Module 34 lecture 1

04 March 2025 23:33

Number of Visible People in a Queue - Detailed Explanation & Solution

Problem Statement:

Given an array heights of distinct integers where heights[i] represents the height of the ith person in a queue (standing from left to right), we need to determine how many people each person can see to their right.
A person i can see another person j (where i < j) if:

- min(heights[i], heights[j]) > max(heights[i+1], ..., heights[j-1])

Approach:

To solve this efficiently, we use a **monotonic decreasing stack**. The stack helps us efficiently determine which people are visible based on their heights while iterating the array in reverse order.

Steps to Solve:

1. **Initialize Data Structures**:
   - A vector ans to store the final answer.
   - A stack st to keep track of indices of heights in decreasing order.
2. **Start Iterating from Right to Left**:
   - Begin from the second-last person (n-2 index) to the first (0th index), because the last person cannot see anyone ahead.
3. **Use Stack to Track Visibility**:
   - Keep a count variable to count how many people a person can see.
   - If the current person is taller than the person at the top of the stack, they will see all shorter people (so pop them from the stack and increase the count).
   - If there is still a taller person in the stack, they will be visible too (increment count).
4. **Store the Result and Push Current Person to Stack**:
   - Store count in ans[i].
   - Push the current person's index to the stack so future people can check visibility.

Code Implementation:

```cpp
class Solution {
public:
    vector<int> canSeePersonsCount(vector<int>& arr) {
        int n = arr.size();
        vector<int> ans(n);
        stack<int> st;
        st.push(n-1);
        ans[n-1] = 0;
        for(int i = n-2; i >= 0; i--) {
            int count = 0;
            while (st.size() > 0 && arr[i] > arr[st.top()]) {
                count++;
                st.pop();
            }
            if (st.size() > 0) {
                count++;
                ans[i] = count;
            }
            else if (st.size() == 0) ans[i] = count;
            st.push(i);
        }
        return ans;
    }
};
```

Explanation of Code Execution:

1. **Initialize variables**:
   - ans vector of size n.
   - stack to maintain a decreasing order of heights.
2. **Rightmost person sees 0 people** (Base Case):
   - ans[n-1] = 0
   - Push n-1 index onto the stack.
3. **Iterate from n-2 to 0**:
   - **While loop** pops all shorter people and counts them.
   - If stack still has elements, one more person is visible.
   - Store the count and push the current person to the stack.

Dry Run Example:

Input: heights = [10, 6, 8, 5, 11, 9]

Execution Steps:

Index	Height	Stack Before	Visible Count	Stack After
5	9	[]	0	[5]
4	11	[5]	1	[4]
3	5	[4]	1	[4,3]
2	8	[4,3]	2	[4,2]
1	6	[4,2]	1	[4,2,1]
0	10	[4,2,1]	4	[4,0]

Output: [4,1,2,1,1,0]

Complexity Analysis:

- **Time Complexity:** O(N)
  - Each element is pushed to the stack once and popped at most once → **O(N) overall**.
- **Space Complexity:** O(N) (for stack and answer array).

Key Takeaways:

- **Monotonic decreasing stack** helps efficiently track visibility conditions.
- **Reverse traversal** ensures we correctly process visibility from right to left.
- **Each element is processed only once**, making the approach optimal.

This method efficiently solves the problem while keeping the code simple and easy to understand!