2381. Shifting Letters II

You are given a string `s` of lowercase English letters and a 2D integer array `shifts` where `shifts[i] = [start`$_i$`, end`$_i$`, direction`$_i$`]`. For every `i`, **shift** the characters in `s` from the index `start`$_i$ to the index `end`$_i$ (**inclusive**) forward if `direction`$_i$` = 1`, or shift the characters backward if `direction`$_i$` = 0`.

Shifting a character **forward** means replacing it with the **next** letter in the alphabet (wrapping around so that `'z'` becomes `'a'`). Similarly, shifting a character **backward** means replacing it with the **previous** letter in the alphabet (wrapping around so that `'a'` becomes `'z'`).

Return *the final string after all such shifts to* `s` *are applied*.

Example 1:

```
Input: s = "abc", shifts = [[0,1,0],[1,2,1],[0,2,1]]
Output: "ace"
Explanation: Firstly, shift the characters from index 0 to index 1 backward. Now s
= "zac".
Secondly, shift the characters from index 1 to index 2 forward. Now s = "zbd".
Finally, shift the characters from index 0 to index 2 forward. Now s = "ace".
```

Example 2:

```
Input: s = "dztz", shifts = [[0,0,0],[1,1,1]]
Output: "catz"
Explanation: Firstly, shift the characters from index 0 to index 0 backward. Now s
= "cztz".
Finally, shift the characters from index 1 to index 1 forward. Now s = "catz".
```

Constraints:

- `1 <= s.length, shifts.length <=` $5*10^4$
- `shifts[i].length == 3`
- `0 <= start`$_i$` <= end`$_i$` < s.length`
- `0 <= direction`$_i$` <= 1`
- `s` consists of lowercase English letters.

2381. Shifting Letters II

```
/*
   Difference array + Prefix sum
   Time compelxity:O(Q+Q+m+min(n,m)), in WC m=n => O(2(Q+n))
   Space compelexity:O(2.m), in WC m=n => O(2n)
   n: length of given s
   Q: length of given array shifts
   m: max right side in shifts' ranges
*/
typedef std::vector<int> vi;
typedef std::vector<vi> vvi;
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution{
   public:
      std::string shiftingLetters(std::string s, vvi& shifts){
         int n=s.size();

         // Get max right side in shifts' ranges
         vi up=*std::max_element(shifts.begin(),shifts.end(),[](vi& a,vi& b){return a[1]<b[1];});
         int m=up[1]+2;
```

```
// Preprocess ranges
// Compute in each range, the number of zeros and the number of ones
// This helps us later to know the final shift on each letter
vii pre(m+2,{0,0});

auto preprocess=[&]()->void{
    for(auto shift: shifts){
        int l=shift[0];
        int r=shift[1];
        int d=shift[2];

        // Using difference array technique to:
        // Count the number of zeros and ones
        pre[l].first+=int(d==0);
        pre[l].second+=int(d==1);

        pre[r+1].first-=int(d==0);
        pre[r+1].second-=int(d==1);

    }

    // Cumulate the counts using the previous computations
    for(int i=1;i<m;++i){
        pre[i].first+=pre[i-1].first;
        pre[i].second+=pre[i-1].second;
    }
};

preprocess();

// Deal with negative dividend
auto mod=[](int a,int b)->int{
    return ((a%b)+b)%b;
};
```

```
        // For each letter in the given string
        std::string ans=s;
        for(int i=0;i<std::min(n,m);++i){
            // Compute the final shift
            auto [zeros_count,ones_count]=pre[i];
            int k=ones_count-zeros_count;

            // Get its position in the alphabet
            int pos=ans[i]-'a';

            // Perform the shifton it
            int shift=mod(pos+k,26);

            // Modify it
            ans[i]=char(shift+'a');
        }

        return ans;
    }
};
```