从递推到动归（下）

一、数字三角形

惊人的发现

$f(i,j)$ 代表从底边走到 i, j 点的最大值

$f(i,j)$ 代表从顶点走到 i, j 点的最大值

1. 数学符号完全一致
2. 语义信息不同
3. 递归公式不同
4. 结论：数学符号无法完全代表状态定义

两种方法的对比

本质：两种状态定义方式的对比

1. 第一种：不用做边界判断，最终结果，直接存储在 $f[0][0]$
2. 第二种：需要做边界判断，最终结果，存储在一组数据中
3. 结论：第一种要比第二种优秀

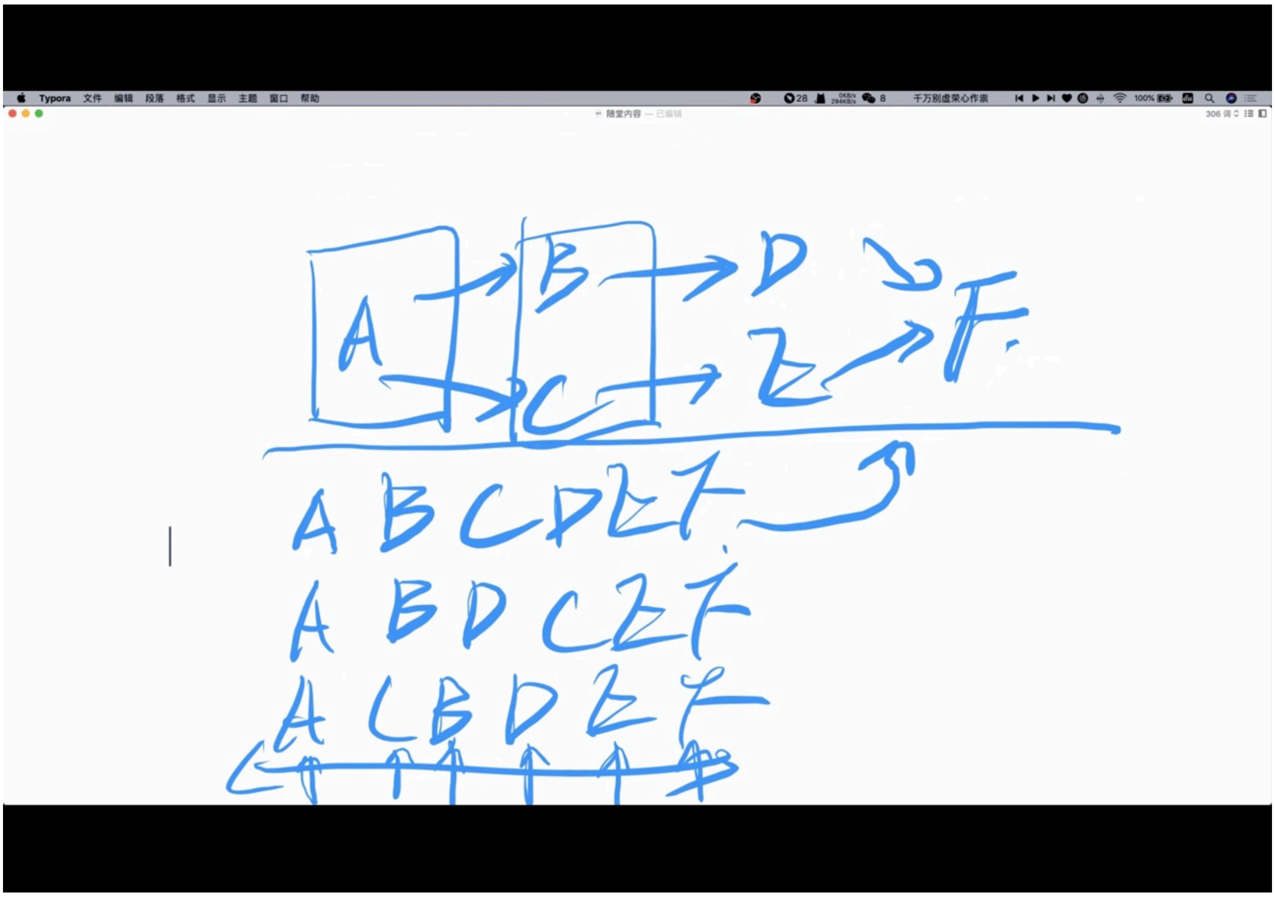
二、动态规划问题的求解套路

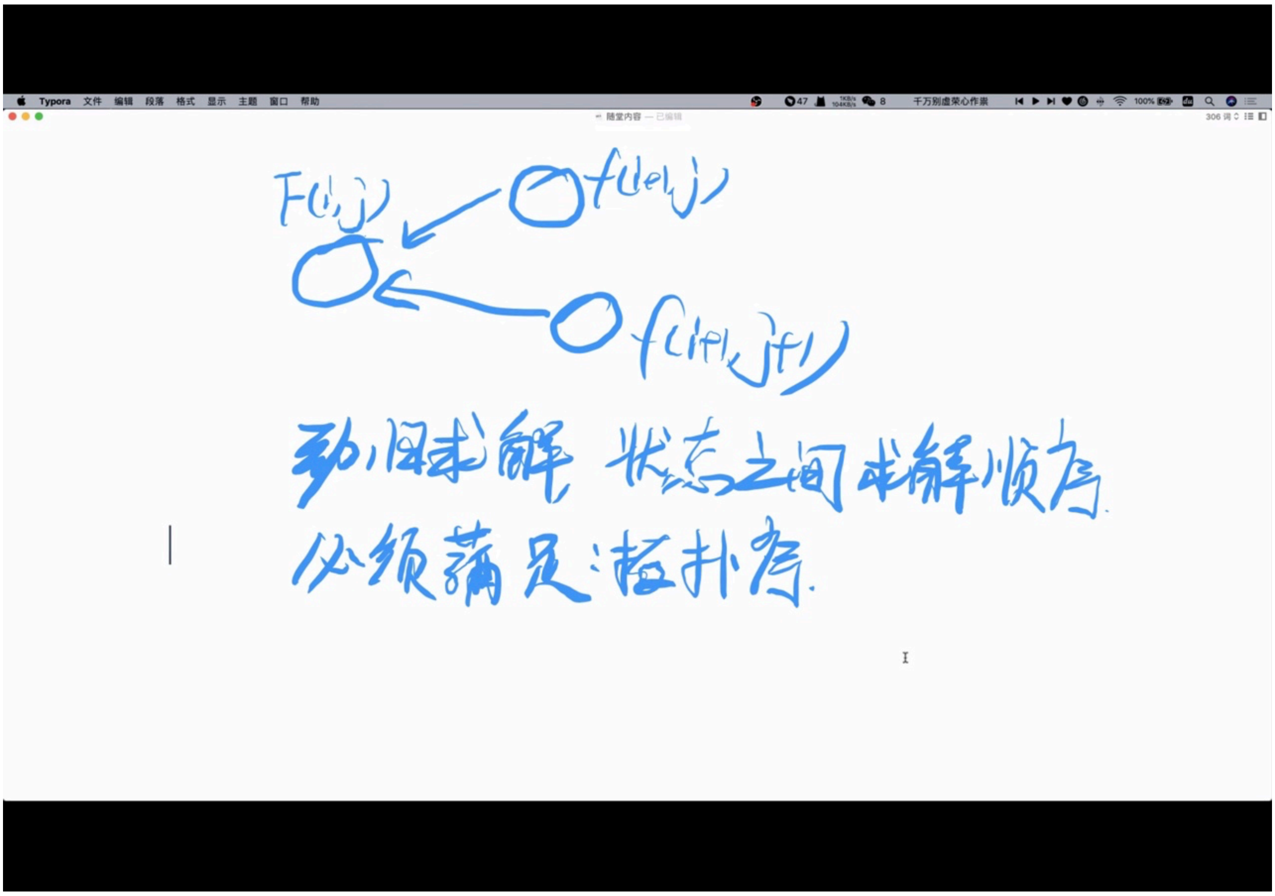
1. 第一步：确定动归状态
2. 第二步：推导状态转移方程，理解：转移、决策
3. 第三步：正确性证明，利用数学归纳法
4. 第四步：程序实现
5. 所谓的转移，把所有决定 $f(i,j)$ 最优值的状态，放入到决策过程中。

三、附加内容：拓扑序

图形结构是最最抽象的数据结构，必须理解成思维逻辑结构

1. 拓扑序是一种图形结构上的依赖顺序，一个图的拓扑序不唯一
2. 拓扑序的本质作用：是把图形结构上变成一个一维序列
3. 图形结构不能用循环遍历的，一维序列可以
4. 所有递推问题中的状态更新过程，本质上满足拓扑序





四、最长上升子序列

状态定义

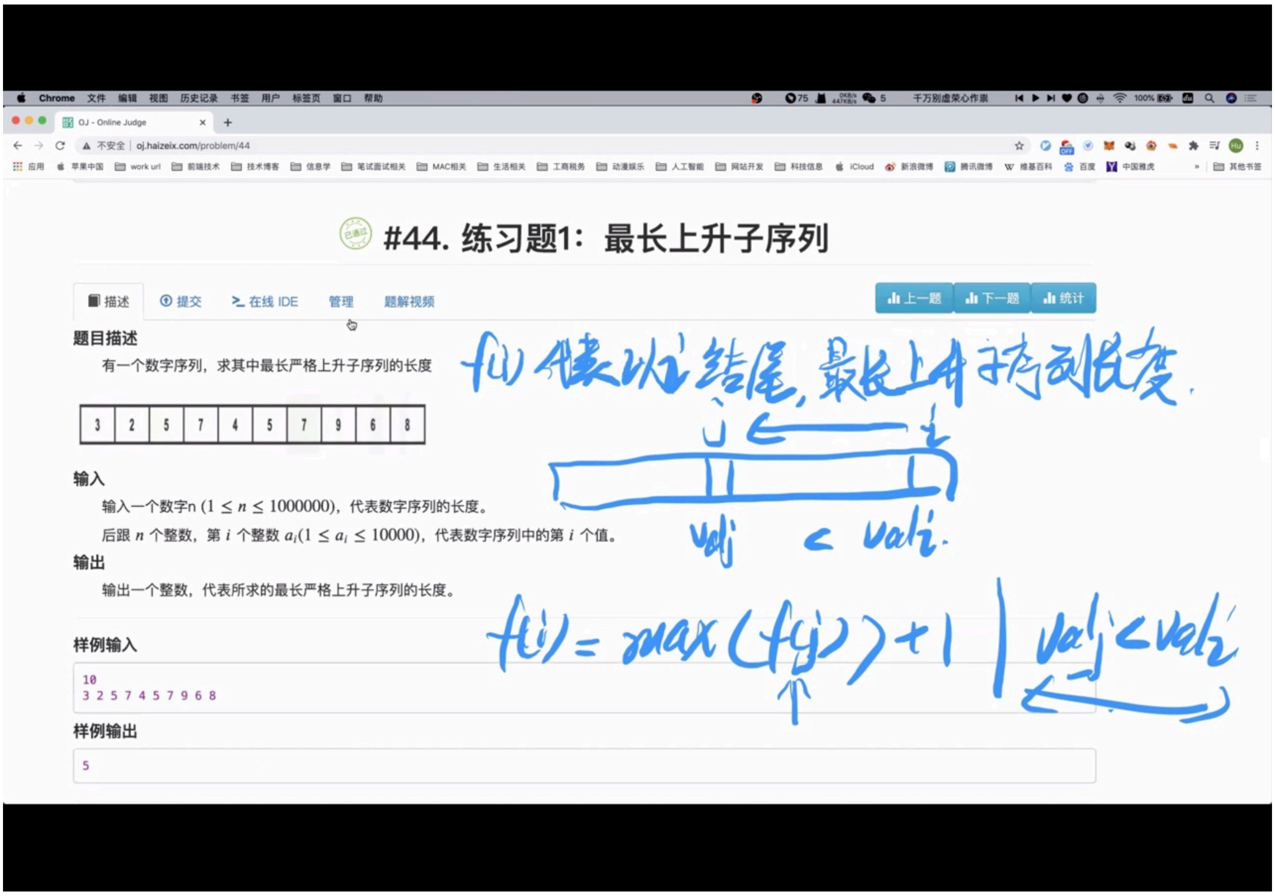
$f(i)$ 代表以为 i 为结尾的，最长上升子序列的长度

状态转移方程

$f(i) = max\left\{f(j)\right\} + 1 | j < i, val[j] < val[i]$

状态转移的时间复杂度：$O(n^2)$

后续重点：优化转移过程



五、最长公共子序列

状态定义

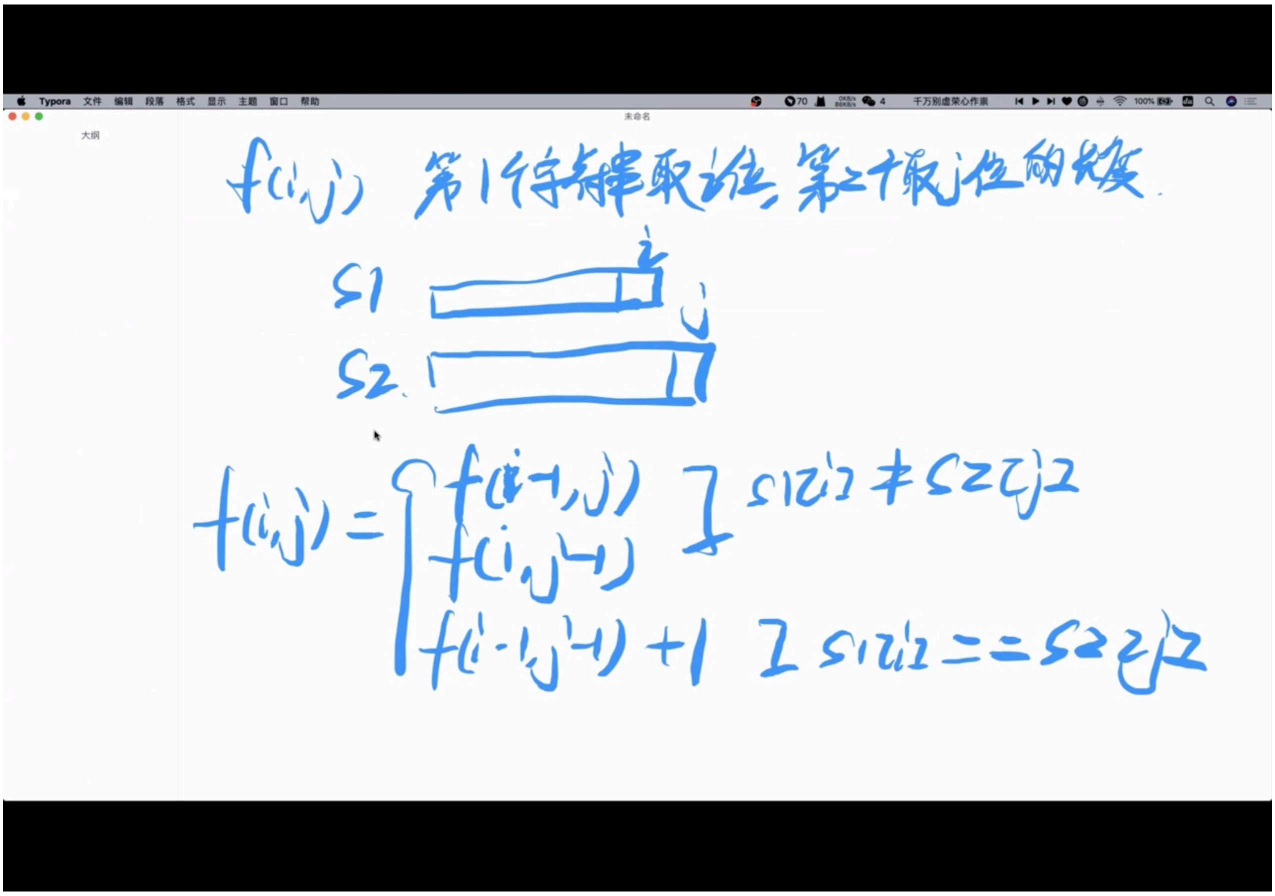
$f(i, j)$ 代表第一个字符串取前 i 位，第二个字符串取前 j 位的，最长公共子序列的长度

状态转移方程

$$f(i,j) = \begin{cases} max[f(i-1,j), f(i,j-1)] & val(i) \neq val(j) \\ f(i-1,j-1) & val(i) = val(j) \end{cases}$$

状态转移的时间复杂度：$O(n \times m)$

学习的重点：注意到，参与决策的状态数量，是会根据条件不同而改变的



六、课后作业题

1. HZOJ46-切割回文
2. HZOJ47-0/1背包
3. HZOJ48-完全背包
4. HZOJ49-多重背包