Daily Temperatures

Created	@2025年4月29日 下午12:39
Question Type	Stack
Difficulty	Medium
LeetCode Question Link	https://leetcode.com/problems/daily-temperatures/

1. Question Self-understanding:

1.1 Description:

Given a list of daily temperatures, determine for every day how many days one must wait until a warmer temperature appears.

1.2 Input:

A list of integers representing daily temperatures.

1.3 Input Assumption

- The list `temperatures` contains at least one element.
- Each temperature is an integer between 30 and 100 (inclusive).

1.4 Output:

The output is a list of integers. Each integer at position `k` indicates the number of days you must wait to encounter a warmer temperature than the one at position `k`. If there are no warmer temperatures after position `k`, the output should be `0`.

1.5 Example:

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

1.6 Other Q&A:

- **Question: What if no warmer temperature follows (e.g., the last day)?**
  - Answer: Set the corresponding output value to 0

2. Attempt 1:

2.1 Thought

- The key task in this problem is frequently checking whether the current day's temperature is higher than previous days' temperatures. To handle this efficiently, we need a container (data structure) to store previous temperatures that have not yet found a warmer day.
- Why do we only care about some of the previous temperatures? Because we only need to check temperatures that haven't found a warmer day yet.

Two important observations guide the choice of data structure:

1. We care about the difference in index positions, indicating how many days apart two temperatures are.
   - Hashmaps or sets are generally not suitable for tracking positions.

2. When we have a temperature at index `k`, we need to find the first future temperature that is warmer.
    - A stack is ideal here because it allows us to efficiently check and remove the most recent (top) temperature.
    - The stack naturally captures this relationship: newer temperatures are checked first against the most recent unprocessed temperature.

2.2 Pseudo-Code: (Ignore this part. It's a draft for brainstorming.)

```
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        # initialize the stack storage
        # intiialize the list to store result
        # put the first temperature value in to the stack
        # go through the list of temperatures
            # if the current temperature higher then the top of the stack
                # then we pop-up the list of temperature in the stack to compare cu
                # until the peak is higher then current temperature
                # Note: we need to store the result as well in this phase
            # Otherwise:
                # put the current value into the stack and continuous.
        # loop through the remaining stack, they will be assign all 0 where there is n
        # return the result
```

2.3 Implementation through python:

```
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        # initiallize the stack and the result list
```

```
    stack = []
    result = [0] * len(temperatures)

    # iterate through the temperatures list
    for i, current_temp in enumerate(temperatures):

        # check if the current value have a greater value than the last value in the

        while stack and current_temp > temperatures[stack[-1]]:

            # pop the last value
            last_index = stack.pop()

            # calculate the difference between the current index and the last index
            result[last_index] = i - last_index

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

    # loop through the stack and set the result to 0
    while stack:
        last_index = stack.pop()
        result[last_index] = 0
    return result
```

2.4 Time Complexity and Space Complexity

2.4.1 Time Complexity:

- Each temperature in the list is accessed at most twice—once when added to the stack and once when removed. Hence, the algorithm takes linear time, $O(2n) \in O(n)$, where n is the number of temperatures.

2.4.2 Space Complexity:

- The algorithm uses two main storage structures: the stack and the result list. Both could store up to n elements, resulting in a space complexity of $O(2n) \in O(n)$.