**软工 算法复习笔记**

**各算法思想的描述**

**1、分治法**

将一个规模为n的问题分解为K个规模较小的子问题，这些子问题互相独立且与原问题相同。递归的解决这些问题，然后将各个子问题的解合并得到原问题的解

**2、贪心法**

当前的选择可能要依赖于已经做出的选择，但不依赖于有待于做出的选择和子问题。因此贪心法是自顶向下，一步一步地做出贪心的选择

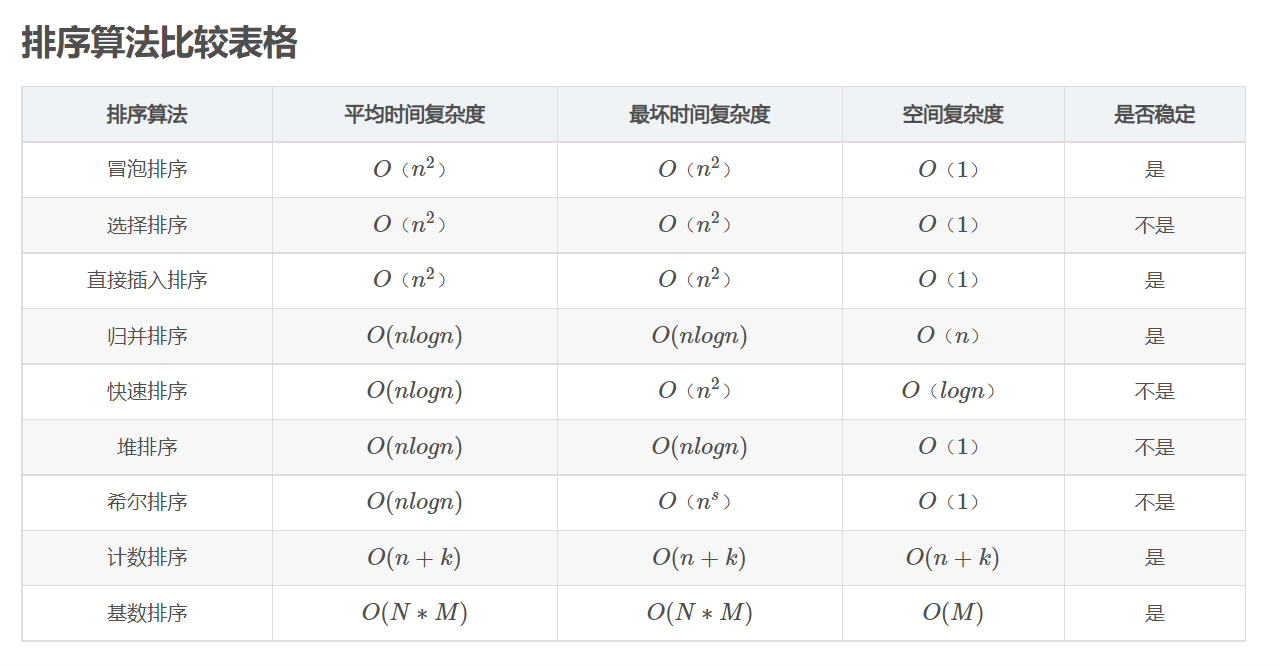
**3、动态规划**

动态规划的实质是分治思想和解决冗余，因此动态规划是一种将问题实例分析为更小的、相似的子问题，并存储子问题的解而避免计算重复的子问题，以解决最优化问题的算法策略

**4、回溯法**

回溯法是一种选优搜索法，按选优条件向前搜索，以达目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法

**排序算法比较**





**稳定排序，**是指在排序算法中，相同值的两个元素，在输入数组中先出现的数在输出数组中也先出现。像冒泡排序，插入排序，基数排序，归并排序等都是稳定排序。

**原地(原址、就地)排序，**是指：基本上不需要额外辅助的的空间，允许少量额外的辅助变量进行的排序。就是在原来的排序数组中比较和交换的排序。像选择排序，插入排序，希尔排序，快速排序，堆排序等都会有一项比较且交换操作(swap(i,j))的逻辑在其中，因此他们都是属于原地(原址、就地)排序，而合并排序，计数排序，基数排序等不是原地排序。



***[插入排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/63252865" \t "http://www.javashuo.com/article/_blank)***

插入排序是一种较为简单的排序算法，它的基本思想是经过构建有序序列，对于未排序数据，在已排序序列中从后向前扫描，找到相应位置并插入。  
形象的能够理解为打扑克抓拍的过程，一般咱们右手抓牌，没抓一张牌，就放到左手，抓下一张牌后，会把这张牌依次与左手上的牌比较，并把它插入到一个合适的位置（按牌面大小）。

***[希尔排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/63683571" \t "http://www.javashuo.com/article/_blank)***

希尔排序是对直接插入排序的一种优化，实质就是把直接插入排序改成了分组插入排序。其基本思想就是将整个待排序元素序列按gap（步长）分割为N个组，对每一个组进行直接插入排序，而后在减少gap（步长）再进行直接插入排序，直到gap达到最小时，即数组基本达到有序时，再对数组进行直接插入排序，此时直接插入排序就能够达到最高效率。

***[选择排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/63688282" \t "http://www.javashuo.com/article/_blank)***

选择排序的排序思想就和它的名字同样，每次经过从无序的数组中选择出一个最小的（要求升序排列）数把他放到数组的最前面。再依次找次小的数字放到数组无序区的最前。直到数组为有序

***[堆排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/64180298" \t "http://www.javashuo.com/article/_blank)***

堆排序（使用大堆，升序）从基本实现原理来讲也是一种选择排序，它一样是肯定了位置选择符合位置的元素，可是堆排序是更加优化的选择排序的版本，它利用了堆的特性。父结点的值大于子结点，且知足彻底二叉树，大大提升了选择排序的效率。

***[冒泡排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/64500492" \t "http://www.javashuo.com/article/_blank)***

冒泡排序（这里指升序）是一种很是简单直观的排序方式，它是一种交换式的排序方法，基本思想就是相近的两个数字做比较，小的放到前面，大的放后面，按照这个规则从头向后比较，最大的数就被换到了数组尾。

***[快速排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/64923749" \t "http://www.javashuo.com/article/_blank)***

快速排序是一种在实际应用中常常用到的排序算法，它的应用场景是大规模的数据排序，而且实际性能要好于归并排序。它的基本原理是从数组中选取一个元素，把全部大于这个元素的数都放到它的后面，全部小于这个元素的数都放到它的前面，而后这个元素就把原数组切分红了两个部分，再分别对这个两个部分进行一样的操做，直到数组不能再切分的时候，此时数组为有序。化

***[归并排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/65451607" \t "http://www.javashuo.com/article/_blank)***

“归并”的含义是将两个或两个以上的有序表组合成一个新的有序表，归并排序和快排同样也采用的是分治的思想，它的基本原理是经过对若干个有序结点序列的合并为一个有序序列来实现排序的。net

***[基数排序](http://www.javashuo.com/link?url=http://blog.csdn.net/mbuger/article/details/67640398" \t "http://www.javashuo.com/article/_blank)***

基数排序（升序）是一种非比较式的排序方式，和以前博文中提到的快排，冒泡排序，插入排序这些排序算法不同，它没有使用任何交换的方式，它的基本思想是经过分配的方法把元素从小到大分配，以到达排序的做用。

**分治策略**

1. 分解，即将一个问题划分为一些子问题
2. 解决，当子问题规模足够小，则停止递归，直接求解
3. 合并，将子问题的解合成原问题的解

**《算法导论》**把分治法的范式描述为以下3步：

1、Divide the problem into a number of subproblems that are smaller instances of the same problem.

2、Conquer the subproblems by solving them recursively. If the subproblem sizes are small enough, however, just solve the subproblems in a straightforward manner.

3、Combine the solutions to the subproblems into the solution for the original problem.

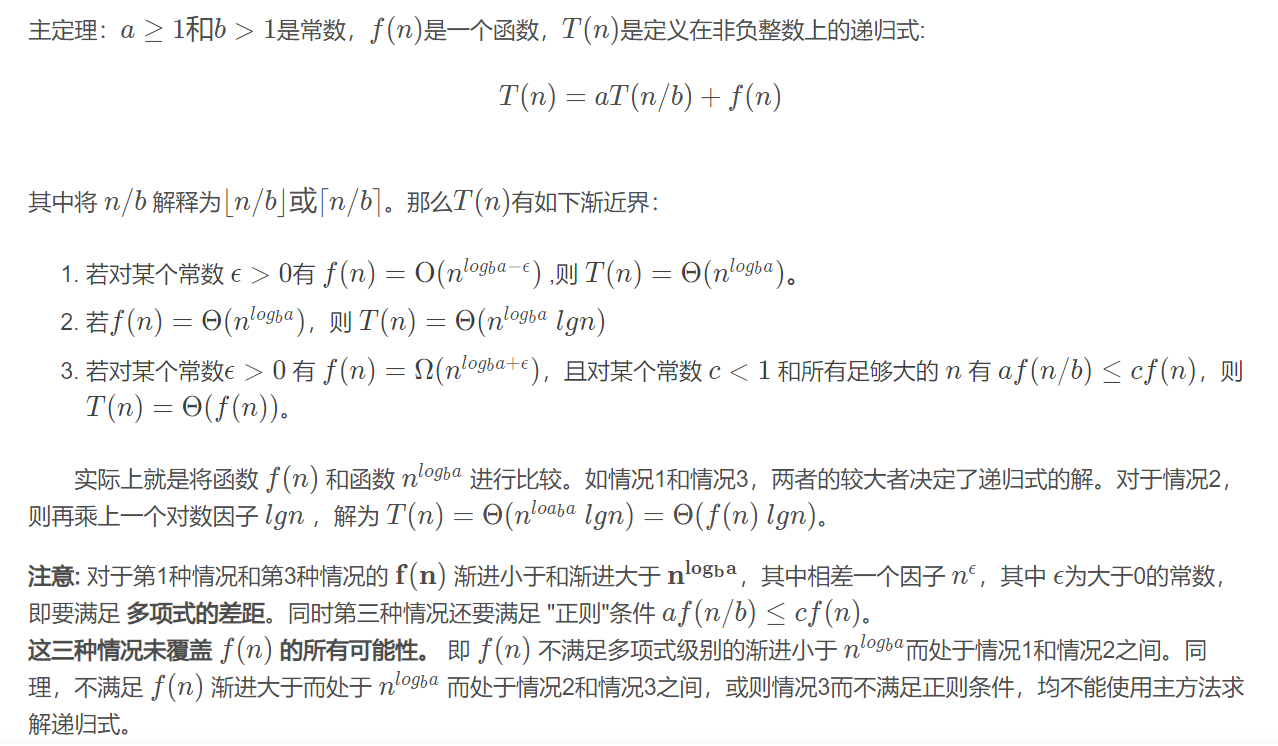
**归并排序**就是一个用分治法的经典例子，这里我用它来举例描述一下上面的步骤：

1、归并排序首先把原问题拆分成2个规模更小的子问题。

2、递归地求解子问题，当子问题规模足够小时，可以一下子解决它。在这个例子中就是，当数组中的元素只有1个时，自然就有序了。

3、最后，把子问题的解（已排好序的子数组）合并成原问题的解。

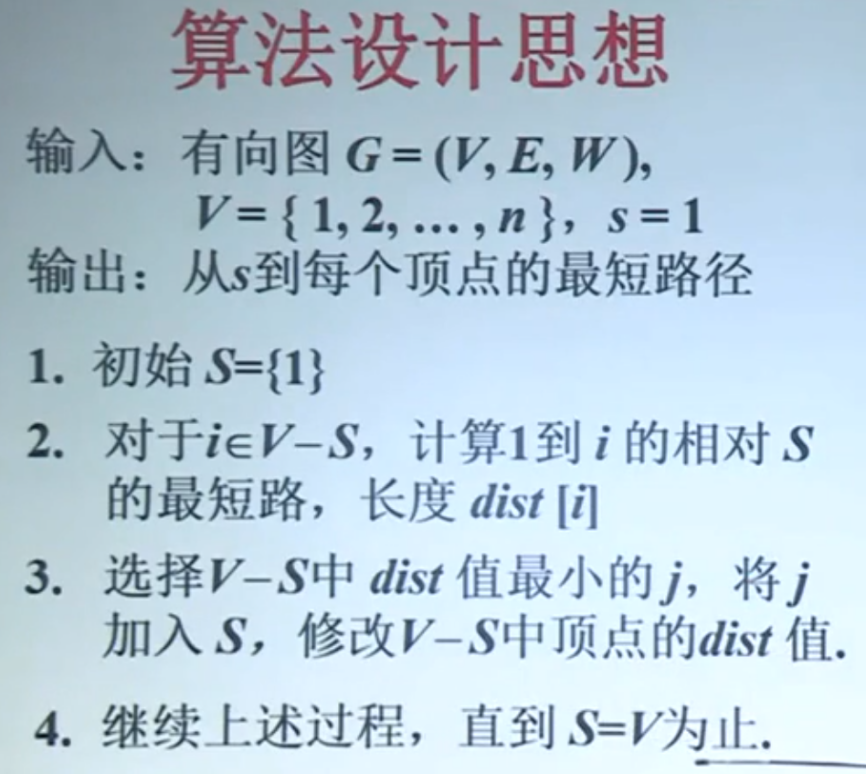
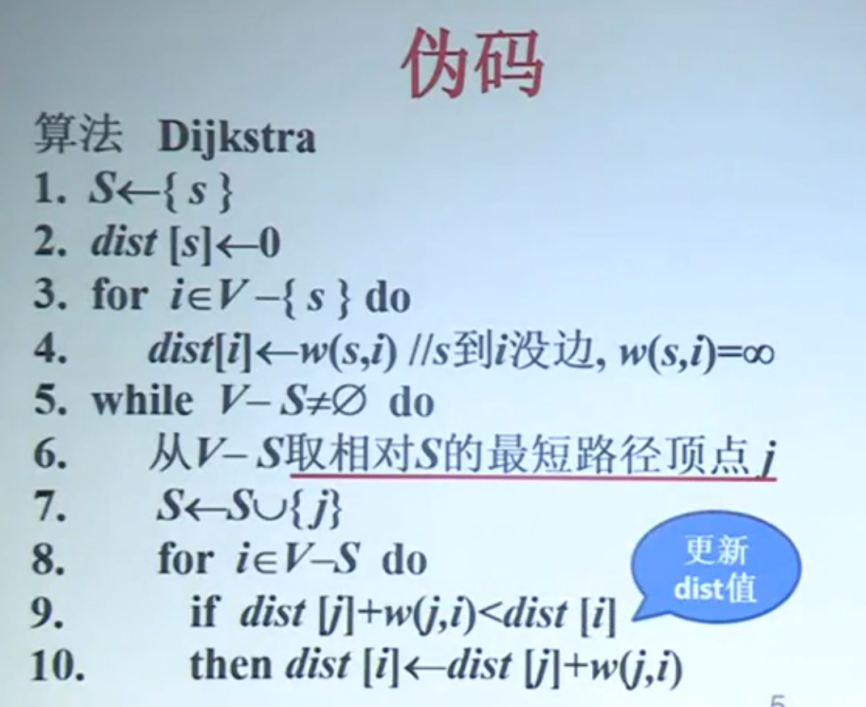
**主定理与主方法**



**贪心算法思想**

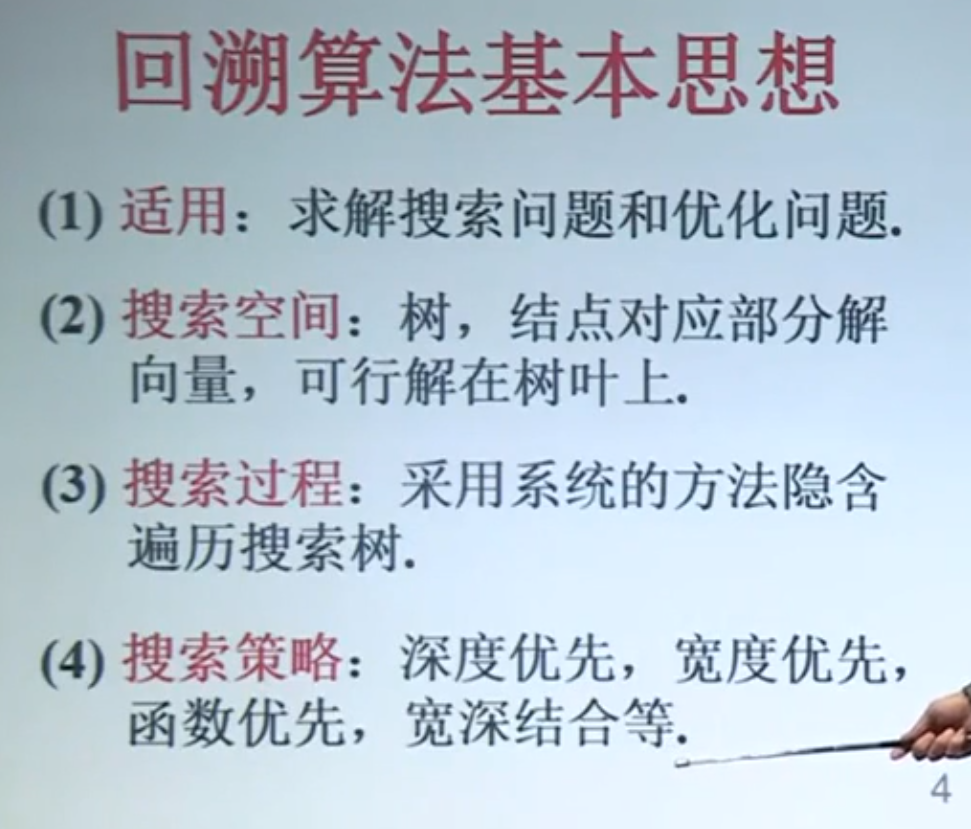
1. 贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。
2. 贪心选择是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素。
3. 当一个问题的最优解包含其子问题的最优解时，称此问题具有最优子结构性质。运用贪心策略在每一次转化时都取得了最优解。问题的最优子结构性质是该问题可用贪心算法求解的关键特征。贪心算法的每一次操作都对结果产生直接影响。贪心算法对每个子问题的解决方案都做出选择，不能回退。
4. 贪心算法的基本思路是从问题的某一个初始解出发一步一步地进行，根据某个优化测度，每一步都要确保能获得局部最优解。每一步只考虑一个数据，他的选取应该满足局部优化的条件。若下一个数据和部分最优解连在一起不再是可行解时，就不把该数据添加到部分解中，直到把所有数据枚举完，或者不能再添加算法停止。

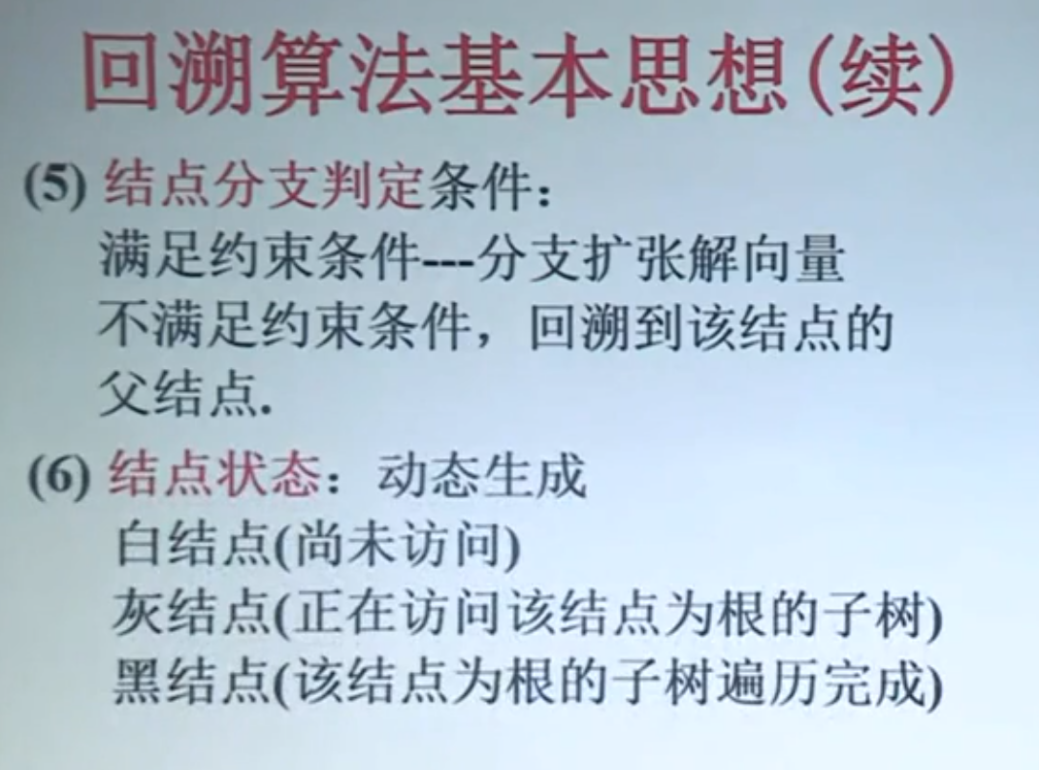
**迪杰斯特拉算法（贪心算法）**



1. 初始化：设点集S={s}，s表示源点；
2. 选择：找出相对于S（只经过S中节点）的所有路径中长度最短的一条，设这条路径抵达的点i∉S，则将i加入S中；
3. 更新：对新的S集合，重新计算到各个顶点的相对最短值；
4. 迭代：重复（2）（3）步骤，直到S=V

**回溯算法思想**





**动态规划算法思想**

**动态规划4个组成部分：**

**1.确定状态**

解动态规划时候通常需要开一个数组，数组的每个元素f[i]或者f[i][j]代表什么

类似于解数学题中，x，y, z代表什么

确定状态需要两个意识

– 最后一步（最优策略中使用的最后一枚硬币ak）

– 化成子问题（最少的硬币拼出更小的面值27-ak）

**2.转移方程**

设状态f[x]=最少用多少枚硬币拼出x

对于任意x，f[x] = min{f[x-2]+1 , f[x-5]+1, f[x-7]+1}

**3.初始条件和边界情况**

f[x] =min{f[x-2]+1, f[x-5]+1, f[x-7]+1}

两个问题：x-2, x-5或者x-7小于0怎么办？什么时候停下来？

初始条件：f[0]=0

如果不能拼出Y，就定义f[Y]=正无穷

– 如f[-1] = f[-2] = … =正无穷

所以f[1]=min{f[-1]+1, f[-4]+1, f[-6]+1} = 正无穷，表示拼不出来1

**4.计算顺序**

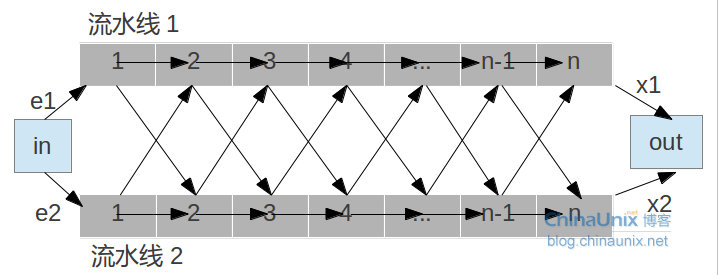
大多数动态规划问题一维计算都是从小到大，二维都是从左到右，从上到下计算，但也有一些动态规划问题计算顺序正好相反

判断原则：当我们需要计算右边状态时，上一个左边的状态是否已经计算完了

递归算法的特点：

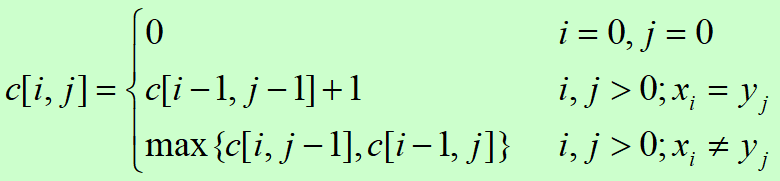
消除冗余，加速计算

与递归算法相比，没有任何重复计算

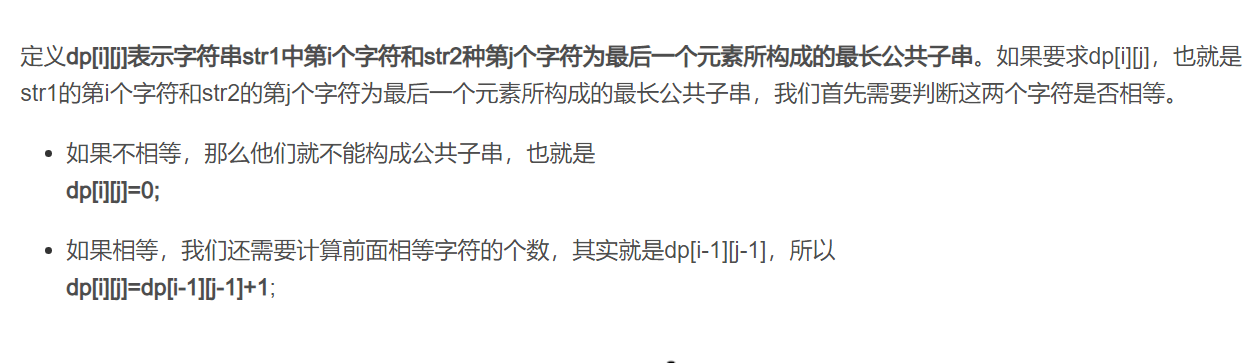


**几个典型算法的状态转移方程**

**最长公共子序列**



**最长公共子串**



**Johnson算法**

