Daily Coding Problem #285

Problem

This problem was asked by Mailchimp.

You are given an array representing the heights of neighboring buildings on a city street, from east to west. The city assessor would like you to write an algorithm that returns how many of these buildings have a view of the setting sun, in order to properly value the street.

For example, given the array `[3, 7, 8, 3, 6, 1]`, you should return 3, since the top floors of the buildings with heights 8, 6, and 1 all have an unobstructed view to the west.

Can you do this using just one forward pass through the array?

Solution

This problem is simpler if we analyze the array from right to left. It must be the case that the last building in the array has a view of the sunset, since there is nothing blocking its path. Working our way backwards, each time we see an element with a greater value than our current maximum, this indicates another building with an unobstructed view of the sunset. When this happens, we reset the maximum and increment our solution counter.

```
def sunset_views(buildings):
    views = 0
    highest = 0
    buildings.reverse()


    for building in buildings:
```

```
        if building > highest:
            views += 1
            highest = building

    return views
```

This solution is straightforward and runs in `O(N)` time, where `N` is the number of buildings, and `O(1)` space.

However, if we must traverse the array from left to right, this solution is not allowed. Instead, we can use a stack. Suppose our stack, `views`, stores a list of buildings that we think have a view of the setting sun. At first this will be empty, and at later points it will be populated by values from our input.

The idea here is that whenever we come across a building that is taller than the last building in the stack, we pop elements from our stack until this is not the case, and then append the building. As a result, we will end up with a monotonically decreasing list of building heights, whose length will be our solution.

```
def sunset_views(buildings):
    views = []
    highest = 0

    for building in buildings:
        while views and views[-1] <= building:
            views.pop()
        views.append(building)

    return len(views)
```

At first it may seem that the time complexity of this algorithm should be greater than that of our first solution, because of the inner `while` loop. However, note that, at most, we can only pop and append `N` elements, so in fact this will run in `O(N)` time as well. In this case, though, we must use `O(N)` space to store the buildings in our stack.

Terms of Service

Press