[452. 用最少数量的箭引爆气球]【贪心算法】【代码处理】

题目描述：

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。

```
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
```

题目分析：

1. 这道题用的是贪心算法
2. 贪心的是有重叠便可以进行射击
3. 如果不进行思考的话，很有可能会认为该题贪心的是每次都射击气球数量最多的点；
4. 尽管每次都设气球数量最多的点可能也可以得到答案，但是这样做的话，每次都要去寻找气球数量最多的点，并且还要去维护，会大大的增加代码的复杂度，增加时间和空间的消耗。
5. 降低贪心的难度：这里我们将气球的空间顺序按照左侧数据排列，只要有重叠便可以射击；
6. 选择不同程度的贪心策略，解题的难度也会大有不同；所以说，对于较难的贪心的题目，凭感觉去选择贪心的策略可能会使得做起来很麻烦；先进行一定的分析，可能会找到更好的贪心的策略。

代码：

```
class Solution {
private:
    static bool cmp(const vector<int>& a, const vector<int>& b) {
        return a[0] < b[0];
    }
public:
    int findMinArrowShots(vector<vector<int>>& points) {
        if (points.size() == 0) return 0;
        sort(points.begin(), points.end(), cmp);

        int result = 1; // points 不为空至少需要一支箭
        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，注意这里不是>=
                result++; // 需要一支箭
            }
            else {  // 气球i和气球i-1挨着
                points[i][1] = min(points[i - 1][1], points[i][1]); // 更新重叠气球最小右边界
            }
        }
        return result;
    }
};
```