区间问题，即线段问题，如合并区间，找出交集等，常用技巧如下：

1. 排序。常见排序，按照起点升序，起点相同则按照终点降序；也可以按照终点升序，终点相同则按照起点升序等。
2. 画图。分析所有可能的情况。

区间覆盖问题

> 给你一个区间列表，请你删除列表中被其他区间所覆盖的区间。
> 只有当 c <= a 且 b <= d 时，我们才认为区间 [a,b) 被区间 [c,d) 覆盖。
> 在完成所有删除操作后，请你返回列表中剩余区间的数目。

分析：

1. 首先排序，按起点升序，起点相同按终点降序；
2. 线段区间有三种可能，
    - 覆盖：需要删除的场景；
    - 重叠部分，可以将这些线段合并，更新终点；
    - 不相交，更新当前线段的起点和终点。
3. 最后要求删除覆盖线段之后的线段数。

```
class Solution {
public:
  int removeCoveredIntervals(std::vector<std::vector<int>>& intervals) {
    int len = intervals.size();
    if (len == 0) {
      return 0;
    }

    // 按起点升序，起点相同按终点降序
    std::sort(intervals.begin(),
              intervals.end(),
              [](const std::vector<int> a, const std::vector<int> b) {
                if (a[0] == b[0]) {
                  return a[1] > b[1];
                }
                return a[0] < b[0];
              });

    int res   = 0;                // 重叠区间数
    int left  = intervals[0][0];  // 合并区间左端点
    int right = intervals[0][1];  // 合并区间右端点

    for (int i = 1; i < len; i++) {
      // 情况一，区间被覆盖
      if (left <= intervals[i][0] && right >= intervals[i][1]) {
        res++;
      }
```

```
      // 情况二，合并区间
      if (right >= intervals[i][0] && right <= intervals[i][1]) {
        right = intervals[i][1];
      }

      // 情况三区间不相交
      if (right < intervals[i][0]) {
        left  = intervals[i][0];
        right = intervals[i][1];
      }
    }

    return len - res;
  }
};
```

区间合并

以数组 intervals 表示若干个区间的集合，其中单个区间为 intervals[i] = [starti, endi] 。请你合并所有重叠的区间，并返回一个不重叠的区间数组，该数组需恰好覆盖输入中的所有区间。

分析： 注意：在循环结束时，最后一对<left, right>未加入结果集。

```
class Solution {
public:
  std::vector<std::vector<int>> merge(
      std::vector<std::vector<int>> &intervals) {
    int                           len = intervals.size();
    std::vector<std::vector<int>> res;
    if (0 == len) {
      return res;
    }

    // 排序，起点升序，起点相同终点降序
    std::sort(intervals.begin(),
              intervals.end(),
              [](std::vector<int> &a, std::vector<int> &b) {
                if (a[0] == b[0]) {
                  return a[1] > b[1];
                }
                return a[0] < b[0];
              });

    // print(intervals);

    int left  = intervals[0][0];  // 起点
    int right = intervals[0][1];  // 终点
```

```
      for (int i = 1; i < len; i++) {
        // 覆盖
        if (left <= intervals[i][0] && right >= intervals[i][1]) {
          continue;
        }

        // 相交，合并
        if (right >= intervals[i][0] && right <= intervals[i][1]) {
          // std::cout << left << ' ' << right << std::endl;

          right = intervals[i][1];
        }

        // 不相交，加入结果集
        if (right < intervals[i][0]) {
          res.push_back(std::vector<int>{left, right});
          left  = intervals[i][0];
          right = intervals[i][1];
        }
      }

      res.push_back(std::vector<int>{left, right});

      return res;
    }
  };
```

区间交集