# 题目

给定一个区间的集合 intervals ，其中 intervals[i] = [starti, endi] 。返回需要移除区间的最小数量，使剩余区间互不重叠 。

注意 只在一点上接触的区间是不重叠的。例如 [1, 2] 和 [2, 3] 是不重叠的。

示例 1:

输入: intervals = [[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: intervals = [ [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: intervals = [ [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。

提示:

1 <= intervals.length <= 105

intervals[i].length == 2

-5 \* 104 <= starti < endi <= 5 \* 104

# 分析

思路：

核心思路是优先保留结束时间早的区间。因为结束时间早的区间留给其他区间的空间更大，能有效减少重叠的可能性。具体步骤如下：

1、排序区间：按照区间的结束时间从小到大排序。

2、遍历筛选：遍历排序后的区间，保留与上一个保留区间不重叠的区间（即当前区间的开始时间 >= 上一个保留区间的结束时间）。

3、计算移除数量：总区间数减去保留的区间数，即为需要移除的最小数量。

代码：

class Solution {

public:

int eraseOverlapIntervals(vector<vector<int>>& intervals) {

if (intervals.empty()) return 0;

// 按照区间的结束时间从小到大排序

sort(intervals.begin(), intervals.end(), [](const vector<int>& a, const vector<int>& b) {

return a[1] < b[1];

});

int count = 1; // 至少可以保留一个区间

int lastEnd = intervals[0][1]; // 记录上一个保留区间的结束时间

for (int i = 1; i < intervals.size(); ++i) {

// 如果当前区间的开始时间 >= 上一个区间的结束时间，说明不重叠，可保留

if (intervals[i][0] >= lastEnd) {

count++;

lastEnd = intervals[i][1]; // 更新结束时间

}

}

// 需要移除的数量 = 总数量 - 保留的数量

return intervals.size() - count;

}

};

解释：

1、排序：通过按结束时间排序，确保我们总是优先考虑那些结束早的区间，为后续区间留出更多空间。

2、筛选保留区间：从排序后的第一个区间开始，每次选择与上一个保留区间不重叠的区间（当前区间的开始时间 >= 上一个区间的结束时间），并更新最后保留区间的结束时间。

3、计算结果：总区间数减去保留的区间数，即为需要移除的最小数量。

这种方法的时间复杂度主要来自排序，为O(n log n)（n为区间数量），空间复杂度为O(1)（忽略排序所需的栈空间），高效适用于题目中n可达10^5的场景。