#LeetCode56

https://leetcode.com/problems/merge-intervals/description/

Yifeng Zeng

#题目描述

Merge Intervals

#思路报告

如果要merge a list of intervals，我们首先想到的是在这个list里面选两个interval来merge，那么就是两个for循环，O(n^2)。对于每一对interval： i1，i2，我们要看i1.start是否在[i2.start, i2.end]之间，还要看i2.star是否在[i1.start, i1.end]之间。那么能不能提速呢？能不能只检查一半，即只检查i1.start是否在[i2.start, i2.end]之间呢？可以，只要我们保证i1.start >= i2.start就可以了。怎样保证呢？我们可以把所有的intervals按照i.start排序就好了，那么sort的时间就是O(nlogn)。所以我们要自己写一个Interval的comparator。那么假设intervals已经排好序了，我们还需要两个for循环做O(n^2)的时间吗？明显不用，我们只需要把第一个interval拿出来(假设为prev)，作为一个base case，再拿后面的interval拿出来(假设为cur)跟它比较。因为我们已经排序了，所以一定有prev.start <= cur.start，那么我们只用检查cur.end是否小于等于pre.end，如果是，那么我们可以merge，如果不是，证明prev不会有其他的可以merge了，因为后面的i.start也一定大于等于pre.end。那么我们就可以把这个prev放在result里面了。但是我们要注意的是，最后一个prev，它将不再会有其他的interval跟它比较了，所以也要把它放到result里面，就刚跟LC186最后一个word类似。所以我们把merge的时间复杂度降到了O(n)，但是排序用了O(nlogn)，所以整个算法的时间复杂度是O(nlogn)。

代码如下：

```
class Solution {
    Comparator<Interval> myComparator = new Comparator<Interval>() {
        @Override
        public int compare(Interval o1, Interval o2) {
            return o1.start - o2.start;
        }
    };
    public List<Interval> merge(List<Interval> intervals) {
        List<Interval> res = new ArrayList<>();
        if (intervals == null || intervals.size() == 0) {
```

```
            return res;
        }
        Collections.sort(intervals, myComparator);
        Interval prev = intervals.get(0);
        for (int i = 1; i < intervals.size(); i++) {
            Interval cur = intervals.get(i);
            if (cur.start <= prev.end) {
                prev.end = Math.max(prev.end, cur.end);
            } else {
                res.add(prev);
                prev = cur;
            }
        }
        res.add(prev);
        return res;
    }
}
```

#套路总结

- 我们从最暴力的O(n^2)看出来，两个interval相互比较能不能merge，想到能否只比较一边，从而想到排序。
- 当输入杂乱无章的时候可以考虑排序，固定一端检查另一端。
- 空间复杂度???
    - 其实我的理解是作为输出的space不计入复杂度，那么就是O(1)，但是这个res确实是extra的，由于输入是List，那么我们想要在它本身里面去merge后删除一个interval其实代价也是O(n)，那么对List做“inplace”操作反而效率不高，所以就直接另外new了一个List，这点可以跟面试官讨论。