Prefix Sums, TLE Eliminators Lect - 1.

Prefix Sums

Definition

- Prefix sum is a preprocessing technique to enable fast subarray sum queries without modifying the original array.
- **Formula**: `Prefix[i] = Sum of array elements from index 0 to i`.

Example:

```
// Array
int arr[] = {1, 2, 9, -1, -2, 3};
// Prefix Array
int prefix[] = {1, 3, 12, 11, 9, 12};
```

Implementation of 1D Prefix Sums

Brute Force Approach

- **Time Complexity**: .

$$O(N2)O(N^2)$$

- **Logic**: Recompute the sum for every query.

```
for (int l = 0; l < n; l++) {
    for (int r = l; r < n; r++) {
        int sum = 0;
        for (int k = l; k <= r; k++) sum += arr[k];
    }
}
```

Optimized Approach

- **Time Complexity**: (Precomputation: , Queries:).

$$O(N+Q)O(N+Q)$$

$$O(N)O(N)$$

$$O(1)O(1)$$

- **Logic**: Use precomputed prefix array.
- **Formula**:

$$Sum\ from\ L\ to\ R = Prefix[R] - Prefix[L-1]\text{Sum from L to R} = \text{Prefix}[R] - \text{Prefix}[L-1]$$

Code:

```
int prefix[n];
prefix[0] = arr[0];
for (int i = 1; i < n; i++) {
    prefix[i] = prefix[i - 1] + arr[i];
}
// Query Example
int l = 2, r = 4;
int result = prefix[r] - prefix[l - 1];
```

2D Prefix Sums

Definition

An extension of 1D prefix sums to 2D grids.

Formula:

$$P[i][j] = Sum\ of\ all\ grid[x][y]\ such\ that\ x \leq i\ and\ y \leq j P[i][j] = \text{Sum of all grid}[\text{x}][\text{y}] \text{ such that } x \leq i \text{ and } y \leq j$$

Example Grid:

```
int grid[5][5] = {
    {2, 6, 5, 9, 10},
    {3, 4, 1, 6, 1},
    {2, 7, 3, 2, 2},
    {1, 3, 5, 4, 3},
    {6, 7, 9, 11, 4}
};
```

Constructing P[i][j]P[i][j]

- **Time Complexity**: .

$$O(N \times M) O(N \times M)$$

- **Formula**:

$$P[i][j] = grid[i][j] + P[i-1][j] + P[i][j-1] - P[i-1][j-1] P[i][j] = \text{grid}[i][j] + P[i-1][j] + P[i][j-1] - P$$

Code:

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        prefix[i][j] = grid[i][j];
        if (i > 0) prefix[i][j] += prefix[i - 1][j];
        if (j > 0) prefix[i][j] += prefix[i][j - 1];
        if (i > 0 && j > 0) prefix[i][j] -= prefix[i - 1][j - 1];
    }
}
```

Querying Submatrices

- **Time Complexity**: .

$$O(1)O(1)$$

- **Logic**: Sum of submatrix from top-left to bottom-right :

$$(r1, c1)(r1, c1)$$

$$(r2, c2)(r2, c2)$$

$$Sum = P[r2][c2] - P[r1-1][c2] - P[r2][c1-1] + P[r1-1][c1-1]\text{Sum} = P[r2][c2] - P[r1-1][c2] - P[r2][c$$

Code:

```
int sum = prefix[r2][c2];
if (r1 > 0) sum -= prefix[r1 - 1][c2];
if (c1 > 0) sum -= prefix[r2][c1 - 1];
if (r1 > 0 && c1 > 0) sum += prefix[r1 - 1][c1 - 1];
```

Bonus: 2D XOR Queries

Problem

Find XOR of elements in a submatrix.

Formula:

$$XOR = X[r2][c2] \oplus X[r1-1][c2] \oplus X[r2][c1-1] \oplus X[r1-1][c1-1]\text{XOR} = X[r2][c2] \oplus X[r1-1][c2] \oplus X[r2][c$$

Code:

```
int xor_sum = xor_prefix[r2][c2];
if (r1 > 0) xor_sum ^= xor_prefix[r1 - 1][c2];
if (c1 > 0) xor_sum ^= xor_prefix[r2][c1 - 1];
if (r1 > 0 && c1 > 0) xor_sum ^= xor_prefix[r1 - 1][c1 - 1];
```
