218. 天际线问题

题目描述：

城市的天际线是从远处观看该城市中所有建筑物形成的轮廓的外部轮廓。给你所有建筑物的位置和高度，请返回由这些建筑物形成的 **天际线** 。

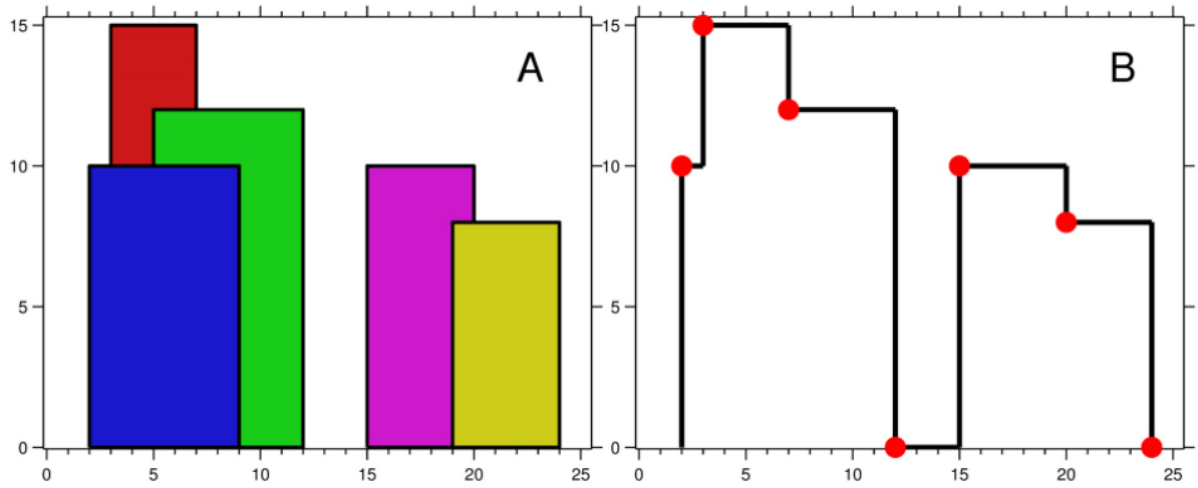
每个建筑物的几何信息由数组 `buildings` 表示，其中三元组 `buildings[i] = [lefti, righti, heighti]` 表示：

- $left_i$ 是第 `i` 座建筑物左边缘的 `x` 坐标。
- $right_i$ 是第 `i` 座建筑物右边缘的 `x` 坐标。
- $height_i$ 是第 `i` 座建筑物的高度。

天际线 应该表示为由 “关键点” 组成的列表，格式 $[[x_1,y_1],[x_2,y_2],...]$ ，并按 **x 坐标** 进行 **排序** 。**关键点是水平线段的左端点**。列表中最后一个点是最右侧建筑物的终点， `y` 坐标始终为 `0` ，仅用于标记天际线的终点。此外，任何两个相邻建筑物之间的地面都应被视为天际线轮廓的一部分。

注意：输出天际线中不得有连续的相同高度的水平线。例如 `[...[2 3], [4 5], [7 5], [11 5], [12 7]...]` 是不正确的答案；三条高度为 5 的线应该在最终输出中合并为一个： `[...[2 3], [4 5], [12 7], ...]`

示例 1：




```
输入：buildings = [[2,9,10],[3,7,15],[5,12,12],
[15,20,10],[19,24,8]]
输出：[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],
[24,0]]
```

解释：

图 A 显示输入的所有建筑物的位置和高度，

图 B 显示由这些建筑物形成的天际线。图 B 中的红点表示输出列表中的关键点。

代码实现：

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        //设置一个关于pair<int,int>类型的比较函数，从小到大的一个排序
auto cmp = [](const pair<int, int>& a, const pair<int, int>& b) -> bool
{ return a.second < b.second; };
        priority_queue<pair<int, int>, vector<pair<int, int>>,
decltype(cmp)> que(cmp);

        //此处也可以设置一个优先队列
        //存在关键点的地方都为边缘点
        vector<int> boundaries;
        for (auto& building : buildings) {
            boundaries.emplace_back(building[0]);
            boundaries.emplace_back(building[1]);
        }
        sort(boundaries.begin(), boundaries.end());
```

```
        vector<vector<int>> ret;
        int n = buildings.size(), idx = 0;
        for (auto& boundary : boundaries) {
            //将符合条件的入队
            while (idx < n && buildings[idx][0] <= boundary) {
                //按高度从小到大排序
                que.emplace(buildings[idx][1], buildings[idx][2]);
                idx++;
            }
            while (!que.empty() && que.top().first <= boundary) {
                que.pop();
            }

            int maxn = que.empty() ? 0 : que.top().second;
            if (ret.size() == 0 || maxn != ret.back()[1]) {
                ret.push_back({boundary, maxn});
            }
        }
        return ret;
    }
};
```