95 | 2

739. Daily Temperatures

Description (/problems/daily-temperatures/description/) Hints (/problems/daily-temperatures/hints/) Submissions (/problems/daily-temperatures/submissions/) Discuss (/problems/daily-temperatures/discuss/)

Quick Navigation ▾ View in Article (/articles/daily-temperatures/)

Notes

Approach #1: Next Array [Accepted]

Intuition

The problem statement asks us to find the next occurrence of a warmer temperature. Because temperatures can only be in `[30, 100]`, if the temperature right now is say, `T[i] = 50`, we only need to check for the next occurrence of `51`, `52`, ..., `100` and take the one that occurs soonest.

Algorithm

Let's process each `i` in reverse (decreasing order). At each `T[i]`, to know when the next occurrence of say, temperature `100` is, we should just remember the last one we've seen, `next[100]`.

Then, the first occurrence of a warmer value occurs at `warmer_index`, the minimum of `next[T[i]+1], next[T[i]+2], ..., next[100]`.

Java | Python | Copy

```java
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int[] ans = new int[T.length];
        int[] next = new int[101];
        Arrays.fill(next, Integer.MAX_VALUE);
        for (int i = T.length - 1; i >= 0; --i) {
            int warmer_index = Integer.MAX_VALUE;
            for (int t = T[i] + 1; t <= 100; ++t) {
                if (next[t] < warmer_index)
                    warmer_index = next[t];
            }
            if (warmer_index < Integer.MAX_VALUE)
                ans[i] = warmer_index - i;
            next[T[i]] = i;
        }
        return ans;
    }
}
```

Complexity Analysis

- Time Complexity: $O(NW)$, where N is the length of `T` and W is the number of allowed values for `T[i]`. Since $W = 71$, we can consider this complexity $O(N)$.
- Space Complexity: $O(N + W)$, the size of the answer and the next array.

Approach #2: Stack [Accepted]

Intuition

Consider trying to find the next warmer occurrence at `T[i]`. What information (about `T[j]` for `j > i`) must we remember?

Say we are trying to find `T[0]`. If we remembered `T[10] = 50`, knowing `T[20] = 50` wouldn't help us, as any `T[i]` that has its next warmer ocurrence at `T[20]` would have it at `T[10]` instead. However, `T[20] = 100` would help us, since if `T[0]` were `80`, then `T[20]` might be its next warmest occurrence, while `T[10]` couldn't.

Thus, we should remember a list of indices representing a strictly increasing list of temperatures. For example, `[10, 20, 30]` corresponding to temperatures `[50, 80, 100]`. When we get a new temperature like `T[i] = 90`, we will have `[5, 30]` as our list of indices (corresponding to temperatures `[90, 100]`). The most basic structure that will satisfy our requirements is a *stack*, where the top of the stack is the first value in the list, and so on.

Algorithm

As in *Approach #1*, process indices `i` in descending order. We'll keep a `stack` of indices such that `T[stack[-1]] < T[stack[-2]] < ...`, where `stack[-1]` is the top of the stack, `stack[-2]` is second from the top, and so on; and where `stack[-1] > stack[-2] > ...`; and we will maintain this invariant as we process each temperature.

After, it is easy to know the next occurrence of a warmer temperature: it's simply the top index in the stack.

Here is a worked example of the contents of the `stack` as we work through `T = [73, 74, 75, 71, 69, 72, 76, 73]` in reverse order, at the end of the loop (after we add `T[i]`). For clarity, `stack` only contains indices `i`, but we will write the value of `T[i]` beside it in brackets, such as `0 (73)`.

Notes

- When `i = 7`, `stack = [7 (73)]`. `ans[i] = 0`.
- When `i = 6`, `stack = [6 (76)]`. `ans[i] = 0`.
- When `i = 5`, `stack = [5 (72), 6 (76)]`. `ans[i] = 1`.
- When `i = 4`, `stack = [4 (69), 5 (72), 6 (76)]`. `ans[i] = 1`.
- When `i = 3`, `stack = [3 (71), 5 (72), 6 (76)]`. `ans[i] = 2`.
- When `i = 2`, `stack = [2 (75), 6 (76)]`. `ans[i] = 4`.
- When `i = 1`, `stack = [1 (74), 2 (75), 6 (76)]`. `ans[i] = 1`.
- When `i = 0`, `stack = [0 (73), 1 (74), 2 (75), 6 (76)]`. `ans[i] = 1`.

Java Python Copy

```java
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int[] ans = new int[T.length];
        Stack<Integer> stack = new Stack();
        for (int i = T.length - 1; i >= 0; --i) {
            while (!stack.isEmpty() && T[i] >= T[stack.peek()]) stack.pop();
            ans[i] = stack.isEmpty() ? 0 : stack.peek() - i;
            stack.push(i);
        }
        return ans;
    }
}
```

Complexity Analysis

- Time Complexity: $O(N)$, where N is the length of `T` and W is the number of allowed values for `T[i]`. Each index gets pushed and popped at most once from the stack.
- Space Complexity: $O(W)$. The size of the stack is bounded as it represents strictly increasing temperatures.

Analysis written by: @awice (https://leetcode.com/awice).

Join the conversation

Signed in as **soleil-guang-gmail-com**.

Post a Reply

F

FreeTymeKiyan commented 5 days ago

(https://discuss.leetcode.com/user/freetymekiyan)

Don't see why we do it backwards. Unnecessary IMHO.

```java
int[] result = new int[temperatures.length];
Stack<Integer> stack = new Stack<>(); // Make it a stack of indices.
for (int i = 0; i < temperatures.length; i++) {
    while (!stack.isEmpty() && temperatures[i] > temperatures[stack.peek()]) {
        int index = stack.pop();
        result[index] = i - index;
    }
    stack.push(i);
}
return result;
```