Daily Coding Problem Blog

Daily Coding Problem #286

Problem

This problem was asked by VMware.

The skyline of a city is composed of several buildings of various widths and heights, possibly overlapping one another when viewed from a distance. We can represent the buildings using an array of `(left, right, height)` tuples, which tell us where on an imaginary x-axis a building begins and ends, and how tall it is. The skyline itself can be described by a list of `(x, height)` tuples, giving the locations at which the height visible to a distant observer changes, and each new height.

Given an array of buildings as described above, create a function that returns the skyline.

For example, suppose the input consists of the buildings `[(0, 15, 3), (4, 11, 5), (19, 23, 4)]`. In aggregate, these buildings would create a skyline that looks like the one below.

```
     ______
    |      |        ___
 ___|      |___    |   |
|   |   B  |   |   | C |
| A |      | A |   |   |
|   |      |   |   |   |
------------------------
```

As a result, your function should return `[(0, 3), (4, 5), (11, 3), (15, 0), (19, 4), (23, 0)]`.

Solution

Solution

By looking at the example above, we can form some general intuition about the problem. The second building is taller but narrower than the first. Therefore, when we get to the right side of `B`, the height of our skyline must change to that of `A`. What if there were a third building, `C`, that also overlapped with `A` and `B`? In this case we would have to inspect the boundaries between pairs of `A`, `B`, and `C`, to find potential height changes.

For each building we consider, then, we may have to compare its height with that of several others that overlap with it. Ideally, we should not have to check every other value in the list, as this would lead to an $O(N^2)$ runtime. Instead, we can leverage a max heap, keyed on each building's height. Essentially, the heap at any given time will hold the buildings that overlap with the current element, ordered by height.

The first step to implementing this solution is to sort the buildings by their left edges, breaking ties by placing the taller one first. For each building in the list, then, we do the following:

- Pop all elements from the heap whose right edges cannot overlap with the current building.
- Push the current building onto the heap.
- If the heap max has a different height from the skyline's last point, add `(left, heap_max_height)` to the solution.

There is a problem, however: we are not properly evaluating the ends of each building, so our solution will exclude points such as `(15, 0)` and `(23, 0)` above. To fix this, we can add zero-length "buildings" starting at each endpoint before we sort our array, which will force the algorithm to consider all relevant points.

```
import heapq

def create_skyline(buildings):
    buildings += [(r, r, 0) for (_, r, _) in buildings]
    buildings.sort(key=lambda x: (x[0], -x[2]))

    skyline = []
    heap = [(0, float("inf"))]


    for left, right, height in buildings:
        while heap and left >= heap[0][1]:
            heapq.heappop(heap)

        heapq.heappush(heap, (-height, right))

        if not skyline or skyline[-1][1] != -heap[0][0]:
            skyline.append((left, -heap[0][0]))

    return skyline
```

After adding these zero-length points, the list of buildings will contain `2N` elements, which will take `O(N log N)` time to sort. For each element in the list, we perform one `push` operation, and possibly several `pop` operations. However, since in total the number of elements that can be popped from or pushed to the heap is `2N`, there will be `O(N)` operations which each take `O(log N)` time, for an overall time complexity of `O(N log N)`.

Both the heap and solution array can have no more than `2N` elements, so the space required will be `O(N)`.