Day 136/180 Stack Variation

1: The Celebrity Problem

```
int celebrity(vector<vector<int>>& M, int n) {
    int celeb = 0; // Initialize the potential celebrity index to the first person
(0-indexed)
    // Iterate through all other persons
    for (int i = 1; i < n; i++) {
        // If the current potential celebrity knows the next person (i.e., M[celeb][i] == 1),
       // update the potential celebrity index to the next person
        if (M[celeb][i] == 1) {
            celeb = i;
        }
    }
    int ct = 0; // Counter to track the number of connections of the potential celebrity
   // Check all persons to confirm whether the potential celebrity satisfies the conditions
    for (int j = 0; j < n; j++) {
        // Check if the potential celebrity knows the current person
        if (M[j][celeb] == 1) {
            ct++; // Increment the counter
        }
        // Check if the current person knows the potential celebrity
        if (M[celeb][j] == 1) {
            ct++; // Increment the counter
        }
    }

    // If the potential celebrity has connections with all other persons except themselves,
   // then return the index of the potential celebrity
    if (ct == n - 1) {
        return celeb;
    }

    // Otherwise, return -1 indicating that there is no celebrity in the group
    return -1;
}
```

2: Rain Trapping Water (Solve it with the help of Stack)

```cpp
class Solution {
public:
    int trap(vector<int>& a) {
        int ans = 0; // Variable to store the total trapped water
        int n = a.size(); // Total number of elements in the input array
        stack<int> st; // Stack to store the indices of elements

        // Traverse through each element of the input array
        for(int i = 0; i < n; i++) {
            // While the stack is not empty and the current element is greater than the element at the top of the stack
            while (!st.empty() && a[st.top()] < a[i]) {
                int cur = st.top(); // Index of the current element (top of the stack)
                st.pop(); // Pop the index from the stack

                // If the stack becomes empty after popping, break the loop
                if (st.empty()) break;

                int left = st.top(); // Index of the left boundary of the trapped water (element before the current top element)
                int width = i - left - 1; // Width of the trapped water (distance between the current element and the left boundary)
                int height = min(a[left], a[i]) - a[cur]; // Height of the trapped water (difference in heights between the lower boundary and the current element)
                ans += height * width; // Add the trapped water to the total
            }
            st.push(i); // Push the current index onto the stack
        }
        return ans; // Return the total trapped water
    }
};
```

3: MAXSPPROD

```cpp
#define ll long long // Define a macro for long long data type

// Definition of the function maxSpecialProduct
int Solution::maxSpecialProduct(vector<int> &nums) {
    int n = nums.size(); // Number of elements in the input vector

    vector<ll> left(n), right(n); // Vectors to store left and right special
products for each element

    left[0] = 0; // Special product for the first element is set to 0
    right[n - 1] = 0; // Special product for the last element is set to 0

    long long mod = 1e9 + 7; // Define a constant for modulo operation

    stack<ll> stk; // Stack to store indices

    // Calculate left special products for each element
    stk.push(0);
    for (int i = 1; i < n; i++) {
        while (!stk.empty() && nums[i] >= nums[stk.top()]) {
            stk.pop();
        }
        if (stk.empty()) {
            left[i] = 0;
        } else {
            left[i] = stk.top() % mod;
        }
        stk.push(i);
    }

    // Clear the stack for further use
    while (!stk.empty()) {
        stk.pop();
    }

    // Calculate right special products for each element
    stk.push(n - 1);
```

```
    for (int i = n - 2; i >= 0; i--) {
        while (!stk.empty() && nums[i] >= nums[stk.top()]) {
            stk.pop();
        }
        if (stk.empty()) {
            right[i] = 0;
        } else {
            right[i] = stk.top() % mod;
        }
        stk.push(i);
    }

    ll max_ans = 0; // Variable to store the maximum special product

    // Iterate through each element to calculate the maximum special product
    for (int i = 0; i < n; i++) {
        // Calculate the product of left and right special products for the current
element
        long long val = (left[i] * 1LL * right[i]);

        // Update max_ans with the maximum of current value and max_ans
        max_ans = max(max_ans, val) % mod;
    }

    // Return the maximum special product modulo 1000000007
    return max_ans % 1000000007;
}
```