05.01.2025

Problem: Shifting Letters II (LeetCode 2381)

Problem Statement:
You are given:

- A string `s` consisting of lowercase English letters.
- A 2D integer array `shifts` where `shifts[i] = [start_i, end_i, direction_i]`.

For every `i`:

- Shift the characters from index `start_i` to `end_i` (inclusive) **forward** if `direction_i = 1`.
- Shift them **backward** if `direction_i = 0`.

Forward Shift: Replace a letter with the next letter (wrapping around 'z' → 'a').
Backward Shift: Replace a letter with the previous letter (wrapping around 'a' → 'z').

Example 1:

```
Input: s = "abc", shifts = [[0,1,0],[1,2,1],[0,2,1]]
Output: "ace"
Explanation:
1. Shift characters from index 0 to 1 backward → "zac".
2. Shift characters from index 1 to 2 forward → "zbd".
3. Shift characters from index 0 to 2 forward → "ace".
```

Example 2:

```
Input: s = "dztz", shifts = [[0,0,0],[1,1,1]]
Output: "catz"
Explanation:
1. Shift characters from index 0 to 0 backward → "cztz".
2. Shift characters from index 1 to 1 forward → "catz".
```

Approach: Difference Array + Prefix Sum

Key Concept:

- Use a **difference array** to efficiently apply multiple range updates.
- The difference array helps track increments and decrements for ranges.

Steps:

1. **Initialize Difference Array:**
   a. Create a `differenceArray` of size `n` with all zeroes.
2. **Populate the Difference Array:**
   a. For each shift `[start, end, direction]`:
      i. If `direction = 1` → Add `+1` at `start` and `-1` at `end + 1`.
      ii. If `direction = 0` → Add `-1` at `start` and `+1` at `end + 1`.
3. **Compute Cumulative Sum:**
   a. Convert the difference array into the final effect on each position using a **prefix sum**.
4. **Apply the Changes to the String:**
   a. Apply the final shift to each character using modular arithmetic (`% 26`).

Code:

```cpp
class Solution {
public:
    string shiftingLetters(string s, vector<vector<int>>& shifts) {
        int n = s.size();
        vector<int> differenceArray(n, 0);

        // Step 1: Populate difference array
        for (auto& shift : shifts) {
            int start = shift[0];
            int end = shift[1];
            int direction = shift[2];
            int change = (direction == 1) ? 1 : -1;

            differenceArray[start] += change;
            if (end + 1 < n) {
                differenceArray[end + 1] -= change;
            }
        }

        // Step 2: Compute cumulative sum
        for (int i = 1; i < n; ++i) {
            differenceArray[i] += differenceArray[i - 1];
        }

        // Step 3: Apply shifts to the string
        for (int i = 0; i < n; ++i) {
            int shiftValue = differenceArray[i] % 26;
            if (shiftValue < 0) shiftValue += 26;  // Handling negative shifts
            s[i] = (s[i] - 'a' + shiftValue) % 26 + 'a';
        }

        return s;
```

```
        }
    };
```

Explanation:

For `s = "abc"` and `shifts = [[0,1,0],[1,2,1],[0,2,1]]`:

- **Step 1:** Initialize `differenceArray = [0, 0, 0]`.
- **Step 2:** Apply the shifts:
  a. `[0,1,0] → [-1, -1, +1]` → Difference Array: `[-1, -1, +1]`.
  b. `[1,2,1] → [+1, +1, -1]` → Difference Array: `[-1, 0, 0]`.
  c. `[0,2,1] → [+1, +1, -1]` → Difference Array: `[0, 1, -1]`.
- **Step 3:** Prefix Sum: `[0, 1, 0]`.
- **Step 4:** Apply shifts:
  a. `s[0] = 'a' → a + 0 = a`.
  b. `s[1] = 'b' → b + 1 = c`.
  c. `s[2] = 'c' → c + 0 = c`.

Final Output: `"ace"`

Complexity:

- **Time Complexity:** `O(n + m)`
  a. `n` for processing the string.
  b. `m` for processing the shifts.
- **Space Complexity:** `O(n)` (for the difference array).

Edge Cases Handled:

- If `s.length == 1` → No shifts possible.
- If all shifts are empty → Return the original string.

Key Takeaways:

- The **difference array** technique is optimal for multiple range updates.
- Always handle negative shifts using modular arithmetic carefully.
- Efficient for competitive programming due to `O(n)` complexity.