⚡ Técnicas de Sumas Parciales y Aplicaciones en Intervalos

📜 Introducción a las Sumas Parciales (Prefix Sums)

Dado un arreglo `arr` de tamaño `N`, muchas veces necesitamos responder consultas que piden la **suma de los elementos entre dos posiciones** `a` y `b` (inclusive).

Problema simple

> Calcular sumas de subarreglos rápidamente para `Q` consultas.

¿Por qué no hacerlo directo?

- Para cada consulta, sumar desde `a` hasta `b` es O(N).
- Con muchas consultas, la complejidad total sería O(NQ), inviable si `N` y `Q` son grandes.

💡 Idea Clave

Dado un arreglo `arr[]`, encuentra la suma parcial (prefix sum) del arreglo.
Un arreglo de suma parcial `prefixSum[]` es otro arreglo del mismo tamaño, tal que `prefixSum[i]` es la suma de `arr[0] + arr[1] + arr[2] + ... + arr[i]`.

Ejemplo

- **Entrada:** `arr[] = [10, 20, 10, 5, 15]`
  Salida: `[10, 30, 40, 45, 60]`
  Explicación:
  `prefixSum[0] = 10`
  `prefixSum[1] = 10 + 20 = 30`
  `prefixSum[2] = 10 + 20 + 10 = 40`
  Y así sucesivamente.

🧮 Teoría y Uso de Prefix Sums

Si queremos calcular la suma de elementos en un rango `[a, b]` (con índices 0-indexados), usando el arreglo de sumas parciales podemos hacerlo en tiempo constante `O(1)`:

$$ \sum_{i=a}^{b} arr[i] = prefixSum[b] - prefixSum[a-1] $$

(con cuidado cuando `a = 0`, en ese caso simplemente es `prefixSum[b]`).

💻 Código en Python para calcular Prefix Sums

```python
def prefix_sums(arr):
    """Calcula los prefix sums de un arreglo."""
    prefixSum = [0] * len(arr)
```

```
    prefixSum[0] = arr[0]
    for i in range(1, len(arr)):
        prefixSum[i] = prefixSum[i - 1] + arr[i]
    return prefixSum


def range_sum(prefixSum, a, b):
    """Devuelve la suma de los elementos entre los índices a y b (inclusive)."""
    if a == 0:
        return prefixSum[b]
    return prefixSum[b] - prefixSum[a - 1]


# Ejemplo de uso
arr = [10, 20, 10, 5, 15]
prefix = prefix_sums(arr)
print("Prefix sums:", prefix)           # Output: [10, 30, 40, 45, 60]
print("Suma de [1, 3]:", range_sum(prefix, 1, 3))  # Output: 35 (20 + 10 + 5)
print("Suma de [0, 4]:", range_sum(prefix, 0, 4))  # Output: 60 (10 + 20 + 10 + 5
+ 15)
```

➕ Arreglos de Diferencias (Difference Arrays)

Una técnica muy útil para **actualizar rangos de forma eficiente**.

Imagina que necesitas realizar muchas operaciones del tipo "**sumar un valor `x` a todos los elementos entre las posiciones `[l, r]`**" antes de conocer el resultado final del arreglo.
Si haces esto directamente, cada operación costaría tiempo proporcional al tamaño del rango.
El **arreglo de diferencias** permite hacerlo en **tiempo constante por operación**.

💡 Idea principal

En lugar de modificar el arreglo original en cada operación, guardas solo los **cambios en los extremos** del rango:

1. Inicializa un arreglo `diff` del mismo tamaño que el original, lleno de ceros.
2. Para sumar `x` al rango `[l, r]`, haz:

   ```
   diff[l] += x
   diff[r + 1] -= x   # si r + 1 está dentro del arreglo
   ```

3. Al final, reconstruye el arreglo final aplicando **sumas prefix** (acumuladas) sobre `diff`.
   Es decir, cada posición del arreglo original será la suma acumulada de `diff` hasta ese punto.

⚙️ Ejemplo rápido

Supón que `arr = [0, 0, 0, 0, 0]`
y quieres:

- Sumar `+2` al rango `[1, 3]`
- Sumar `+3` al rango `[2, 4]`

Operaciones en `diff`:

```
diff[1] += 2
diff[4] -= 2
diff[2] += 3
diff[5] -= 3
```

Después de aplicar prefijos acumulados:

```
arr = [0, 2, 5, 5, 3]
```

Resultado final: las actualizaciones se aplican correctamente con complejidad **O(1)** por operación de rango y **O(n)** al final.

Problemas: Clase 9 – Algoritmos Voraces (Greedy) 1