Daily Coding Problem #30

Problem

This problem was asked by Facebook.

You are given an array of non-negative integers that represents a two-dimensional elevation map where each element is unit-width wall and the integer is the height. Suppose it will rain and all spots between two walls get filled up.

Compute how many units of water remain trapped on the map in O(N) time and O(1) space.

For example, given the input [2, 1, 2], we can hold 1 unit of water in the middle.

Given the input [3, 0, 1, 3, 0, 5], we can hold 3 units in the first index, 2 in the second, and 3 in the fourth index (we cannot hold 5 since it would run off to the left), so we can trap 8 units of water.

Solution

Notice that the amount of water that can be filled up at a certain index i is the smaller of the largest height to the left and the largest height to the right minus the actual value at that point, because it will be trapped by the smaller of the two sides. So what we can do is to create two arrays that represent the running maximum heights, one from the left and one from the right. Then to count the total capacity, we can run through the both arrays and add up the smaller of the two arrays at that index.

```
def capacity(arr):
    n = len(arr)
    left_maxes = [0 for _ in range(n)]
    right_maxes = [0 for _ in range(n)]

    current_left_max = 0
    for i in range(n):
        current_left_max = max(current_left_max, arr[i])
        left_maxes[i] = current_left_max

    current_right_max = 0
    for i in range(n - 1, -1, -1):
        current_right_max = max(current_right_max, arr[i])
        right_maxes[i] = current_right_max

    total = 0
    for i in range(n):
        total += min(left_maxes[i], right_maxes[i]) - arr[i]
    return total
```

This is O(N) time, but also O(N) space, and we want constant space. So instead, we can do this. We can find the largest element in the array, and then when we're looking on the left of it, we only need to keep the running total to the left (since we know the largest element on the array is on the right). And then do a similar thing, but starting from the right side. So the general gist is this:

- Find the maximum element in the array -- let's say it's at index i
- Initialize a running maximum on the left to arr[0]
- Iterate from index 1 to i. At each step, update the running maximum if necessary and then increment a variable counter with the running maximum minus the value at that array.
- Do the same thing but from len(arr) - 2 to i backwards, and keep the running maximum on the right.

```python
def capacity(arr):
    if not arr:
        return 0

    total = 0
    max_i = arr.index(max(arr))

    left_max = arr[0]
    for num in arr[1:max_i]:
        total += left_max - num
        left_max = max(left_max, num)

    right_max = arr[-1]
    for num in arr[-2:max_i:-1]:
        total += right_max - num
        right_max = max(right_max, num)

    return total
```