Bài 8. Prefix Prefix

📌 Bài toán:

> Bạn được cho một mảng số nguyên `A[]` gồm `N` phần tử và `M` thao tác. Mỗi thao tác sẽ cộng thêm `D` đơn vị vào một đoạn `[L, R]` trong mảng `A`. Ngoài ra, bạn còn có `K` truy vấn, mỗi truy vấn yêu cầu thực hiện hàng loạt thao tác đánh số từ `x` đến `y` (trong số `M` thao tác ban đầu).
>
> Nhiệm vụ của bạn là áp dụng toàn bộ các thao tác được yêu cầu trong `K` truy vấn và in ra mảng `A[]` sau cùng.

🎯 Mục tiêu:

Tính toán chính xác mảng `A[]` sau khi áp dụng toàn bộ các thao tác tương ứng với `K` truy vấn.

📘 Ví dụ

Input:

```
6 5 2
1 7 5 2 8 7
2 6 5
1 5 1
1 6 1
2 6 4
2 5 4
2 5
2 4
```

Output:

```
7 38 36 33 39 27
```

💡 Ý tưởng giải:

Bước 1: Xác định tần suất của từng thao tác

- Khởi tạo mảng hiệu `opFreq[]` có kích thước `M + 2`, dùng để lưu số lần **mỗi thao tác** được yêu cầu.
- Với mỗi truy vấn `[x, y]`, ta cập nhật:

  ```
  opFreq[x] += 1
  opFreq[y + 1] -= 1
  ```

- Sau đó cộng dồn `opFreq[]` để biết mỗi thao tác được thực hiện bao nhiêu lần.

Bước 2: Gộp hiệu ứng các thao tác lên mảng A bằng mảng hiệu

- Dùng mảng hiệu `diff[]` kích thước `N + 2`, ban đầu là 0.

- Với mỗi thao tác `(L, R, D)` được dùng `count` lần, ta cập nhật:

```
diff[L] += count * D
diff[R + 1] -= count * D
```

Bước 3: Áp dụng mảng hiệu để cập nhật A

- Tính `prefix sum` trên `diff[]` và cộng trực tiếp vào `A[i]`.

🧠 Độ phức tạp:

- **Thời gian**: `O(N + M + K)`
- **Không gian**: `O(N + M)`
- Thuật toán đảm bảo chạy tốt với `N, M, K ≤ 10^5`.

✅ Kết quả đầu ra:

Mảng `A[]` sau khi thực hiện xong tất cả các thao tác được yêu cầu trong `K` truy vấn.

📌 Ghi chú:

Đây là bài toán ứng dụng 2 lớp **mảng hiệu (difference array)**:

- Lớp ngoài: Ghi nhận số lần thực hiện mỗi thao tác.
- Lớp trong: Ghi nhận ảnh hưởng của thao tác đến mảng A.

Kỹ thuật này giúp ta xử lý hàng trăm nghìn truy vấn và thao tác trong thời gian tối ưu.