452-用最少数量的箭引爆气球

题述

452. 用最少数量的箭引爆气球

难度 中等 536 收藏 分享 切换为英文 接收动态 反馈

在二维空间中有许多球形的气球。对于每个气球，提供的输入是水平方向上，气球直径的开始和结束坐标。由于它是水平的，所以纵坐标并不重要，因此只要知道开始和结束的横坐标就足够了。开始坐标总是小于结束坐标。

一支弓箭可以沿着 x 轴从不同点完全垂直地射出。在坐标 x 处射出一支箭，若有一个气球的直径的开始和结束坐标为 x_{start}，x_{end}，且满足 $x_{start} \leqslant x \leqslant x_{end}$，则该气球会被引爆。可以射出的弓箭的数量没有限制。 弓箭一旦被射出之后，可以无限地前进。我们想找到使得所有气球全部被引爆，所需的弓箭的最小数量。

给你一个数组 `points` ，其中 `points [i] = [x_start, x_end]` ，返回引爆所有气球所必须射出的最小弓箭数。

示例 1：

```
输入：points = [[10,16],[2,8],[1,6],[7,12]]
输出：2
解释：对于该样例，x = 6 可以射爆 [2,8],[1,6] 两个气球，以及 x = 11 射爆另外两个气球
```

示例 2：

```
输入：points = [[1,2],[3,4],[5,6],[7,8]]
输出：4
```

示例 3：

```
输入：points = [[1,2],[2,3],[3,4],[4,5]]
输出：2
```

示例 4：

```
输入：points = [[1,2]]
输出：1
```

思路

如何使用最少的弓箭呢?

直觉上来看，貌似只射重叠最多的气球，用的弓箭一定最少，那么有没有当前重叠了三个气球，我射两个，留下一个和后面的一起射这样弓箭用的更少的情况呢?

尝试一下举反例，发现没有这种情况。

那么就试一试贪心吧！局部最优：当气球出现重叠，一起射，所用弓箭最少。全局最优：把所有气球射爆所用弓箭最少。

算法确定下来了，那么如何模拟气球射爆的过程呢？是在数组中移除元素还是做标记呢?

如果真实的模拟射气球的过程，应该射一个，气球数组就remove一个元素，这样最直观，毕竟气球被射了。

但仔细思考一下就发现：如果把气球排序之后，从前到后遍历气球，被射过的气球仅仅跳过就行了，没有必要让气球数组remote气球，只要记录一下箭的数量就可以了。

以上为思考过程，已经确定下来使用贪心了，那么开始解题。

为了让气球尽可能的重叠，需要对数组进行排序。

那么按照气球起始位置排序，还是按照气球终止位置排序呢?

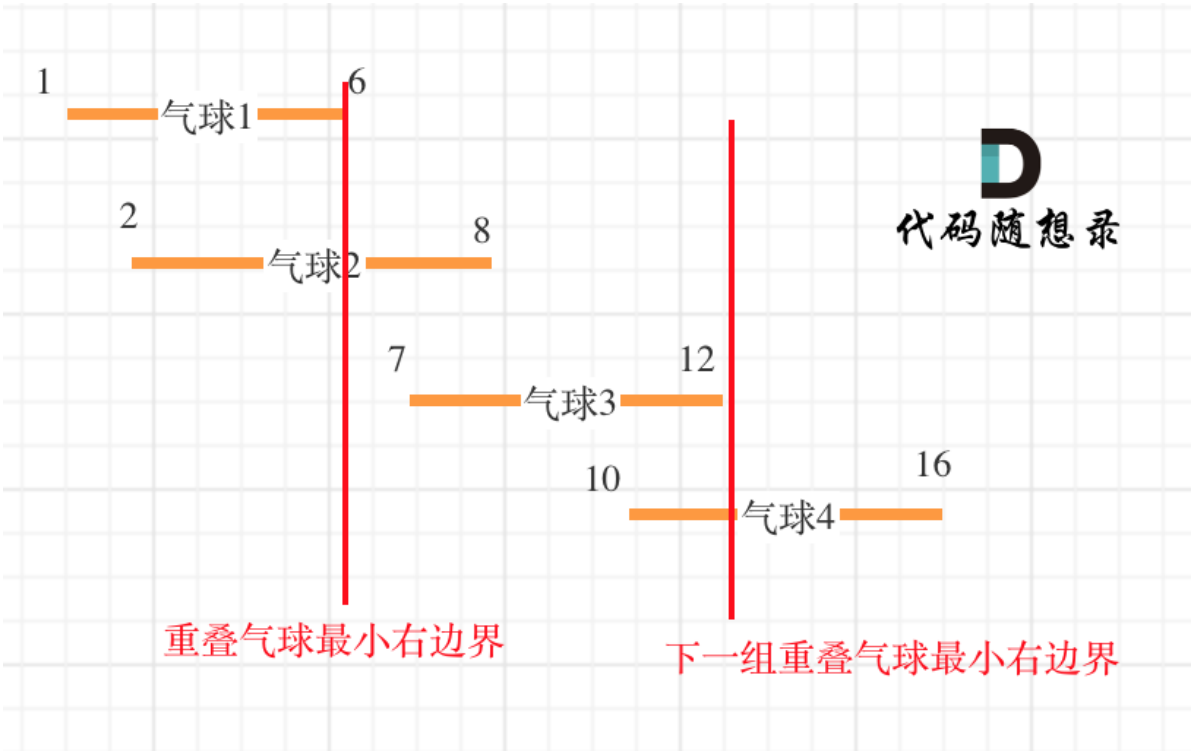
其实都可以！只不过对应的遍历顺序不同，我就按照气球的起始位置排序了。

既然按照起始位置排序，那么就从前向后遍历气球数组，靠左尽可能让气球重复。

从前向后遍历遇到重叠的气球了怎么办？

如果气球重叠了，重叠气球中右边边界的最小值 之前的区间一定需要一个弓箭。

以题目示例：[[10,16],[2,8],[1,6],[7,12]]为例，如图：（方便起见，已经排序）



可以看出首先第一组重叠气球，一定是需要一个箭，气球3，的左边界大于了 第一组重叠气球的最小右边界，所以再需要一支箭来射气球3了。

题解

C++

```
class Solution {
public:
    static bool cmp(const vector<int>& a,const vector<int>& b)
    {
        //排序规则
        return a[0] < b[0];
    }
    int findMinArrowShots(vector<vector<int>>& points)
    {
        //模拟法  模拟射爆最多的重叠气球
        if(points.size() == 0)
        {
            return 0;
        }

        sort(points.begin(),points.end(),cmp);  //按照数组的第一维度进行排序
        int result = 1; //points不为空的情况下，至少需要射一支箭
        for(int i = 1; i < points.size(); i++)
        {
            if(points[i][0] > points[i-1][1])
            {
```

```
                //气球i 和 气球i-1不挨着 不是 >=
                result++;   //需要一支箭
            }
            else
            {
                //挨着的情况下
                points[i][1] = min(points[i-1][1],points[i][1]);    //更新重叠气球
最小右边界
            }
        }
        return result;
    }
};
```

Python

```
class Solution:
    def findMinArrowShots(self, points: List[List[int]]) -> int:
        if len(points) == 0:
            return 0
        points.sort(key=lambda x:x[0])
        result = 1
        for i in range(1,len(points)):
            if(points[i][0] > points[i-1][1]):
                #气球i和气球i-1不挨着，注意这里不是>=
                result += 1
            else:
                # 更新重叠气球最小右边界
                points[i][1] = min(points[i-1][1],points[i][1])
        return result
```

思考

这道题目贪心的思路很简单也很直接，就是重复的一起射了，但是模拟起来感觉很复杂，不是那么容易上手。

而且寻找重复的气球，寻找重叠气球最小右边界，其实都有代码技巧。

贪心题目有时候就是这样，看起来很简单，思路很直接，但是一写代码就感觉贼复杂无从下手。