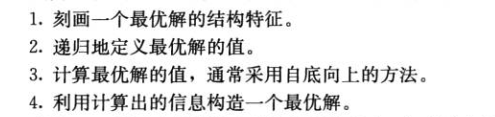
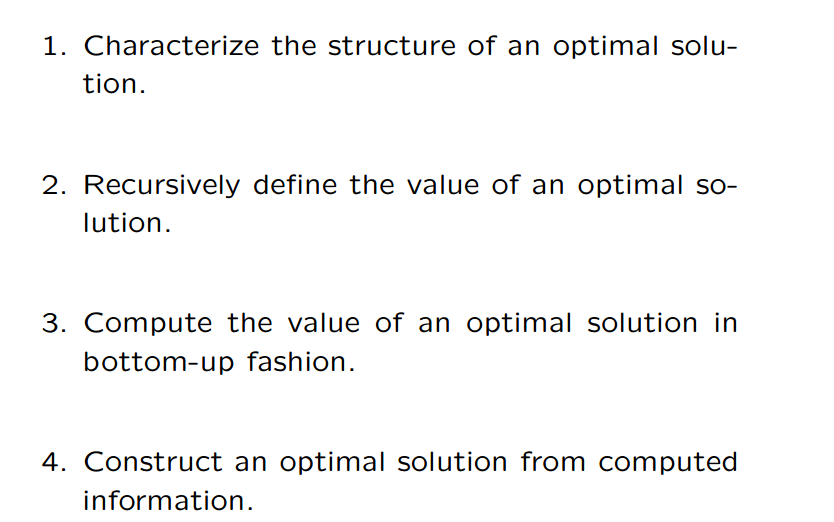
dynamic programming，有的时候在recursion中会有重复计算，dynamic programming的核心思路就是当subproblem并非相互independent，或者说share sub-sub problems的时候，dynamic programing只解决sub-sub problem一次，然后把答案存储在表或array里

dynamic programming通常都是用在 optimization problems 优化问题里

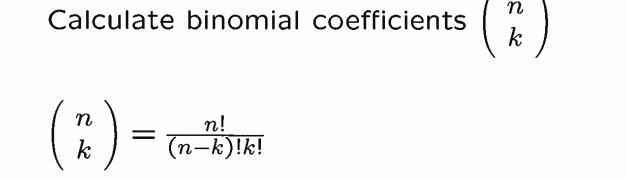
换句话来说find the optimal value of a solution where several solutions exists(最大值最小值)

Dynamic programming algorithm通常有四步



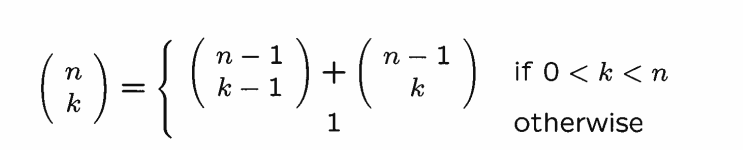


例子，计算binomial coefficient



计算感叹号会变得非常大

我们可以用recursive definition

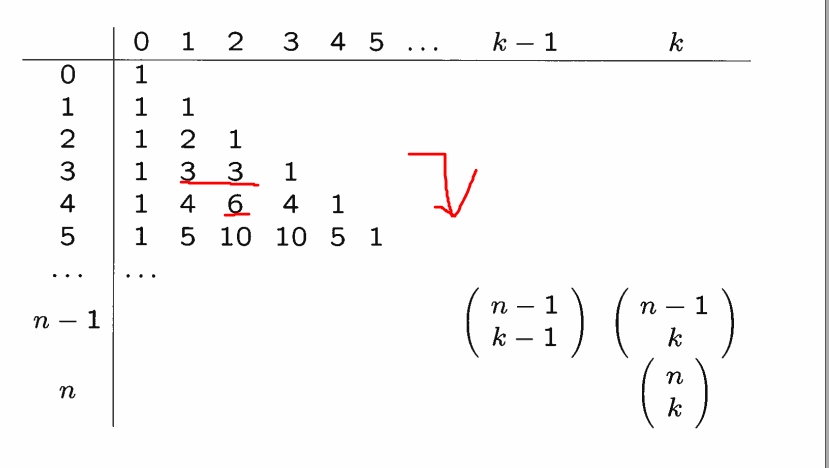


他把一步到位的东西分成了好几步recursive

但是现在的问题在于会重复计算，

例如(8 7)=(7 6)+（7 5）….一直往下会大量重复

不如列一张表

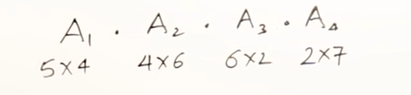


记录下来值

把这张表完全算出来也只要nk时间，因为计算量不会特别大，就是单纯的加法，（加上面两个）

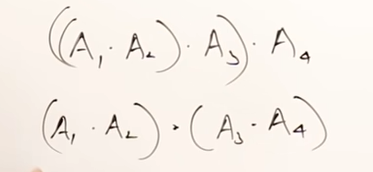
space commplexisty是O（k），计算的时候我们并不用存储整张表，保证两行也就是上面一行存在就行了

15.2 matrix-chain multiplication

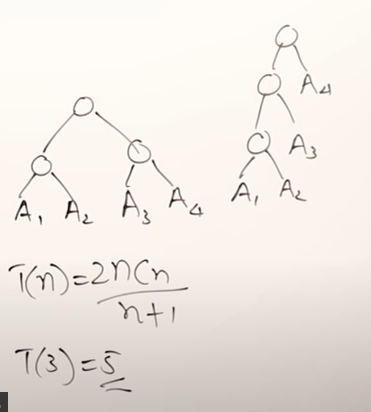


矩阵连乘

在矩阵连乘中，交换律是不行的，但先乘哪个却不影响



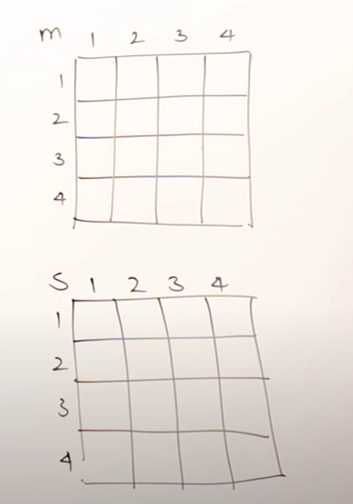
我们的目的就是知道，按怎样的连乘顺序，效率最高

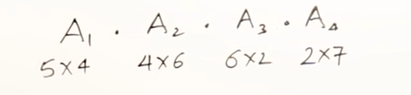
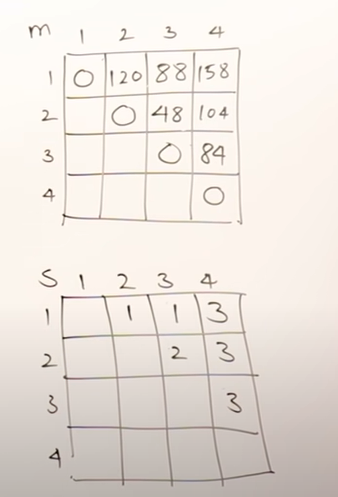
，看这个树，每个都要进行三次计算，dynamic programming目的就是从每个可能的解中取出最优解，我们需要每一个都建一个树吗？

T（n）=2nCn/n+1

STEP1

创造两个正方形表格，长宽等于有几个连乘





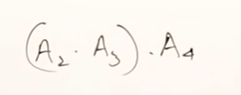
S表统一算法

他们在s表中的算法

最终结果必然是矩阵A×矩阵B //可能并不是原来的A1，A2，A3..，而是乘出来的东西

我们选矩阵A

如果矩阵A是原始矩阵，例如A1，那么对应的位置就写1

如果矩阵A是复合矩阵，例如,是A2\*A3,那么就取最后一个

第一层

m11,22,33,44这条斜对角全部都是零，他们代表A1，A2，A3，A4.也就是不用处理，

第二层

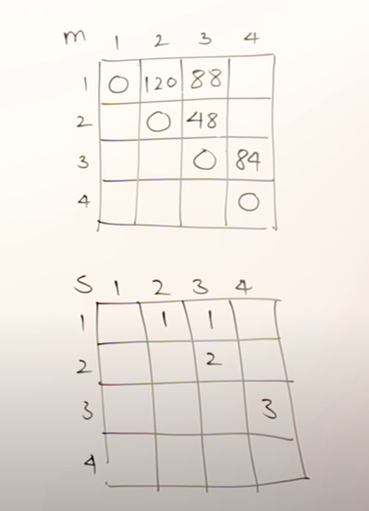
A（1，2） (2，3) (3，4)也就是第二层 ，也就是如果先连乘这些，需要多少时间，这个时间判断的计算方式如下

(1,2)也就是a1\*a2时，如果他们的下标行列数字越大，矩阵计算起来越麻烦5\*4\*6=120也就是我们认为的A1\*A2所需要时间

23同理4\*6\*2=48

34同理6\*2\*7=84

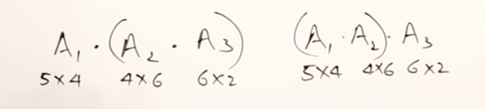
第二层S表，这里A\*B都是最简单的形式，因此都取第一个数，分别是1,2,3



第三层

这层开始要选择了

例如m(1,3)

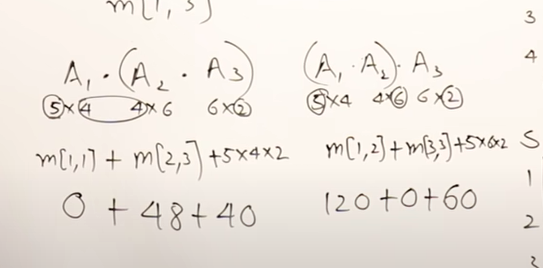


它有两种排序可能

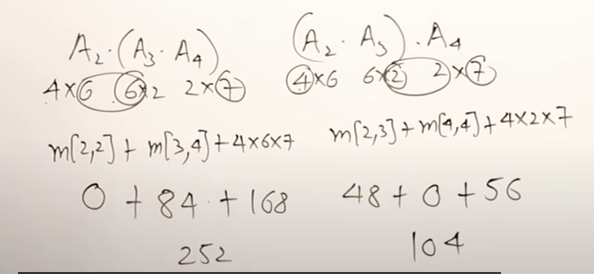
我们要分别计算

第一个：4\*6\*2得到A2A3 combo， 然后5\*4\*2得到整体combo,相加等于=88

同理第二个等于180



第一个更小，我们选第一个，A是简单形式所以S(1，3)=1

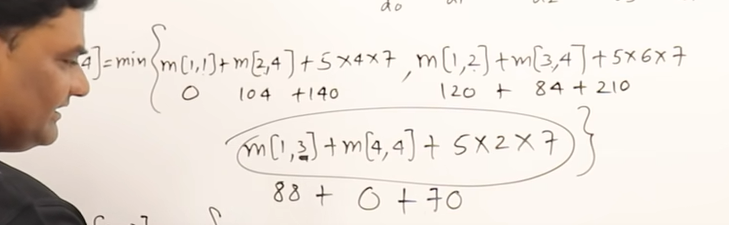


2,4 第二个更小，A是复合形式，因此S(2，4)=3

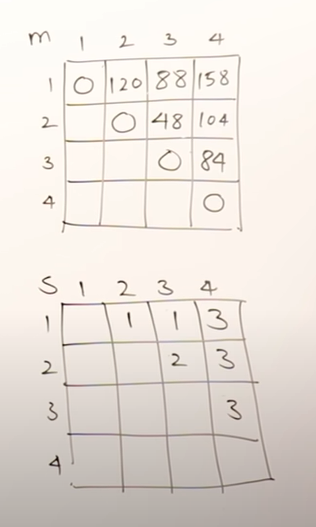
第四层

也就是最后一层

三中排列组合形式

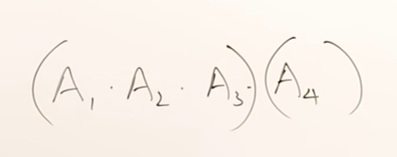


最后一种也就是A123连乘，再A4最小，3是最后的，因此是3

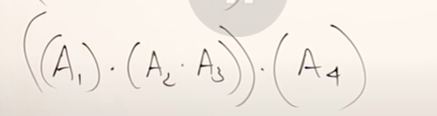


然后具体顺序先看第一行

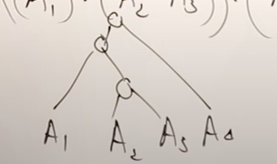
我们剩下1234，也就是说我们需要14最优，3，代表到3结尾



然后我们剩下123，也就是要检查第一行第三列，是1，也就是1最优

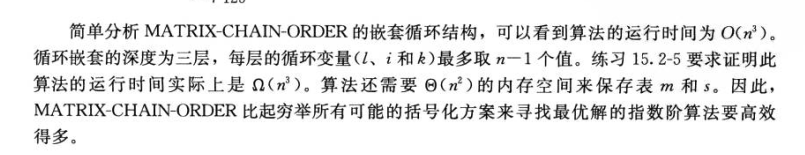


然后剩下23，看第二行第三列，数字是2，剩下的项目是2，不用继续切割，结束

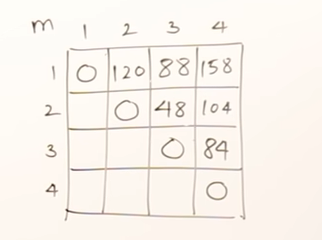


建立我们的树

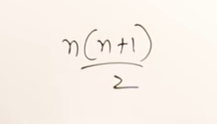
复杂度是n^3



因为我们要准备一半的表

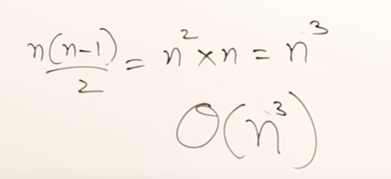


也就是1+2+3+….n的等差数列，

是n^2

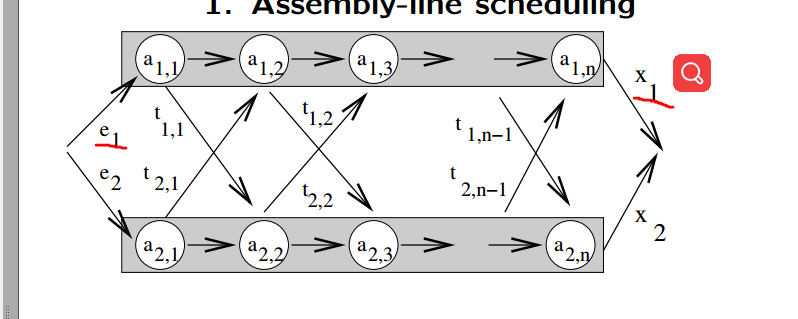
但计算每个值我们需要计算可能性，然后寻找最小值

需要n



1.Assembly-line scheduling

就是最简单的一道动态规划



e1 x1指的是进入的时间

a指的是在站点停留的的时间

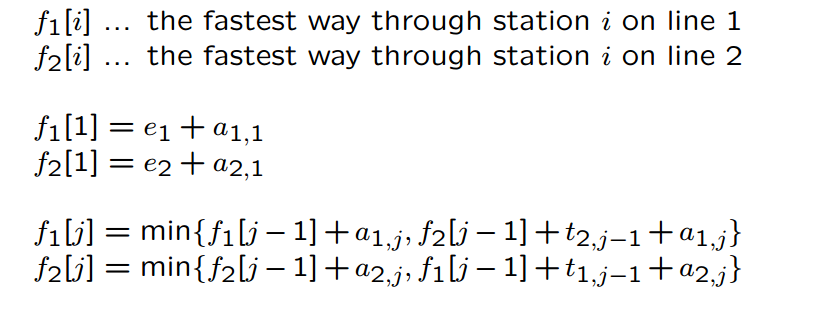
我希望通过合理的路线，在最短时间内通过

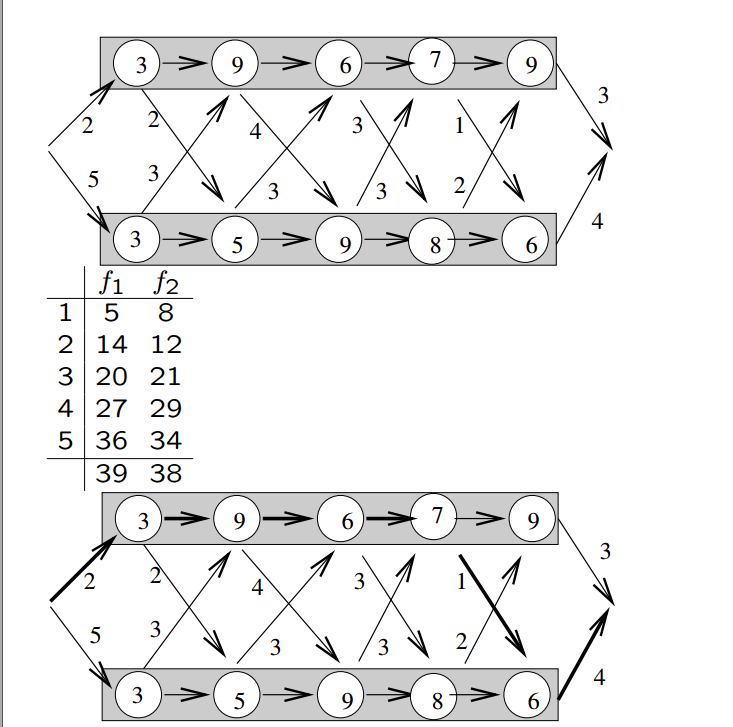
注意，我们不能够通过比较t11与a11来决定是走哪条路

假设a11较小，但是a12很大，我们就必须要走a12，就得不偿失了

但是我们可以用动态规划，我们要知道在1以后的点无论是哪条线，都是基于前一个点的，

而前一个点，必定有一个最短路径，不管是怎么到达的，前方这个点的到达方式对后面不影响，因此我们可以从第一个点开始一步一步构建



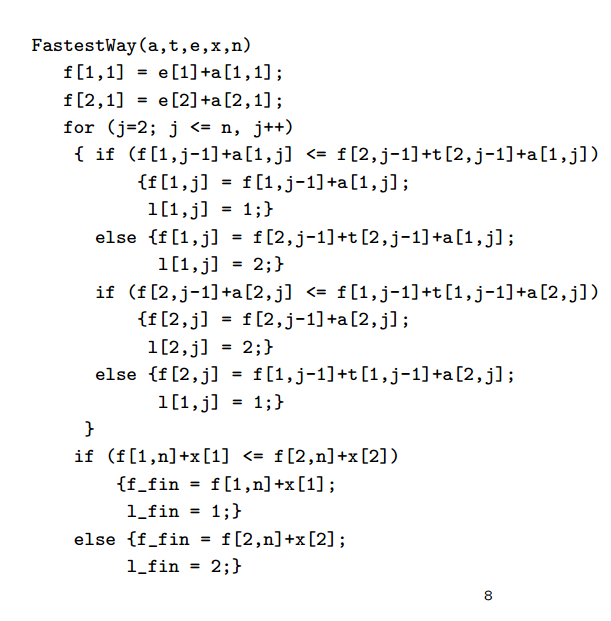


f1是2+3=5，f2是5+3=8

然后基于此开始构建，到f1,2要么是5+9，要么是8+3+9，答案是14

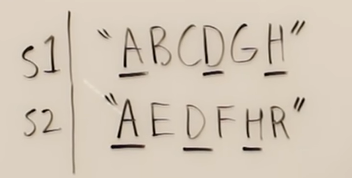
同理f2,2要么是5+2+5，要么是8+5，因此是12

………

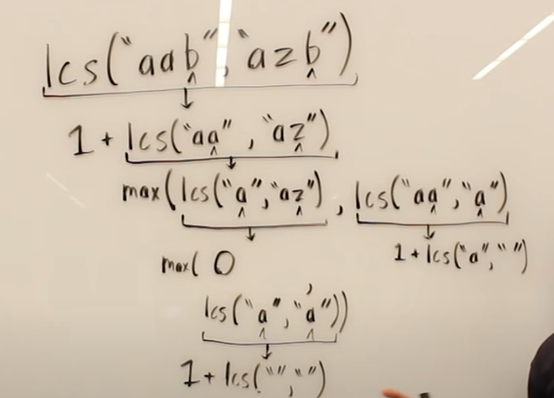


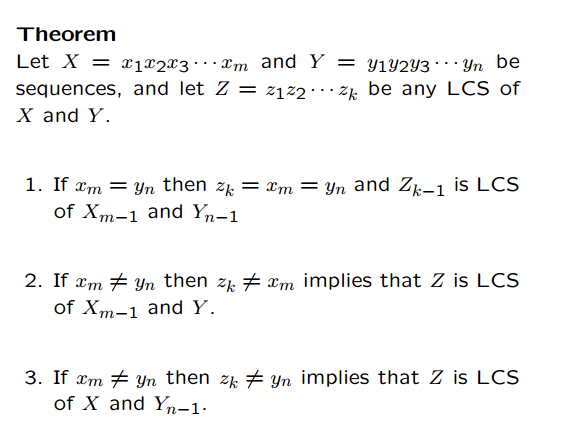
2.Longest common subsequence

首先，Longest common subsequence并不追求一定要是continuous的



我们希望算法lcs能够return最长的subsequence





三个定理，

1.如果最后一项相等，那么我们同时去掉最后一项，剩下的LCS Zn-1仍然是LCS

2.如果不相等，且LCS Z的最后一项不等于Xm，可以去掉Xm //因为Xm对结果没影响

3.如果不相等， ……………………………yn,可以去掉Yn

这三个定理就围绕着一个点来说，就是如果全相等，那么可以同时去掉最后一项，如果不相等，看可以去掉没影响的一项，也就是说Zn是包含了无数子问题recursion的，

思路：我们可以从后往前比较，一样的就都减去，同时LCS长度加一，

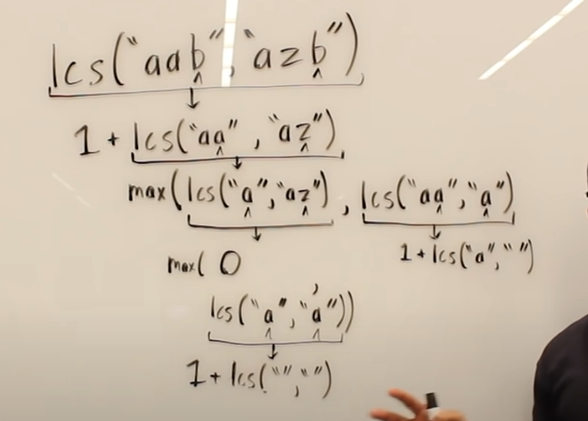
如果不一样，我们就分别讨论两种状况，也就是去掉X和去掉Y的，最后recursion取其max

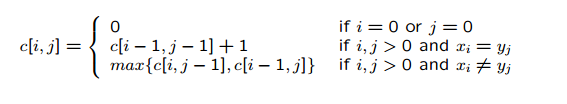
//这个思路实际上是因为我们不知道什么时候从前面开始，然而不管什么时候开始，若想取到最大，那么最后必然是一样的

例如

ABCDEFG DEACFG

不管我们认为是DE匹配，还是AC匹配，FG是必然存在最优解的

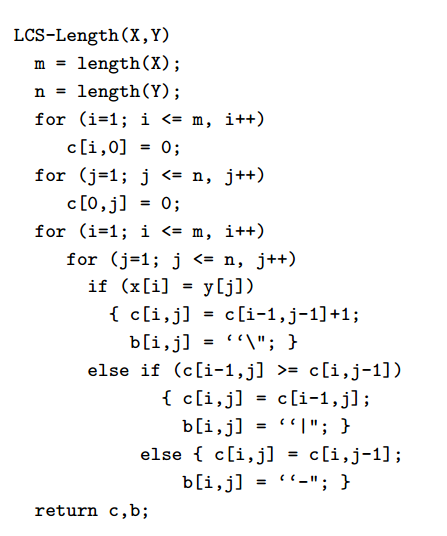




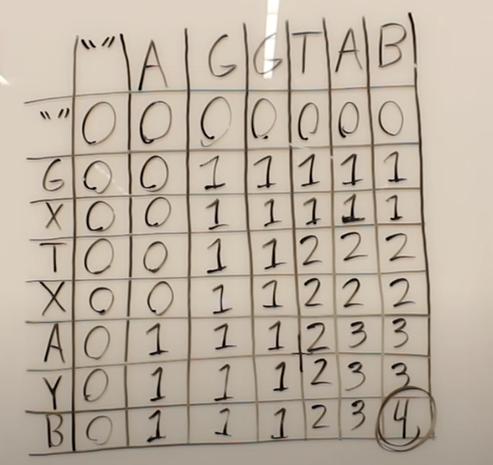
如果某个字符已经只剩Null string了，也就是base case, return 0

如果两者相等，运用第一定律，

如果不相等，recursion成max，运用第二第三定律



表格法，注意，表格法与上面的内在思路是完全一样的

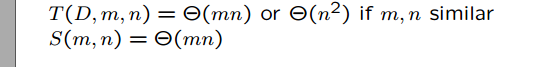


STEP1，第一行与第一列全部填0，因为这是理论1，null string和所有String common都是0

STEP2，从上到下一行一行填，从左到右，就像正常写文章的顺序

行与列不相同时，往上看一格或者往左看一格，取最大值，//对应定理23， 如果不一样去掉一个，取最大的

行与列相同时，在左上角一格的基础上加一//对应定理1，相同时，加一



Time complexity与Space complexity， m与n指的是两个String的长度

4.LIS问题

很直接