435. 无重叠区间

题目地址（435. 无重叠区间）

https://leetcode-cn.com/problems/non-overlapping-intervals/

入选理由

1. 贪心的进阶难度。 贪心的题目都比较符合”常识和直觉“，不好归纳以及证明。因此大家可适当练习，我们就出两三道题 mu

题目描述

给定一个区间的集合，找到需要移除区间的最小数量，使剩余区间互不重叠。

注意：

可以认为区间的终点总是大于它的起点。
区间 [1,2] 和 [2,3] 的边界相互“接触”，但没有相互重叠。
示例 1:

输入: [[1,2], [2,3], [3,4], [1,3]]

输出: 1

解释: 移除 [1,3] 后，剩下的区间没有重叠。
示例 2:

输入: [[1,2], [1,2], [1,2]]

输出: 2

解释: 你需要移除两个 [1,2] 来使剩下的区间没有重叠。
示例 3:

输入: [[1,2], [2,3]]

输出: 0

解释: 你不需要移除任何区间，因为它们已经是无重叠的了。

标签

- 贪心

难度

- 中等

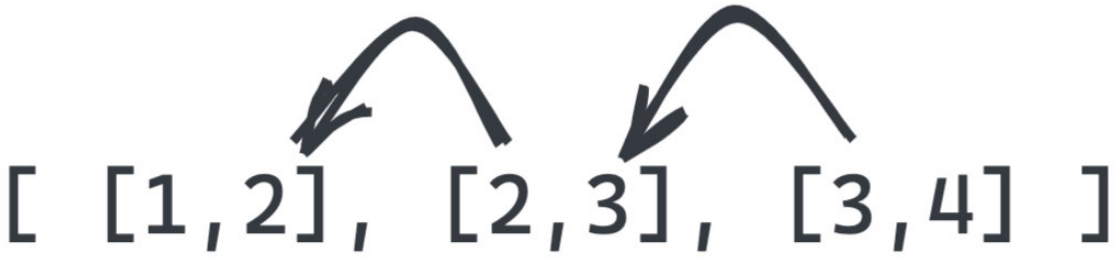
动态规划

思路

我们先来看下最终**剩下**的区间。由于剩下的区间都是不重叠的，因此剩下的**相邻区间的后一个区间的开始时间一定是不小于前一个区间的结束时间的**。 比如我们剩下的区间是 `[ [1,2], [2,3], [3,4] ]` 。就是第一个区间的 2 小于等于 第二个区间的 2，第二个区间的 3 小于等于第三个区间的 3。

不难发现如果我们将 前面区间的结束 和 后面区间的开始 结合起来看，其就是一个**非严格递增序列**。而我们的目标就是删除若干区间，从而**剩下最长的非严格递增子序列**。这不就是上面的题么？只不过上面是严格递增，这不重要，就是改个符号的事情。上面的题你可以看成是删除了若干数字，然后剩下**剩下最长的严格递增子序列**。 **这就是抽象的力量，这就是套路。**

如果对区间按照起点或者终点进行排序，那么就转化为上面的最长递增子序列问题了。和上面问题不同的是，由于是一个区间。因此实际上，我们是需要拿**后面的开始时间**和**前面的结束时间**进行比较。



而由于：

- 题目求的是需要移除的区间，因此最后 return 的时候需要做一个转化。
- 题目不是要求严格递增，而是可以相等，因此我们的判断条件要加上等号。

代码

```
class Solution:
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        n = len(intervals)
        if n == 0: return 0
        dp = [1] * n
```

```
        ans = 1
        intervals.sort(key=lambda a: a[0])

        for i in range(len(intervals)):
            for j in range(i - 1, -1, -1):
                if intervals[i][0] >= intervals[j][1]:
                    dp[i] = max(dp[i], dp[j] + 1)
                    break # 由于是按照开始时间排序的，因此可以剪枝

        return n - max(dp)
```

复杂度分析

- 时间复杂度：$O(n^2)$
- 空间复杂度：$O(n)$

贪心

思路

这道题还有一种贪心的解法，其效率要比动态规划更好。

LIS 也可以用 **贪心 + 二分** 达到不错的效率。

代码

```
class Solution:
    def lengthOfLIS(self, A: List[int]) -> int:
        d = []
        for s, e in A:
            i = bisect.bisect_left(d, e)
            if i < len(d):
                d[i] = e
            elif not d or d[-1] <= s:
                d.append(e)
        return len(d)
    def eraseOverlapIntervals(self, intervals: List[List[int]]) -> int:
        n = len(intervals)
        if n == 0: return 0
        ans = 1
        intervals.sort(key=lambda a: a[0])
        return n - self.lengthOfLIS(intervals)
```

复杂度分析

- 时间复杂度：$O(nlogn)$
- 空间复杂度：$O(n)$

参考

LIS 问题都可以用贪心的策略来解决，关于 LIS 问题可参考：

- **穿上衣服我就不认识你了？来聊聊最长上升子序列**

上一页 下一页