动态规划 子序列问题（连续和不连续）专题

最长上升子序列（不）

力扣题目链接

`dp[i]` 数组的含义：表示i之前包括i的以 `nums[i]` 为结尾的字符串最长递增子序列长度为 `dp[i]`

递推公式：

`if (nums[i] > nums[j]) dp[i] = max(dp[i], dp[j] + 1);`

因为是要求的子序列是不连续的，所以每个当前的值都需要和 `dp[i]` 算出来的最后那个值进行比较。所以需要两层**for循环**

```
    // i和之前下标为j的子序列长度有关系
    public int lengthOfLIS(int[] nums) {
        int[] dp = new int[nums.length];
        Arrays.fill(dp, 1);
        int result = 1;
        for (int i = 1; i < dp.length; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    dp[i] = Math.max(dp[i], dp[j] + 1);
                }
            }
            if(dp[i] > result) result = dp[i];
        }
        return result;
    }
/*
时间复杂度：O(n^2)
空间复杂度：O(n)
*/
```

最长公共子序列（不）

力扣题目链接

做完 最长重复子数组 后，来到该题，该题是不连续的，而且还是两个字符串（相当于两个数组）进行的比较，可以对比一下“两个数组”的比较下，连续和不连续子序列的区别。

`dp[i][j]` 数组的含义：字符串A中以 `nums[i]` 为结尾的 和 字符串B中以 `nums[j]` 为结尾最长的公共子序列的长度为 `dp[i][j]`

递推公式就是两个角度推算而来：①假如 `nums[i-1]` 和 `nums[j-1]` 相同 ②假如 `nums[i-1]` 和 `nums[j-1]` 不相同 （画图自行推演）

- 相同：`dp[i][j] = dp[i-1][j-1] + 1`
- 不相同：看字符串A[0, i - 2]与 字符串B[0,j - 1]的最长公共子序列 和 字符串A[0, i - 1]与 字符串B[0,j - 2]的最长公共子序列，看谁大取谁；即：`dp[i][j] = Math.max(dp[i-1][j],dp[i][j-1])`

到此：**可以看到连续的公共子序列和不连续的有啥区别了**，就是 连续的公共子序列不需要计算 当两个字符不相同的情况的，但是不连续的公共子序列是需要计算两个字符不相同的情况的。

```
public int longestCommonSubsequence(String text1, String text2) {
        int[][] dp = new int[text1.length() + 1][text2.length() + 1];
        for (int i = 1; i <= text1.length(); i++) {
            for (int j = 1; j <= text2.length(); j++) {
                if (text1.charAt(i-1) == text2.charAt(j-1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                }else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[text1.length()][text2.length()];
    }
/*
时间复杂度：O(n*m)
空间复杂度：O(n*m)
*/
```

不相交的线（不）

力扣题目链接

该题和 最长的公共子序列 解法和思路是一样的

```
 public int maxUncrossedLines(int[] nums1, int[] nums2) {
        int[][] dp = new int[nums1.length + 1][nums2.length + 1];
        for (int i = 1; i <= nums1.length; i++) {
            for (int j = 1; j <= nums2.length; j++) {
                if (nums1[i - 1] == nums2[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
        }
        return dp[nums1.length][nums2.length];
    }
```

最长连续递增序列

力扣题目链接

做完了最长上升子序列（不连续的）后，就来到了这个题目，可以对比一下看看**连续和不连续的子序列在解题上有啥区别。**

`dp[i]` 数组的含义：表示以 `nums[i]` 为结尾的字符 最长连续递增序列 长度为 `dp[i]`

递推公式：

如果 `nums[i]` > `nums[i-1]`，那么以 i 为结尾的连续递增的子序列长度一定 等于 以 i-1 为结尾的连续递增的子序列长度 + 1。即 `dp[i] = dp[i-1] + 1;`

因为这里要求的是连续子序列，所以就有必要比较`nums[i]`和`nums[i-1]`，而不用去比较`nums[j]`和`nums[i]`（j是从0到i的遍历）

```
public int findLengthOfLCIS(int[] nums) {
        if (nums.length <= 1) return 1;
        int[] dp = new int[nums.length];
        Arrays.fill(dp, 1);
        int result = 1;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] > nums[i - 1]) {
                dp[i] = Math.max(dp[i - 1] + 1, dp[i]);
            }
            if(dp[i] > result) result = dp[i];
        }
        return result;
    }
/*
时间复杂度：O(n)
空间复杂度：O(n)
*/
```

最长重复子数组

[力扣题目链接]

看到子序列问题的两个数组的比较，毫无疑问，用**二维数组**去解决。（也可以去画图解决，相当于模板题）

`dp[i][j]`数组的含义：以`nums[i]`为结尾的A数组和以`nums[j]`为结尾的B数组最长的重复子数组的长度为`dp[i][j]`

`dp[i][j]`从两个方向推出：（这个需要自行画图验证）

假如当前A数组的当前值与B数组的当前值**相等**：

`dp[i][j] = dp[i-1][j-1] + 1`

```
 public int findLength(int[] nums1, int[] nums2) {
        int[][] dp = new int[nums1.length + 1][nums2.length + 1];
        int result = 0;
        for (int i = 1; i <= nums1.length; i++) {
            for (int j = 1; j <= nums2.length; j++) {
                if (nums1[i - 1] == nums2[j - 1]) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                }
                if(dp[i][j] > result) result = dp[i][j];
            }
        }
        return result;
    }
/*
```

```
时间复杂度：O(n*m)
空间复杂度：O(n*m)
*/
```

最大子序和

力扣题目链接

`dp[i]` 数组的含义：以 `nums[i]` 为结尾的连续子数组的和最大为 `dp[i]` 。

递推公式：只有两个方向推出

- `dp[i-1] + nums[i]`，即：`nums[i]` 加入当前的连续子序列之和；
- `nums[i]`，从头开始算起。

取最大的：`dp[i] = max(dp[i - 1] + nums[i], nums[i]);`

```
 public static int maxSubArray(int[] nums) {
        if (nums.length == 0) {
            return 0;
        }
        int res = nums[0];
        int[] dp = new int[nums.length];
        dp[0] = nums[0];
        for (int i = 1; i < nums.length; i++) {
            dp[i] = Math.max(dp[i - 1] + nums[i], nums[i]);
            if(res < dp[i]) res = dp[i];
        }
        return res;
    }
/*
时间复杂度：O(n)
空间复杂度：O(n)
*/
```

总结

1. `dp[i]` 或者 `dp[i][j]` 的含义；
2. 在求最长的公共子序列的时候，需要注意该子序列是否是连续的，如果是连续的就不用理会 当 `nums[i-1]` 和 `nums[j-1]` 不相同的情况，但是假如是求不连续的时候，需要去推敲 `nums[i-1]` 和 `nums[j-1]` 不相同的情况；
3. 把握递推公式