452. 用最少数量的箭引爆气球

题目链接：https://leetcode-cn.com/problems/minimum-number-of-arrows-to-burst-balloons/

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 xstart，xend， 且满足 xstart ≤ x ≤ xend，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。

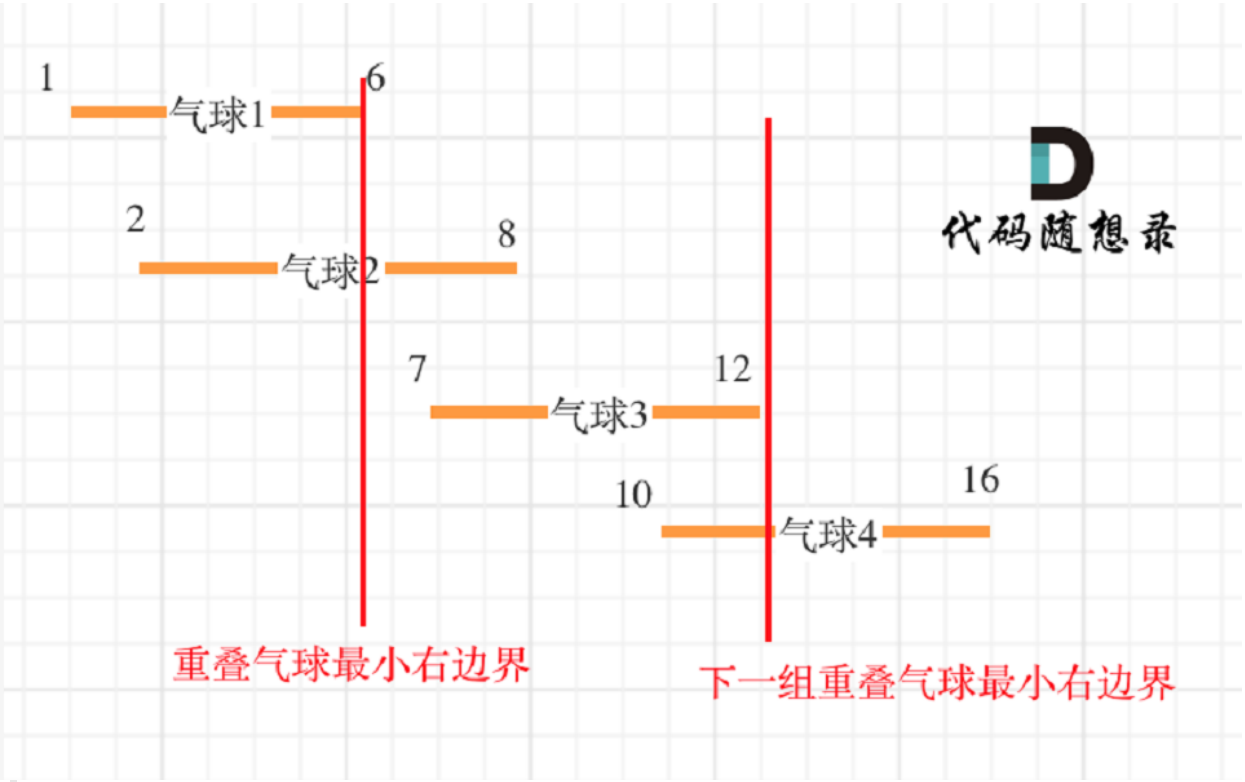
给你一个数组 points ，其中 points [i] = [xstart,xend] ，返回引爆所有气球所必须射出的最小弓箭数。

两个区间如何求交集区域

> [a1,a2],[b1,b2],交集区域为[c1,c2];
> if((a1>=b1&&a1<=b2)||(a2>=b1&&a2<=b2)){
> c1=max(a1,b1);
> c2=min(a2,b2);
> }
> 数据结构：
> vector<vector<int>> unionSet;//存放的现有的区域存在的交集，
> 假设unionSet中存在的交集有[c1,c2][d1,d2],现有一个区域[e1,e2],
> 1、若[e1,e2]与[c1,c2]以及[d1,d2]没有交集,那么就新建一个交集区域[e1,e2]插入unionSet
> 2、若[e1,e2]只与[c1,c2]或[d1,d2]有交集则更新相应的交集区域
> 3、若[e1,e2]与[c1,c2]和[d1,d2]均有交集，那么就选择更新后交集最大的区域更新

进一步优化

> 其实气球的重叠情况没必要用vector<vector<int>> unionSet;显性表示出来。
> 只要按照气球的起始位置排序即可。



只要当前要考虑的气球的左边界不在前面以及考虑的重叠气球的最小右边界那么就需要新的一支箭。

最小右边界的更新时间:在于当前考虑的气球可以与前面气球重叠时

```
class Solution {
public:
 //事先定义如何排序
 static bool cmp(const vector<int> a,const vector<int> b){
 return a[0]<b[0];
 }
    int findMinArrowShots(vector<vector<int>>& points) {
 if(points.size()==0) return 0;
 sort(points.begin(),points.end(),cmp);
  int result=1;//所需要的箭数
  for(int i=1;i<points.size();i++){
  if(points[i][0]>points[i-1][1]){
  //条件一定不能为points[i][0]>=points[i-1][1]
  result++;//此时当前考虑的气球没有与前面重叠
  }else{//存在重叠时
  points[i][0]=min(points[i-1][1],points[i][1]);
  }
  }
  return result;
     }
};
```

注意:

注意题目中说的是：满足 xstart ≤ x ≤ xend，则该气球会被引爆。那么说明两个气球挨在一起不重叠也可以一起射爆，
所以代码中 `if (points[i][0] > points[i - 1][1])` 不能是>=

435. 无重叠区间

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

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。

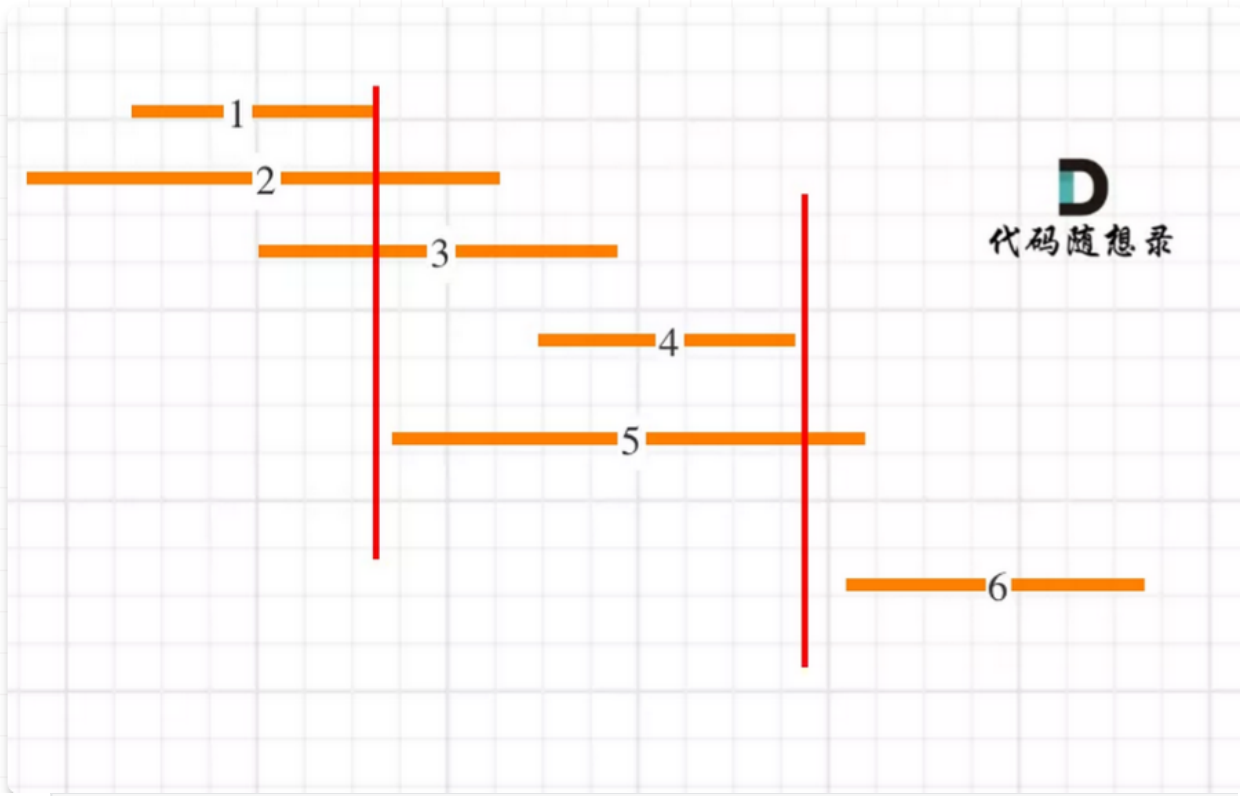
示例 3:

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。

这道题目类似于上面那道题,需要事先**对给定的数组进行排序,**
思考1：按照左边界排序还是右边界排序呢？
其实两者都行，如果按照左边界排序就要从左向右遍历，如果按照右边界排序就要从右向左遍历.
按照左边界排序，就要从左向右遍历，因为右边界越小越好，只要右边界越小，留给下一个区间的空间就越大，所以从左向右遍历，优先选右边界小的
排序后隐藏条件：
intervals[i][0]>=intervals[i-1][0]
如果又满足intervals[i][0]<intervals[i-1][1]
那么第i个区间与第i-1个区间有重叠




```cpp
class Solution {
public:
 static bool cmp(const vector<int> a,const vector<int> b){
 return a[0]<b[0];
 }
    int eraseOverlapIntervals(vector<vector<int>>& intervals) {
 int result=0;
 if(intervals.size()==0) return result;
 sort(intervals.begin(),intervals.end(),cmp);
  for(int i=1;i<intervals.size();i++){
  if(intervals[i][0]<intervals[i-1][1]){
  result++;
  //一定要用min才能使得移除区间最小数量。
  intervals[i][1]=min(intervals[i][1],intervals[i-1][1]);
  }
  }
  return result;
    }
};
```

763.划分字母区间

题目链接：https://leetcode-cn.com/problems/partition-labels/

字符串 S 由小写字母组成。我们要把这个字符串划分为尽可能多的片段，同一字母最多出现在一个片段中。返回一个表示每个字符串片段的长度的列表。

示例：
输入：S = "ababcbacadefegdehijhklij"
输出：[9,7,8]
解释：
划分结果为 "ababcbaca", "defegde", "hijhklij"。
每个字母最多出现在一个片段中。
像 "ababcbacadefegde", "hijhklij" 的划分是错误的，因为划分的片段数较少。

提示：

- S的长度在[1, 500]之间。
- S只包含小写字母 'a' 到 'z' 。

题目要求同一个字母最多出现在一个片段中，那么如何把同一个字母都圈在同一个区间里呢？

> 在遍历过程中相当于是要将每一个字母的边界找到，**「如果找到之前遍历过的所有字母的最远边界，说明这个边界就是分割点了」**此时前面出现过所有字母，最远也就到这个边界了。
> 首先先将题目给出的数据结构转换为区间的数据结构：
> map<char,vector<int>> intervalMap;=>再转换为vector<vector<int>> intervalArray;
> 然后接下来就是按照左边界排序，然后进行重叠区域处理
> if(intervalArray[i][0]<=intervalArray[i-1][1]){//则有重叠区域
> intervalArray[i][0]=intervalArray[i-1][0];
> intervalArray[i][1]=max(intervalArray[i][1],intervalArray[i-1][1]);
> }else{
> result.push_back(intervalArray[i-1][1]-intervalArray[i-1][0]+1);
> }
> 注意再跳出循环时还要再往result里push一个元素
> 最后输出vector<int> result;

```
class Solution {
public:
    static bool cmp(vector<int> a,vector<int> b){
        return a[0]<b[0];
    }
```

```
    vector<int> partitionLabels(string S) {
        map<char,vector<int>> intervalsMap;
        vector<vector<int>> intervalsArray;
        vector<int> result;
        for(int i=0;i<S.size();i++){
            if(intervalsMap.find(S[i])!=intervalsMap.end()){
                intervalsMap[S[i]][1]=i;//更新右边界
            }else{
                vector<int> a={i,i};
                intervalsMap[S[i]]=a;
            }
        }
        map<char,vector<int>>::iterator it=intervalsMap.begin();
        while(it!=intervalsMap.end()){
            intervalsArray.push_back(it->second);
            it++;
        }
        sort(intervalsArray.begin(),intervalsArray.end(),cmp);
        for(int i=1;i<intervalsArray.size();i++){
            if(intervalsArray[i][0]<=intervalsArray[i-1][1]){
                intervalsArray[i][0]=intervalsArray[i-1][0];
                intervalsArray[i][1]=max(intervalsArray[i][1],intervalsA
rray[i-1][1]);
            }else{
                result.push_back(intervalsArray[i-1][1]-
intervalsArray[i-1][0]+1);
            }
        }
        int len=intervalsArray.size()-1;
        result.push_back(intervalsArray[len][1]-intervalsArray[len]
[0]+1);
        return result;
    }
};
```

56. 合并区间

题目链接：https://leetcode-cn.com/problems/merge-intervals/

给出一个区间的集合，请合并所有重叠的区间。

示例 1:

输入: intervals = [[1,3],[2,6],[8,10],[15,18]]

输出: [[1,6],[8,10],[15,18]]

解释: 区间 [1,3] 和 [2,6] 重叠, 将它们合并为 [1,6].

示例 2:

输入: intervals = [[1,4],[4,5]]

输出: [[1,5]]

解释: 区间 [1,4] 和 [4,5] 可被视为重叠区间。

注意：输入类型已于2019年4月15日更改。请重置默认代码定义以获取新方法签名。

提示：

- intervals[i][0] <= intervals[i][1]

区间问题都是一个套路**(有一个代码模板)**

```
class Solution {
public:
    static bool cmp(const vector<int> a,const vector<int> b){
        return a[0]<b[0];
    }
    vector<vector<int>> merge(vector<vector<int>>& intervals) {
        vector<vector<int>> result;
        sort(intervals.begin(),intervals.end(),cmp);//从小到大排序
        for(int i=1;i<intervals.size();i++){
            if(intervals[i][0]<=intervals[i-1][1]){
                //有重叠区域
                intervals[i][0]=intervals[i-1][0];
                intervals[i][1]=max(intervals[i-1][1],intervals[i][1]);
            }else{
                result.push_back(vector<int>{intervals[i-1][0],intervals[i-1][1]});
            }
        }
```

```
        int len=intervals.size()-1;
        result.push_back(vector<int>{intervals[len][0],intervals[len][1]});
        return result;
    }
};
```