

# Estructuras de datos

Wilmer Emiro Castrillón Calderón

13 de marzo de 2020

## 1. Tablas aditivas

Son estructuras de datos utilizadas para realizar operaciones acumuladas sobre un conjunto de datos estáticos en un rango específico, es decir, ejecutar una misma operación (como por ejemplo la suma) sobre un intervalo de datos, se asume que los datos en la estructura no van a cambiar. Estas estructuras también son conocidas como *Summed-area table* para el procesamiento de imágenes, a pesar de su nombre no necesariamente son exclusivas para operaciones de suma, pues la idea general es aplicable a otras operaciones.

Durante el cálculo de una misma operación sobre diferentes rangos se presenta superposición de problemas, las tablas aditivas son utilizadas para reducir la complejidad computacional aprovechando estas superposiciones utilizando programación dinámica.

### Ejemplo inicial.

Dado un vector  $V = \{5, 2, 8, 2, 4, 3, 1\}$  encontrar para múltiples consultas la suma de todos los elementos en un rango  $[i, j]$ , indexando desde 1, por ejemplo con el rango  $[1, 3]$  la suma es  $[5+2+8] = 15$ .

La solución trivial es hacer un ciclo recorriendo el vector entre el intervalo  $[i, j]$ , en el peor de los casos se debe recorrer todo el vector, esto tiene una complejidad  $O(n)$  puede que para una consulta sea aceptable, pero en casos grandes como por ejemplo un vector de tamaño  $10^5$  y una cantidad igualmente grande de consultas el tiempo de ejecución se hace muy alto, por lo tanto se hace necesario encontrar una mejor solución.

La operación suma tiene propiedades que nos pueden ayudar a resolver este problema de una forma mas eficiente:

1. La suma es asociativa es decir, se cumple:  $a + (b + c) = (a + b) + c$ , esto indica que sin importar la agrupación que se realice el resultado sera igual (no confundir con propiedad conmutativa).
2. La suma posee elemento neutro, es decir existe un  $\beta$  tal que  $a + \beta = a$ , en la suma  $\beta = 0$ .
3. La suma tiene operación inversa, es decir existe una operación que puede revertir la suma, la cual es la resta: si  $a + b = c$  entonces  $c - a = b$ .

Considerando las anteriores propiedades el problema se puede trabajar desde otro enfoque, primero se puede definir  $suma(x) = \sum_{k=1}^x V_k$ , ahora tomando como ejemplo una consulta en el rango  $[3, 5]$  del vector  $V$ , se tomara convenientemente:  $suma(2) = V_1 + V_2$  y  $suma(5) = V_1 + V_2 + V_3 + V_4 + V_5$ , pero se necesita encontrar  $V_3 + V_4 + V_5$ , usando la propiedad asociativa se obtiene:  $(V_1 + V_2) + (V_3 + V_4 + V_5) = (V_1 + V_2 + V_3 + V_4 + V_5)$  y usando la propiedad inversa se llega a:  $(V_3 + V_4 + V_5) = (V_1 + V_2 + V_3 + V_4 + V_5) - (V_1 + V_2)$  por lo tanto  $(V_3 + V_4 + V_5) = suma(5) - suma(2)$ .

De esta manera el problema se puede generalizar como  $\sum_{k=i}^j V_k = suma(j) - suma(i-1)$  cuando  $i \neq 1$  y  $suma(j)$  cuando  $i = 1$ .

Ahora pre-calculando  $suma(x)$  se puede dar una solución inmediata a cada consulta, esto se puede resolver utilizando un enfoque básico de programación dinámica. Para encontrar  $suma(x)$  se puede reescribir como:  $suma(x) = V_x + suma(x-1)$  con caso base  $suma(1) = V_1$  y por definición la operación acumulada sobre un conjunto vacío es igual al elemento neutro, a partir de esto se puede obtener la siguiente solución en C++ con consultas indexando desde 1.

```

1  int V[] = {5,2,8,2,4,3,1}, memo[8];
2
3  void precalcular(){
4      memo[0] = 0;
5      for(int i = 0; i < 7; i++){
6          memo[i+1] = V[i] + memo[i];
7      }
8
9  int consulta(int i, int j){ return memo[j] - memo[i-1]; }
```

De manera general las tablas aditivas son aplicables a cualquier operación que posea las tres propiedades descritas anteriormente: ser asociativa, tener elemento neutro y operación inversa, por ejemplo la suma, multiplicación o el operador de bits XOR.

## Tablas aditivas en 2D

Las operaciones acumuladas no solo se pueden usar sobre una dimension, sino también sobre n-dimensiones, en estos casos se debe trabajar con el principio de inclusión-exclusión pues se debe considerar mejor las operaciones entre intervalos, si no tiene en cuenta este principio las soluciones contendrían elementos duplicados o faltantes, lo que produciría soluciones incorrectas.

**Ejemplo:** dada la matriz  $M$  encontrar para múltiples consultas la suma de todos los elemento en una submatriz  $Q_{(i1,j1),(i2,j2)}$ :

$$M =$$

|   |   |   |   |   |
|---|---|---|---|---|
| 1 | 9 | 6 | 3 | 7 |
| 7 | 5 | 3 | 0 | 5 |
| 0 | 7 | 6 | 5 | 3 |
| 7 | 8 | 9 | 5 | 0 |
| 9 | 5 | 3 | 7 | 8 |

En el intervalo  $Q_{(2,2),(3,4)}$  el resultado es  $5 + 3 + 0 + 7 + 6 + 5 = 26$ .

En el caso de 1D se definió  $suma(x) = \sum_{k=1}^x V_k$ , ahora esta debe tener dos dimensiones, es decir,  $suma(i, j)$  debe contener la suma de los elementos en la submatriz  $Q_{(1,1),(i,j)}$ , entonces ahora se definirá:  $suma(i, j) = \sum_{k=1}^i \sum_{w=1}^j M_{k,w}$ , mas sin embargo pre-calculando  $suma(i, j)$  de forma eficiente requiere de usar el principio de inclusión-exclusión, de manera trivial se puede calcular la primera fila como  $suma(1, j) = \sum_{w=1}^j M_{1,w}$  y la primera columna como  $suma(i, 1) = \sum_{k=1}^i M_{k,1}$ , en base a esto se puede calcular el resto de la matriz pero se debe tener algo de cuidado, por ejemplo si se toma  $suma(2, 2) = suma(1, 2) + suma(2, 1) + M_{2,2}$  se obtendría un resultado incorrecto pues se estaría realizando la siguiente operación:  $(M_{1,1} + M_{1,2}) + (M_{1,1} + M_{2,1}) + M_{2,2}$ , se puede observar que el elemento  $M_{1,1}$  se esta sumando dos veces, acá se aplica el principio de inclusión-exclusión:  $|A| \cup |B| = |A| + |B| - |A \cap B|$  lo que significa que hace falta quitar la intersección, esta es  $suma(1, 1)$  entonces se puede generalizar como:  $suma(i, j) = M_{i,j} + suma(i-1, j) + suma(i, j-1) - suma(i-1, j-1)$  cuando  $i, j \neq 1$ .

Una vez construido el pre-calculo es necesario realizar las consultas, se utilizara como ejemplo la consulta en el rango  $Q_{(2,2),(3,4)}$ , de igual manera se debe tener cuidado de no sumar un

mismo intervalo mas de una vez, entonces para encontrar la suma en este intervalo se tomaría  $suma(i2, j2)$  esta contendría  $\sum_{k=1}^{i2} \sum_{w=1}^{j2} M_{k,w}$ , esta tiene elementos adicionales como lo muestra la figura 1, al restarle  $suma(i1 - 1, j2)$  se quitan algunos elementos (figura 2), al restarle  $suma(i2, j1 - 1)$  pasaríamos a restar dos veces el intervalo  $M_{(i1,1),(i1-1,j1-1)}$  (figura 3), por lo tanto es necesario reponer lo faltante agregando  $suma(i1 - 1, j1 - 1)$  (figura 4), de esta manera se puede generalizar la formula:  $\sum_{k=i1}^{i2} \sum_{w=j1}^{j2} M_{k,w} = suma(i2, j2) - suma(i1 - 1, j2) - suma(i2, j1 - 1) + suma(i1 - 1, j1 - 1)$ .

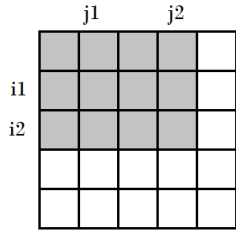


Figura 1

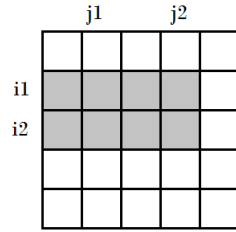


Figura 2

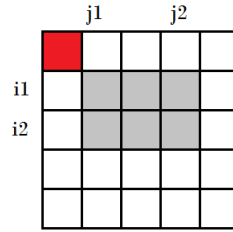


Figura 3

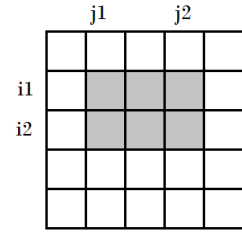


Figura 4

Para la solución en C++ la fila y columna cero de la tabla donde se guardara el pre-cálculo se deberá llenar con el elemento neutro, el cero, luego se debe llenar el resto de la tabla siguiendo las formulas antes establecidas, este ejemplo es para consultas indexando desde 1.

```

1 void precalcular(){
2     memset(memo, 0, sizeof(memo));
3     for (int i = 1; i <= fila; i++)
4         for (int j = 1; j <= col; j++)
5             memo[i][j] = memo[i][j - 1] + memo[i - 1][j] +
6                 M[i - 1][j - 1] - memo[i - 1][j - 1];
7 }
8
9 int consulta(int f1, int c1, int f2, int c2){
10     return memo[f2][c2] - memo[f1-1][c2] - memo[f2][c1-1] + memo[f1-1][c1-1];
11 }

```

### Tablas aditivas en 3D

Las tablas aditivas se pueden generalizar para trabajar en n-dimensiones, mas sin embargo la dificultad de hacer los pre-cálculos y las consultas aumenta bastante, pues crece considerablemente la cantidad de operaciones a realizar. En el caso 3D igualmente se debe tener cuidado con el principio de inclusión-exclusión, el pre-cálculo se realizaría de la siguiente manera: sea  $suma(i, j, k) = \sum_{x=1}^i \sum_{y=1}^j \sum_{z=1}^k V_{x,y,z}$ , entonces  $suma(i, j, k) = V_{i,j,k} + suma(i, j, k - 1) + suma(i, j - 1, k) + suma(i, j - 1, k - 1) - suma(i - 1, j, k) - suma(i - 1, j, k - 1) - suma(i - 1, j - 1, k) - suma(i - 1, j - 1, k - 1)$ , y para las consultas:  $\sum_{x=i1}^{i2} \sum_{y=j1}^{j2} \sum_{z=k1}^{k2} V_{x,y,z} = suma(i2, j2, k2) - suma(i2, j2, k1 - 1) - suma(i1 - 1, j2, k2) + suma(i1 - 1, j2, k1 - 1) - suma(i2, j1 - 1, k2) + suma(i2, j1 - 1, k1 - 1) + suma(i1 - 1, j1 - 1, k2) - suma(i1 - 1, j1 - 1, k1 - 1)$ .

Esta estructura de datos puede ser aplicada para otras operaciones que cumplan con las tres propiedades descritas anteriormente, la complejidad computacional de hacer el pre-cálculo es lineal a la cantidad de elementos en la estructura, y las consultas se realizan en complejidad constante al realizar solamente operaciones directas sobre elementos en la tabla del pre-cálculo.

## 2. Bibliografia

<http://trainingcamp.org.ar/antiores/2017/clases.shtml>.  
libro: competitive programming 3.