3.1 任务调度问题

输入：需要处理的作业总数， A,B 处理第 i 个作业需要的时间；

输出：完成 n 个作业的最短时间；
规模参数：作业量
问题分析：
当完成了 k 个作业，设机器 A 共花费了时间 x ,机器 B 花费时间的最小值是 x 的一个函数。

设置 $T[x][k]$ 是机器 B 花费时间的最小值。

$$T[x][k]=\begin{cases}\sum_{i=1}^{k}b[i] & x=0\\ \min\{T[x-a[k]][k-1],T[x][k-1]+b[k]\} & x\geq 0\end{cases}$$

$T[0][k]=\sum_{i=1}^{k}b[i]$：表示的是若机器 A 花费的时间是 0，即 A 没有参与 k 个作业的处理， k 个作业全部是交给 B 处理。此时 B 机器花费的最小时间就是 B 独立完成 k 个作业的时间和。

$T[x-a[k][k-1]]$：表示若已经完成了 $k-1$ 个作业，其中机器 A 花费时间是 $x-a[k]$，机器 B 花费的最小时间是 $T[x-a[k][k-1]]$。

若第 k 个作业交给 A 做，则完成 k 个任务的时候， A 花费的时间为 $x-a[k]+a[k]=x$。

$T[x][k-1]+b[k]$： 表示的是若已经完成了 $k-1$ 个作业，其中机器 A 花费时间小于等于 x，机器 B 花费的最小时间是 $T[x][k-1]+b[k]$。

若第 k 个作业交给 B 做，则完成 k 个任务的时候，处理完 $k-1$ 个任务 B 花费的最少时间+第 k 个任务在 B 上处理的时间，即 $T[x][k-1]+b[k]$。

综上： 完成 n 个作业的总时间是 $\max(x,T[x][k])$

验证最优子结构：因为计算时间是加法计算，很明显，子问题的最优解也是问题的最优解，满足最优子结构。

复杂度分析： $O(n*\min\{sum(ai),sum(bi)\})$

3.5 乘法表

	a	b	c
a	b	b	a
b	c	b	a
c	a	c	c

定义一个三维数组 $f[n][n][3]$，n 为输入字符串的长度；

$f[i][j][0]$：字符串中第 i 个元素到第 j 个元素的子串表达式值为 a 的加括号方式数

$f[i][j][1]$：字符串中第 i 个元素到第 j 个元素的子串表达式值为 b 的加括号方式数

$f[i][j][2]$：字符串中第 i 个元素到第 j 个元素的子串表达式值为 c 的加括号方式数

由乘法表的定义可知：

$$f[i][j][0]=\sum_{k=i}^{j-1} f[i][k][0]*f[i][k+1][2]+f[i][k][1]*f[i][k+1][2]+f[i][k][2]*f[i][k+1][1]$$

$$f[i][j][1]=\sum_{k=i}^{j-1} f[i][k][1]*f[i][k+1][0]+f[i][k][1]*f[i][k+1][1]+f[i][k][0]*f[i][k+1][0]$$

$$f[i][j][2]=\sum_{k=i}^{j-1} f[i][k][1]*f[i][k+1][0]+f[i][k][2]*f[i][k+1][1]+f[i][k][2]*f[i][k+1][2]$$

算法复杂度：$O(n^3)$

电子版作业

编辑距离问题

一、待解决问题：

问题描述：设 A 和 B 是 2 个字符串。要用最少的字符操作将字符串 A 转换为字符串 B 。这里所说的字符操作包括:

(1)删除一个字符;

(2)插入一个字符；

(3)将一个字符改为另一个字符。

将字符串 A 变换为字符串 B 所用的最少字符操作数称为字符串 A 到 B 的编辑距离,记为 $d(A,B)$ 。试设计一个有效算法,对任给的 2 个字符串 A 和 B ,计算出它们的编辑距离 $d(A,B)$ 。

数据输入：第 1 行是字符串 A ,第 2 行是字符串 B 。

结果输出：字符串 A 和 B 的编辑距离 $d(A,B)$ ，将找到的最大间隙输出到文件 output.txt 中。

二、题目及算法分析

题目输入：两串字符串 A,B 题目输出：字符串 A 和 B 的编辑距离 $d(A,B)$

规模参数：由于本题主要是计算两个字符串之间的编辑距离，这与字符串的长度是息息相关的，易知道规模参数便是源字符串 A 和目标字符串 B 的长度，此处用 m,n 来表示。

目标：字符串 A 和 B 的编辑距离 $d(A,B)$

规模下降思路

本题主要用动态规划来求解，因此，将规模是首先要考虑的

字符串 A ： $a_1,a_2,a_3...a_m$

字符串 B ： $b_1,b_2,b_3...b_n$

$d[i][j]$ 用于记录字符串 A_i 到字符串 B_j 的最短编辑距离，现在任务就是要将 $a_1,a_2,a_3...a_i$ 转换为 $b_1,b_2,b_3...b_j$ ，具体情况如下：

1. 若 $a_i=b_j$

$$d[i][j]=d[i-1][j-1]$$

上式表示的是如果 A 、 B 两个字符串末尾字符相等，则编辑距离就等于将 $a_1,a_2,a_3...a_{i\text{-}1}$ 转换为 $b_1,b_2,b_3...b_{j\text{-}1}$ 的编辑距离。

2. 若 $a_i!=b_j$ ，有如下三种情况进行变换

1）直接将 a_i 变为 b_j

$$d[i][j]=d[i-1][j-1]+1$$

上式表示的是将 A_i 到 B_j 的编辑距离转变为从 A_{i-1} 到 B_{j-1} 的编辑距离，然后再加上直接将 a_i 变为 b_j 的编辑操作。

2）删去 a_i

$$d[i][j]=d[i][j-1]+1$$

上式表示的是将 A_i 到 B_j 的编辑距离转变为从 A_{i-1} 到 B_j 的编辑距离，然后再加上直接将 a_i 删去的编辑操作。

3）增加一个等于 b_j 的 ch（$ch==b_j$），即插入字符

$$d[i][j]=d[i][j-1]+1$$

上式表示的是将 A_i 到 B_j 的编辑距离转变为从 A_i 到 B_{j-1} 的编辑距离，然后再加上直接将增加某 a_i 的编辑操作。

最优子结构验证

规模下降的思路可以看出，本算法采用的是规模降 1 的思路，而每个降规模后的子问题的最优解很明显都是导致原问题有最优解的前提，即问题的最优解所包含的子问题的解也是最优的，因此满足最优子结构。

递归方程的推导

若某个字符串为空，为编辑距离就是另外一个字符串的长度，即直接增加就可以实现，因此递归方程如下：

A 为空，$i=0;d[0][j]=j;$（边界条件）

B 为空，$j=0;\ d[i][0]=i;$（边界条件）

AB 均不为空，存在两个字符相同时，$a_i==b_j$

$$d[i][j]=\min\{d[i][j]=d[i-1][j]+1,d[i][j]=d[i][j-1]+1,d[i][j]=d[i-1][j-1]\}$$

AB 均不为空，两个字符不相同时，$a_i!=b_j$

$$d[i][j]=\min\{d[i][j]=d[i-1][j-1]+1,d[i][j]=d[i-1][j]+1,d[i][j]=d[i][j-1]+1\}$$

算法复杂度分析

需要对两个字符串进行遍历，因此会有双重循环，故算法复杂度是 $O(m*n)$，其中 m 和 n 分为是字符串 A,B 的长度。