A different approach to mono stack problems

1

123GJWEQ2 ★395 October 26, 2023 6:32 PM 339 VIEWS

Hello everyone. In this post I want to share an algorithmic technique that can be used on many monotonic stack problems. Although this technique is more intuitive (in my opinion) than the monotonic stack, it seems like fewer people know about it, so I want to hopefully inform some people about it with this post.

Let's look at the infamous problem Largest Rectangle in Histogram: https://leetcode.com/problems/largest-rectangle-in-histogram. When I first saw this problem in the neetcode 150, I found the stack solution super counter-intuitive. Luckily, there is a more intuitive technique that can be used here. I made a picture to help explain this technique: https://imgur.com/a/e22s05m

The intuition behind this is that, from the stack solution, we know that a rectangle in the histogram is bounded by the minimum height it contains. So, if we are at a certain minimum height h, we can only go to adjacent heights that are >= h. This is why we start at the largest height and go down from there: so we can have and readily use the intervals of all of the heights that are >= h. If h is adjacent to an interval or two, we merge them together and calculate the rectangle by (interval_end - interval_start + 1) * h. Otherwise, we create a new interval that starts at the index of h and ends at the index of h and our rectangle would just be h. Since we iterate through each height, we are guaranteed to get the maximum rectangle.

Code:

```
def largestRectangleArea(heights):
    start_to_end = {} #I use two dictionaries to store the interval information. This one maps the start of an interval to
    end_to_start = {} #This one maps the end of an interval to its start.
    val_to_inds = {} #This dictionary maps each height to its corresponding index/indices.
    res = 0

    for ind, val in enumerate(heights):
        if val not in val_to_inds:
            val_to_inds[val] = []
        val_to_inds[val].append(ind)

    for val in sorted(list(set(heights)), reverse = True): #iterate through the values of heights, largest to smallest

        for ind in val_to_inds[val]: #iterate through each index at the current value

            if ind + 1 in start_to_end and ind - 1 in end_to_start: #checks if our current index has an interval to its lef
                start, end = end_to_start[ind - 1], start_to_end[ind + 1] #creates new starting and ending indices for the
                end_to_start.pop(ind - 1) #gets rid of defunct keys
                start_to_end.pop(ind + 1) #gets rid of defunct keys

            elif ind + 1 in start_to_end: #checks if our current index only has an interval to its right
                start, end = ind, start_to_end[ind + 1] #creates new starting and ending indices for the merged interval
                start_to_end.pop(ind + 1) #gets rid of defunct keys

            elif ind - 1 in end_to_start: #checks if our current index only has an interval to its left
                start, end = end_to_start[ind - 1], ind #creates new starting and ending indices for the merged interval
                end_to_start.pop(ind - 1) #gets rid of defunct keys

            else: #otherwise our current index isn't near any intervals
                start, end = ind, ind #create a new interval containing only our current index

            #update our intervals
            start_to_end[start] = end
            end_to_start[end] = start

            res = max(res, (end - start + 1) * val) #update our result

    return res
```

Pros of this technique

- It is intuitive
- It is easy to remember and implement

Cons

- Time complexity is O(nlogn)/O(sort)

This technique, of course, isn't only useful on Largest Rectangle in Histogram. Another problem where it can be used is https://leetcode.com/problems/maximum-subarray-min-product/description/. All we would have to do is alter our code to use prefix sums.

Here is another mono-stack problem where this technique works: https://leetcode.com/problems/subarray-with-elements-greater-than-varying-threshold/description/

Sometimes, instead of iterating from the largest value to the smallest, we have to iterate from the smallest to the largest.

For example, consider the problem statement: You are given an Array of integers. Find the count of all the subarrays whose maximum is either its first or last element (this one isn't a leetcode problem; I found it on r/leetcode).

Let's call the array arr. We can see that, for each index i, arr[i] is a subarray that satisfies the conditions above. In order for arr[i] to be the maximum start/end of any more subarrays, it must be that arr[i - 1] <= arr[i] or arr[i + 1] <= arr[i]. This is the exact opposite of Largest Rectangle in Histogram, because we can only move to lower adjacent values rather than to higher adjacent values.

Code:

```
def maximum_element(arr):
    start_to_end = {}
    end_to_start = {}
    val_to_inds = {}
    res = 0

    for ind, val in enumerate(arr):
        if val not in val_to_inds:
            val_to_inds[val] = []
        val_to_inds[val].append(ind)


    #iterate from smallest to largest element
    for val in sorted(list(set(arr))):
        element_stack = 0
        last_end = val_to_inds[val][0]
        for ind in val_to_inds[val]:
            if ind + 1 in start_to_end and ind - 1 in end_to_start:
                start, end = end_to_start[ind - 1], start_to_end[ind + 1]
                end_to_start.pop(ind - 1)
                start_to_end.pop(ind + 1)
            elif ind + 1 in start_to_end:
                start, end = ind, start_to_end[ind + 1]
                start_to_end.pop(ind + 1)
            elif ind - 1 in end_to_start:
                start, end = end_to_start[ind - 1], ind
                end_to_start.pop(ind - 1)
            else:
                start, end = ind, ind
            start_to_end[start] = end
            end_to_start[end] = start

            res += end - start + 1

            #we have to do some extra counting for indices with the same value and in the same interval here
            if start > last_end:
                element_stack = 0
            res += element_stack * (end - ind)
            element_stack += 1
            last_end = end

    return res
```

I want to go over one more kind of stack problem where this technique can be used. Let's consider Daily Temperatures: https://leetcode.com/problems/daily-temperatures/description/

Now this problem is a little different in that we don't use the exact same algorithm or a slight variation of it. But we still use the intuition of iterating from the largest value to the smallest to get our answer.

If we start at the greatest temp, we know that 100% there is not a warmer day. We can add this temp's index to a SortedList. We move on to the second greatest temp. If there is an index in our SortedList that is larger than the index of our second greatest temp, then we know that there is a day with a greater temp and we can calculate our answer for our second greatest temp by subtracting the second greatest temp's index from the smallest index in our SortedList that is bigger than our second greatest temp's index. We add our second greatest temp index to our list and repeat this process for each temp until we finish.

Code

```
from sortedcontainers import SortedList
def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
    val_to_inds = {}
    greater_temps = SortedList()
    res_list = [0] * len(temperatures)

    for ind, val in enumerate(temperatures):
        if val not in val_to_inds:
            val_to_inds[val] = []
        val_to_inds[val].append(ind)

    for temp in sorted(list(set(temperatures)), reverse = True):
        for ind in val_to_inds[temp]:
            next_warmer_day = greater_temps.bisect_left(ind)
            if next_warmer_day < len(greater_temps):
                res_list[ind] = greater_temps[next_warmer_day] - ind
            greater_temps.add(ind)

    return res_list
```

Not the fastest algorithm, but still cool imo :)
If you want to, you can try this one: https://leetcode.com/problems/next-greater-element-iv/description/

And that is all! If you didn't know about this technique, I hope you learned something from this post.