739. Daily Temperatures

Medium Stack Array Monotonic Stack

Problem Description

The problem provides an array named `temperatures`, which represents the daily temperatures recorded. The goal is to find out how many days one would have to wait for a warmer temperature than the current day. To solve this problem, we need to construct an array called `answer` where `answer[i]` represents the number of days you would have to wait after the `i`-th day to experience a warmer temperature. If there is no such day in the future where the temperature is warmer, then `answer[i]` should be set to `0`.

For example, given `temperatures = [73, 74, 75, 71, 69, 72, 76, 73]`, the output should be `[1, 1, 4, 2, 1, 1, 0, 0]`. This means that on day 0 with a temperature of 73, you have to wait 1 day to get a temperature higher than 73, which occurs on day 1 with a temperature of 74. By the end of the array, the temperatures are not followed by any warmer temperatures, hence the `0`s.

Intuition

The intuition behind the solution is to use a stack that helps to track the temperatures and indexes. We traverse the temperatures from left to right, and for each day's temperature, we check whether it is higher than the temperature at the indexes recorded on the stack. If so, this means we have found a day with a warmer temperature for the days corresponding to those indexes. Therefore, for each such index, `j`, we can update `answer[j]` to the current day's index minus `j`, indicating the number of days that had to be waited.

The stack keeps track of indexes with temperatures that we haven't yet found a warmer day for. This is an effective approach because the temperatures are processed in order and the stack ensures we only compare temperatures where the future warmer temperature hasn't been found yet. When a warmer temperature is encountered, it is the immediate next warmer temperature for all temperatures currently in the stack. Once updated, we no longer need to consider those days because their next warmer temperature has been determined.

In cases where there are no warmer temperatures in the future, the answer will remain `0` by default, as established at the start of the solution.

Overall, the idea is to push the index of the current temperature to the stack if it cannot find a higher temperature immediately. Subsequently, with each new day's temperature, we compare it against the peak temperatures on the stack until we find one that is lower or until the stack is empty. This process efficiently calculates the number of days to wait for each day, leading to the final answer array when we have processed all temperatures.

Solution Approach

The implementation of the solution uses a stack data structure to keep track of the indices of days that have not yet found a warmer future temperature. The stack will help us maintain the order in which we need to find the next warmer temperature while iterating through the temperatures only once, which achieves a time complexity of O(n), where n is the number of days.

Here is a step-wise breakdown of the algorithm used in the solution:

1. Initialize an answer array `ans` with the same length as the `temperatures` array, filled with zeros. This array will hold the final number of days one has to wait for a warmer temperature.
2. Create an empty stack `stk` that will store indices of the `temperatures` array.
3. Iterate through the `temperatures` array using an index `i` and temperature `t`.
   - While there are indices on the stack and the current temperature `t` is greater than the temperature at the top index of the stack (i.e., `temperatures[stk[-1]] < t`), pop the index `j` from the top of the stack. This indicates that we have found a warmer day for the day at index `j`.
   - Calculate the number of days waited for index `j` by subtracting `j` from the current index `i` (i.e., `ans[j] = i - j`). This gives us the number of days that had to pass to reach a warmer temperature.
   - Continue popping from the stack and updating the `ans` array until the stack is empty or a day with a lower temperature is found.
4. If the current day's temperature isn't higher than the temperature at the top index of the stack, or if the stack is empty, push the current index `i` onto the stack. This signifies that we are still looking for a future warmer temperature for this day.
5. Once we exit the loop, we have filled out the `ans` array with the number of days to wait for a warmer temperature after each day. In cases where we do not find a warmer temperature, the default value of `0` remains.

This approach utilizes a monotonic stack in which, instead of storing the values, we store indices and ensure that the temperatures related to these indices are in ascending order. A monotonic stack is helpful when dealing with problems where we need to know the next greater or smaller element in an array.

The beauty of this algorithm lies in its efficient use of the stack to find the next greater element and its ability to accomplish this in a single pass through the `temperatures` array, hence having a linear time complexity relative to the input size.

Example Walkthrough

Let's follow the solution approach using a small example with the `temperatures` array `[71, 72, 70, 76, 69]`.

1. We start by initializing an `answer` array, `ans`, of the same length as `temperatures`, filled with zeros: `[0, 0, 0, 0, 0]`.
2. We also create an empty stack `stk` to keep track of indices where we haven't found a warmer temperature yet.
3. Now let's iterate through the `temperatures` array:
   - **Day 0**: `t = 71`
     - The stack `stk` is empty, so we push the index `0` onto `stk`.
   - **Day 1**: `t = 72`
     - The top index on `stk` is `0` and `temperatures[0] < 72`, so we pop `0` from `stk` and update `ans[0] = 1 - 0 = 1`. We then push the index `1` onto `stk`.
   - **Day 2**: `t = 70`
     - The top index on `stk` is `1` and `temperatures[1] > 70`, so we don't pop anything from `stk`. Push the index `2` onto `stk`.
   - **Day 3**: `t = 76`
     - The top index on `stk` is `2` and `temperatures[2] < 76`, so we pop `2` from `stk` and update `ans[2] = 3 - 2 = 1`.
     - Next, we check the new top index which is `1`. Since `temperatures[1] < 76`, we pop `1` from `stk` and update `ans[1] = 3 - 1 = 2`. There are no more indices on `stk`, so we push the current index `3` onto `stk`.
   - **Day 4**: `t = 69`
     - The top index on `stk` is `3` and `temperatures[3] > 69`, so we don't pop anything from `stk`. Push the index `4` onto `stk`.
4. After the iteration, we have the `ans` array updated as `[1, 2, 1, 0, 0]`, which means:
   - On Day 0, you have to wait 1 day to get a higher temperature on Day 1 (71 to 72).
   - On Day 1, you have to wait 2 days to get a higher temperature on Day 3 (72 to 76).
   - On Day 2, you have to wait 1 day to get a higher temperature on Day 3 (70 to 76).
   - Day 3 and Day 4 are followed by no warmer temperatures, so their values remain `0`.

This walkthrough has provided a step-by-step execution of the solution approach, showcasing how we use a stack to efficiently calculate the number of days one would have to wait for a warmer temperature.

Solution Implementation

Python

```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        # Initialize a list of zeros for the answer with the same length as the input list
        answer = [0] * len(temperatures)
        # Initialize an empty list to be used as a stack to keep track of temperatures indices
        stack = []

        # Enumerate over the list of temperatures
        for index, current_temp in enumerate(temperatures):
            # Loop through the stack as long as it's not empty and the current temperature
            # is greater than the temperature at the index of the last element in the stack
            while stack and temperatures[stack[-1]] < current_temp:
                # Pop the index of the temperature that is less than the current temperature
                previous_index = stack.pop()
                # Calculate the number of days between the previous and current temperature
                # and update the answer list
                answer[previous_index] = index - previous_index

            # Append the current index to the stack
            stack.append(index)

        # Return the answer list which contains the number of days to wait until a warmer temperature
        return answer
```

Java

```java
class Solution {
    // This method returns an array that contains the number of days you would
    // have to wait until a warmer temperature for each day represented in 'temperatures'.
    public int[] dailyTemperatures(int[] temperatures) {
        int n = temperatures.length; // Total number of days
        int[] result = new int[n]; // Initialize the result array with the same length as temperatures
        Deque<Integer> stack = new ArrayDeque<>(); // Use a stack to keep track of indices

        // Iterate through each day in temperatures
        for (int currentIndex = 0; currentIndex < n; ++currentIndex) {
            // While the stack is not empty and the current temperature is greater
            // than the temperature at the top index of the stack
            while (!stack.isEmpty() && temperatures[stack.peek()] < temperatures[currentIndex]) {
                int prevIndex = stack.pop(); // Get the index from the top of the stack
                result[prevIndex] = currentIndex - prevIndex; // Calculate the number of days and update result
            }
            // Push current index onto the stack
            stack.push(currentIndex);
        }
        // At the end, result array contains the answer
        return result;
    }
}
```

C++

```cpp
#include <vector>
#include <stack>
using namespace std;

class Solution {
public:
    // Function to find the number of days to wait for a warmer temperature for each day
    vector<int> dailyTemperatures(vector<int>& temperatures) {
        int n = temperatures.size(); // Number of days based on the temperature list
        vector<int> daysToWait(n); // Initialize the answer array with the same size as temperatures
        stack<int> indexStack; // Stack to keep track of temperatures indices

        // Iterate over each day in temperatures
        for (int i = 0; i < n; ++i) {
            // Check if the current day's temperature is higher than the temperature at the
            // top of the stack (which represents the last unprocessed day's temperature)
            while (!indexStack.empty() && temperatures[indexStack.top()] < temperatures[i]) {
                int previousDayIndex = indexStack.top(); // Get the index of the day with the lower temperature
                daysToWait[previousDayIndex] = i - previousDayIndex; // Calculate the days to wait
                indexStack.pop(); // Remove that day from the stack since it's now processed
            }

            // Always push the current day's index to the stack to process later
            indexStack.push(i);
        }

        // The stack will automatically contain 0s where there is no warmer temperature in the future
        return daysToWait;
    }
};
```

TypeScript

```typescript
// Function to calculate the number of days until a warmer temperature for each day.
function dailyTemperatures(temperatures: number[]): number[] {
    // Determine the total number of days in the temperatures array.
    const totalDays = temperatures.length;
    // Initialize an array to store the number of days to wait for a warmer temperature.
    const daysUntilWarmer = new Array(totalDays).fill(0);
    // Stack to keep track of indices of days which temperatures haven't been processed yet.
    const indexStack: number[] = [];

    // Iterate through the temperatures array in reverse order.
    for (let currentIndex = totalDays - 1; currentIndex >= 0; --currentIndex) {
        // While the stack is not empty and the current temperature is greater than or equal to
        // the temperature at the top index of the stack, pop the stack.
        while (indexStack.length && temperatures[indexStack[indexStack.length - 1]] <= temperatures[currentIndex]) {
            indexStack.pop();
        }
        // If the stack is not empty after the popping elements, compute the days to wait
        // for a warmer temperature by subtracting the current index from the top index in the stack.
        if (indexStack.length) {
            daysUntilWarmer[currentIndex] = indexStack[indexStack.length - 1] - currentIndex;
        }
        // Push the current index onto the stack.
        indexStack.push(currentIndex);
    }
    // Return the array of days to wait.
    return daysUntilWarmer;
}
```

```python
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        # Initialize a list of zeros for the answer with the same length as the input list
        answer = [0] * len(temperatures)
        # Initialize an empty list to be used as a stack to keep track of temperatures indices
        stack = []

        # Enumerate over the list of temperatures
        for index, current_temp in enumerate(temperatures):
            # Loop through the stack as long as it's not empty and the current temperature
            # is greater than the temperature at the index of the last element in the stack
            while stack and temperatures[stack[-1]] < current_temp:
                # Pop the index of the temperature that is less than the current temperature
                previous_index = stack.pop()
                # Calculate the number of days between the previous and current temperature
                # and update the answer list
                answer[previous_index] = index - previous_index

            # Append the current index to the stack
            stack.append(index)

        # Return the answer list which contains the number of days to wait until a warmer temperature
        return answer
```

Time and Space Complexity

Time Complexity

The time complexity of the given code is `O(N)`, where `N` is the number of days in the `temperatures` list. The reason for this is that each element is processed as it's pushed into the stack `stk` and then processed again when it's popped from the stack. Each element can be pushed and popped at most once which gives us a linear time complexity over the number of elements in the temperatures list.

Space Complexity

The space complexity of the given code is `O(N)` as well, where `N` is the number of days in the `temperatures` list. This is because we have an auxiliary stack `stk` that, in the worst case, might contain all temperature indices (`N`) at some point in time. Additionally, we have an array `ans` to store the answer for each day, which also contains `N` elements.