易错校验项：

1. Dp本身定义的ij代表的是不是i+1，j+1。
2. 数组范围校验，从头推
3. 初始化条件，没有考虑dp数组中j-1为-1时的初始化问题

初始条件：

1. 这个构造dp数组的时候正常构造，到后面状态转移的时候，如果你经常需要用到0这的状态的话，那你写长度+1的我觉得OK

动态规划解题的一般思路：

1. 确定dp数组的维度，主要看有几个可以自由改变的变量，如单个整数约束，数组长度，看改变这些能否达到出现子问题的效果。
2. 确定dp类型，常见的有线性dp与区间dp
3. 根据题中特征，看是否可以对输入进行预处理，如排序、单调栈等
4. 根据【每一步的选择（下面简称选择）】，写出状态转移方程。
5. ***判断dp顺序，看看初始条件在哪里，最简单的问题在哪里，问题是怎样演变的，到底哪里受哪里的影响。***
6. ***首先需要看【操作影响了什么，什么又影响了操作】***
7. ***选择改变了某些之前没注意到的状态 / 这些状态又反过来影响了选择】***

***这种好做，把这些没注意到的状态变成dp数组的一个维度即可***

1. ***选择改变了已经用了的状态】***

***复杂，待确认（好像没有什么已经用了的状态，只不过，你以为改变了，其实没有改变，你心里要有遍历是在哪里遍历的数）***

1. ***再看【选择会有哪些可能性，选择又有哪些约束】***

你的i或j一定是参与了操作的

1. ***新增的状态放等号左边还是右边，主要是看这个状态是否需要在某固定范围内遍历，是则放右边，否则放左边。***

***如果可以放左边也可以放右边，那要看这个状态需不需要3***

1. ***影响自己的，其实被影响的都是过去。Dp顺序可能就在这里***
2. ***最佳状态转移的理论证明：你要看遍历这边，能不能最佳状态直接转移，关键就是看最佳是否代表了所有状态***

再看以第一步确定的维度，能不能写出状态方程。如果无法写出，将【选择】影响的其他状态添加到dp数组维度中，扩展dp数组的维度，以更方便的描述状态转移

1. dp的顺序里，dp的头就是那个子问题中最简单的一个，最起始的
2. 困难题预处理，要看能不能减少操作维度，去掉无用的维度或排序(选择dp的执行顺序)。

tips：

1. 在第三步中，状态转移方程可以正写，也可以反写，即：

若当前问题可以数量和特征确定的子问题解决时，采用正写的方式，如下：

dp[i] = f(dp[k1], dp[k2], …, dp[kn]) n为单步可确定的值

若当前问题不能以数量和特征确定的子问题解决时，而子问题可以清晰地看到对后续问题的影响时，采用反写的方式，去为未来做记录，如下：

dp[i + k1] = f(dp[i])

dp[i + k2] = f(dp[i])

dp[i + k3] = f(dp[i])

1. 当dp数组某个维度的下一状态，仅由前一状态，或数量确定、规律简单的前几个状态得出时，可以省去一个维度，以优化空间。如：

若dp[i] = f(dp[i-1])，可将dp这个一维数组改为单个整数，以优化空间，即改为：dp = f(dp)

为什么要求最大的那个维度，总是要放在dp函数的返回值处？

BFS：

Bfs比较好的一点就是可以找最小的步数之类的事，因为它是一层一层来的

DFS:

Dfs适合找所有答案的，不过其实bfs也可以做