Daily Coding Problem #293

Problem

This problem was asked by Uber.

You have `N` stones in a row, and would like to create from them a pyramid. This pyramid should be constructed such that the height of each stone increases by one until reaching the tallest stone, after which the heights decrease by one. In addition, the start and end stones of the pyramid should each be one stone high.

You can change the height of any stone by paying a cost of `1` unit to lower its height by `1`, as many times as necessary. Given this information, determine the lowest cost method to produce this pyramid.

For example, given the stones `[1, 1, 3, 3, 2, 1]`, the optimal solution is to pay `2` to create `[0, 1, 2, 3, 2, 1]`.

Solution

To solve this problem, it will be useful to have a function that tells us what the cost will be to make a pyramid centered at a given location `x` with maximum height `h`.

This function will find out what height each stone should be, and subtract the ideal heights from the stones' current heights to find the total cost. If for some values of `x` and `h`, the ideal height is actually larger than the current height, there is no way for us to create this

pyramid, so we should return `False`.

```
def find_cost(x, h, stones):
    heights = []
    for i in range(len(stones)):
        heights.append(h - abs(x - i))

    costs = [s - h for (s, h) in zip(stones, heights)]
    if any(c < 0 for c in costs):
        return False

    return sum(costs)
```

With this function in place, one way to solve this problem is to loop over every possible centerpoint for our pyramid, and every possible maximum height, calling this function for each case. As we do this, we can keep track of the minimum cost seen so far, and return that in the end.

```
def build_pyramid(stones):
    min_cost = float('inf')

    for x in range(1, len(stones) - 1):
        for h in range(stones[x], 0, -1):
            cost = find_cost(x, h, stones)
            if cost is not False and cost < min_cost:
                min_cost = cost

    return min_cost
```

Our `find_cost` function takes `O(N)` time to run, where `N` is the number of stones. Since we must call this function for each stone and possible height, this solution has a time complexity of $O(N^2 * h)$, where `h` is the maximum value of any stone.

We can improve this solution by using dynamic programming, taking advantage of the facts that each stone can be at most one unit higher than either neighbor, and that we cannot make a stone taller than it already is.

In particular, we can make two passes through the input, one forward and one backward, to populate a `left` array and `right` array respectively. With each pass, we will set an array's value at a given index according to the following formula:

array's value at a given index according to the following formulas:

```
left[i] = min(current_value, left[i - 1] + 1)
right[i] = min(current_value, right[i + 1] + 1)
```

Since both of these lists provide upper bounds on the height of any stone, we can find the maximum height of each stone by taking the smaller of each list's value.

For example, suppose we are give the stones, `[1, 3, 3, 1]`. In our forward pass, we would create the array `[1, 2, 3, 1]`. Then, in our backward pass, we would create the array `[1, 3, 2, 1]`. Taking the minimum values, element-wise, would give us `[1, 2, 2, 1]`, meaning that the best pyramid we can make is one with maximum height `2`, centered at either the first or second index.

Finally, once we know where the center should be, and what the maximum height should be, we can call `find_cost` to determine the price of building this pyramid.

```
def build_pyramid(stones):
    left = [0 for _ in range(len(stones))]
    right = [0 for _ in range(len(stones))]

    left[0] = 1

    for i in range(1, len(stones)):
        left[i] = min(stones[i], left[i - 1] + 1)

    right[-1] = 1
    for i in range(len(stones) - 2, -1, -1):
        right[i] = min(stones[i], right[i + 1] + 1)

    min_heights = [min(l, r) for (l, r) in zip(left, right)]

    center = min_heights.index(max(min_heights))
    height = min_heights[center]
    cost = find_cost(center, height, stones)

    return cost
```

The left and right pass over our input will each take `O(N)` time, as will the call to `find_cost`, so this algorithm will take `O(N)` time overall. We will also require `O(N)` space to store the additional arrays.