**56. 合并区间**

给出一个区间的集合，请合并所有重叠的区间。

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].

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。

|  |
| --- |
| class Solution {  public:  vector<vector<int>> merge(vector<vector<int>>& intervals) {  if (intervals.size() < 2) {  return intervals;  }  sort(intervals.begin(), intervals.end(), [](const vector<int>& left, const vector<int>& right) {  return left[0] < right[0];  });  vector<vector<int>> res;  res.push\_back(intervals[0]);  for (size\_t i = 1; i < intervals.size(); i++) {  if ((\*res.rbegin())[1] >= intervals[i][0]) {  (\*res.rbegin())[1] = max((\*res.rbegin())[1], intervals[i][1]);  } else {  res.push\_back(intervals[i]);  }  }  return res;  }  }; |

**57. 插入区间**

给出一个无重叠的 ，按照区间起始端点排序的区间列表。

在列表中插入一个新的区间，你需要确保列表中的区间仍然有序且不重叠（如果有必要的话，可以合并区间）。

输入：intervals = [[1,3],[6,9]], newInterval = [2,5]

输出：[[1,5],[6,9]]

输入：intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

输出：[[1,2],[3,10],[12,16]]

解释：这是因为新的区间 [4,8] 与 [3,5],[6,7],[8,10] 重叠。

|  |
| --- |
| class Solution {  public:  vector<vector<int>> insert(vector<vector<int>>& intervals, vector<int>& newInterval) {  vector<vector<int>> res;  int i=0;  int length = intervals.size();  while(i < length && intervals[i][1] < newInterval[0]){  res.push\_back(intervals[i]);  i++;  }  while(i< length && intervals[i][0] <= newInterval[1]){  newInterval[0] = min(newInterval[0], intervals[i][0]);  newInterval[1] = max(newInterval[1], intervals[i][1]);  i++;  }  res.push\_back(newInterval);  while(i< length)  res.push\_back(intervals[i++]);  return res;  }  }; |

**435. 无重叠区间**

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。

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。

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。

|  |
| --- |
| class Solution {  public int eraseOverlapIntervals(int[][] intervals) {  Arrays.sort(intervals,new Comparator<int [] >(){  public int compare(int [] a1,int [] a2) {  return a1[1] - a2[1]; //升序排列  }  });  int count = 1; //最多能组成的不重叠区间个数  int preEnd = intervals[0][1];  for (int i = 1; i < intervals.length; i++) {  if (intervals[i][0] < preEnd) {  continue;  }  preEnd = intervals[i][1];  count++;  }  return intervals.length - count;  }  } |

贪心策略，先计算最多能组成的不重叠区间个数，然后用区间总个数减去不重叠区间的个数。

在每次选择中，选择的区间结尾越小，留给后面的区间的空间越大，那么后面能够选择的区间个数也就越大。按区间的结尾进行升序排序，每次选择结尾最小，并且和前一个区间不重叠的区间。

**452. 用最少数量的箭引爆气球**

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y坐标并不重要，因此只要知道开始和结束的x坐标就足够了。开始坐标总是小于结束坐标。平面内最多存在多个气球。

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。

输入:[[10,16], [2,8], [1,6], [7,12]]

输出:2

对于该样例，我们可以在x = 6（射爆[2,8],[1,6]两个气球）和 x = 11（射爆另外两个气球）。

|  |
| --- |
| class Solution {  public:  static bool cmp(vector<int> a, vector<int>b)  {  return a[1] < b[1];  }  int findMinArrowShots(vector<vector<int>>& points) {  if(points.empty())  return 0;  sort(points.begin(), points.end(), cmp);  int end = points[0][1];  int res = 1;  for(int i = 1; i < points.size(); i++)  {  if(points[i][0] > end)  {  res++;  end = points[i][1];  }  }  return res;  }  }; |

这题和435. 无重叠区间一样，按照第二个分量排序。

或者可以理解为点覆盖问题。比如对于第一个气球，必须有一箭需要刺破，因此次数至少＋1，那么就看这个箭还能刺破多少气球

**455. 分发饼干**

假设你是一位很棒的家长，想要给你的孩子们一些小饼干。但是，每个孩子最多只能给一块饼干。

输入: g = [1,2,3], s = [1,1]

输出: 1

你有三个孩子和两块小饼干，3个孩子的胃口值分别是：1,2,3。虽然你有两块小饼干，由于他们的尺寸都是1，你只能让胃口值是1的孩子满足。

输入: g = [1,2], s = [1,2,3]

输出: 2

你有两个孩子和三块小饼干，2个孩子的胃口值分别是1,2。

你拥有的饼干数量和尺寸都足以让所有孩子满足。

所以你应该输出2.

|  |
| --- |
| class Solution {  public int findContentChildren(int[] g, int[] s) {  //用最小的饼干满足孩子 每次分发出去最小的饼干  Arrays.sort(g);  Arrays.sort(s);  int i = 0, j = 0, sum = 0;  while(i < g.length && j < s.length){  if(s[j] >= g[i]){  sum ++;  i++;  }  j ++;  }  return sum;  }  } |

**327. 区间和的个数**

给定一个整数数组 nums，返回区间和在 [lower, upper] 之间的个数，包含 lower 和 upper。

区间和 S(i, j) 表示在 nums 中，位置从 i 到 j 的元素之和，包含 i 和 j (i ≤ j)。

最直观的算法复杂度是 O(n2) ，请在此基础上优化你的算法。

输入: nums = [-2,5,-1], lower = -2, upper = 2,

输出: 3

解释: 3个区间分别是: [0,0], [2,2], [0,2]，它们表示的和分别为: -2, -1, 2。

|  |
| --- |
| class Solution {  public:  int countRangeSum(vector<int>& nums, int64\_t l, int64\_t u) {  auto sum = int64\_t{};  auto s = multiset<int64\_t>{};  auto cnt = 0;  for(auto n : nums)  {  sum += n;  if(sum >= l && sum <= u)  {  ++cnt;  }  auto li = s.lower\_bound(sum - u);  auto ui = s.lower\_bound(sum - l + 1);  cnt += distance(li, ui);  s.insert(sum);  }  return cnt;  }  }; |