Diffuse

solved by	Senan
Platform	LeetCode
difficulty	Easy
Serial	1652
tags	Sliding Window
language	C++
solved on	@18/11/2024
link	https://leetcode.com/problems/defuse-the-bomb/
Completion	✓

Intuition

The problem involves decrypting a circular array based on the value of `k`. The decryption process requires summing up elements either to the left or the right of the current index.

- If `k > 0`, sum the next `k` elements for each index.
- If `k < 0`, sum the previous `|k|` elements for each index.
- If `k == 0`, all decrypted values are `0`.

The sliding window technique is optimal here, allowing us to efficiently compute the sums while accounting for the circular nature of the array.

Approach

1. **Base Case**:

   If `k == 0`, simply return a vector of zeros.
2. **Sliding Window for `k > 0`**:
   - Compute the sum of the first `k` elements to initialize the sliding window.
   - For each index `i`, store the current window sum in the answer, then adjust the window by adding the next element and removing the current element.
   - Use modulo arithmetic to handle the circular nature of the array.
3. **Sliding Window for `k < 0`**:
   - Work with the absolute value of `k` (`|k|`).
   - Compute the sum of the last `|k|` elements to initialize the sliding window.
   - For each index `i`, store the current window sum in the answer, then adjust the window by adding the previous element and removing the current one.
   - Use modular arithmetic to ensure indices wrap around correctly.

Complexity

Time Complexity:

- **Initialization of `sum`**: **O(|k|)** for the `accumulate` function.
- **Sliding Window Iteration**: **O(n)**, where **n** is the size of the array.

- Total: **O(n + |k|)**.
  Since
  k is bounded by **n**, the worst-case time complexity is **O(n)**.

Space Complexity:

- The space complexity is **O(n)** for the `answer` vector.

Code

```
class Solution {
public:
    vector<int> decrypt(vector<int>& code, int k) {
        int n = code.size();
        vector<int> answer(n, 0);

        if (k == 0) return answer;

        if (k > 0) {
            int sum = accumulate(code.begin(), code.begin() + k, 0);
            for (int i = 0; i < n; i++) {
                answer[i] = sum;
                sum += code[(i + k) % n] - code[i];
            }
        }
        else {
            k = -k;
            int sum = accumulate(code.end() - k, code.end(), 0);
            for (int i = 0; i < n; i++) {
                answer[i] = sum;
                sum += code[i] - code[(i - k + n) % n];
            }
        }

        return answer;
    }
};
```