Binary Indexed Tree

B.S. Rodolfo Mercado Gonzales

Binary Indexed Tree

- Estructura de datos creada por Peter Fenwick.
- Conocida también como Fenwick Tree.
- Permite manipular eficientemente tablas de frecuencias acumuladas.

Tabla de frecuencias acumuladas

Tabla de frecuencias acumuladas

$$\mathrm{ac}[i] = f[1] + f[2] + \cdots + f[i]$$

índice	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{f}[\,]$	0	1	0	1	2	3	2	1	1	0
$\boldsymbol{ac}[\,]$	0	1	1	2	4	7	9	10	11	11

Tabla de frecuencias acumuladas

¿Cómo generamos la tabla de frecuencias acumuladas?

La podemos generar con programación dinámica:

$$ac[1] = f[1],\ i = 1$$

$$ac[i] = ac[i-1] + f[i],\ i > 1$$

construcción : $O(n)$

consulta : $O(1)$

actualizar frecuencia : $O(n)$

¿ y si tenemos muchas actualizaciones?

Binary Indexed Tree

Dado un arreglo, permite realizar dos tipos de operaciones:

1. Hallar la suma de los primeros k elementos del arreglo ($ac[\,]$) $O(\log n)$
2. Cambiar el valor del k-ésimo elemento ($f[\,]$) $O(\log n)$

Arreglo de sumas parciales

$$\boldsymbol{tree[i] = f[i - 2^r + 1] + \ldots + f[i]}$$

$$\boldsymbol{r\ es\ la\ posición\ del\ último\ bit\ encendido\ de\ i}$$

$tree[12] = ??$

$12 = 0 \ldots 01\mathbf{1}00, \qquad \mathrm{r} = 2$

$tree[12] = f[9] + f[10] + f[11] + f[12]$

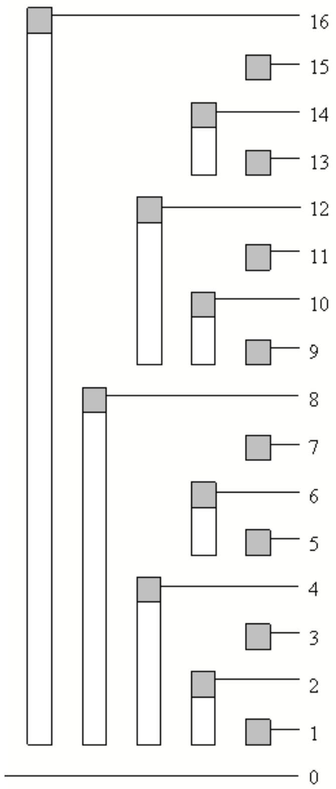
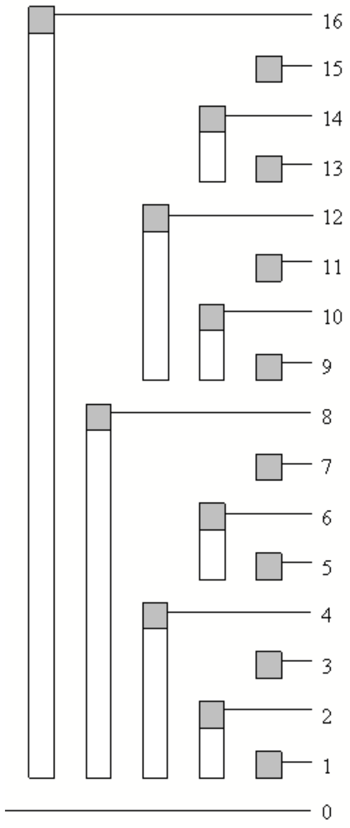


Tabla de responsabilidad

Arreglo de sumas parciales

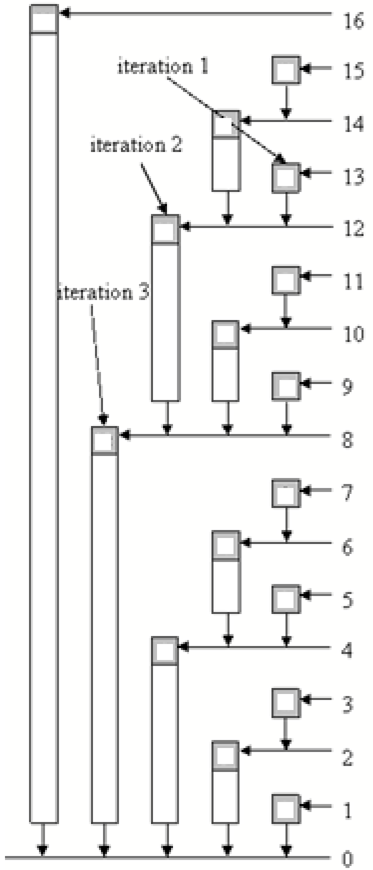
índice	1	2	3	4	5	6	7	8	9	10
$f[\,]$	0	1	0	1	2	3	2	1	1	0
$ac[\,]$	0	1	1	2	4	7	9	10	11	11
$tree[\,]$	0	1	0	2	2	5	2	10	1	1



Frecuencia acumulada

¿ Cómo hallamos $ac[\ i\ ]$?

Así como todo número puede ser representado como suma de potencias de 2, la frecuencia acumulada puede ser representada a través de sumas parciales.



$$\boldsymbol{ac}[\mathbf{13}] = ac[1101] = \ ??$$

$$\boldsymbol{tree}[\mathbf{1101}] = f[13] +$$
$$\boldsymbol{tree}[\mathbf{1100}] = f[9] + \cdots + f[12] +$$
$$\boldsymbol{tree}[\mathbf{1000}] = f[1] + \cdots + f[8]$$
$$\boldsymbol{tree}[\mathbf{0000}]$$

vamos eliminando el último bit encendido (potencia de 2) hasta quedarnos con el índice 0.

Último bit encendido

Para un $x = 14$, tenemos:

bit	1	1	1	0
posición	3	2	1	0

- El último bit encendido nos lo da la operación: $\boldsymbol{x} \,\&\, -\boldsymbol{x}$
- **(-)** es un operador unario que genera el negativo de un número.
- Como los números se guardan en complemento a 2, entonces: $-\boldsymbol{x} = \sim\boldsymbol{x} + \mathbf{1}$

Último bit encendido

<u>Demostración</u>

Sea $x = \overline{a10 \ldots 0}$

$$-x = \overline{\sim(a10 \ldots 0)} + 1, \qquad -x = \overline{(\sim a)01 \ldots 1} + 1, \qquad -x = \overline{(\sim a)10 \ldots 0}$$

Finalmente,

$$x \; \& \; -x = \overline{a10 \ldots 0} \; \& \; \overline{(\sim a)10 \ldots 0} = \mathbf{0 \ldots 010 \ldots 0} = \mathbf{2}^r$$

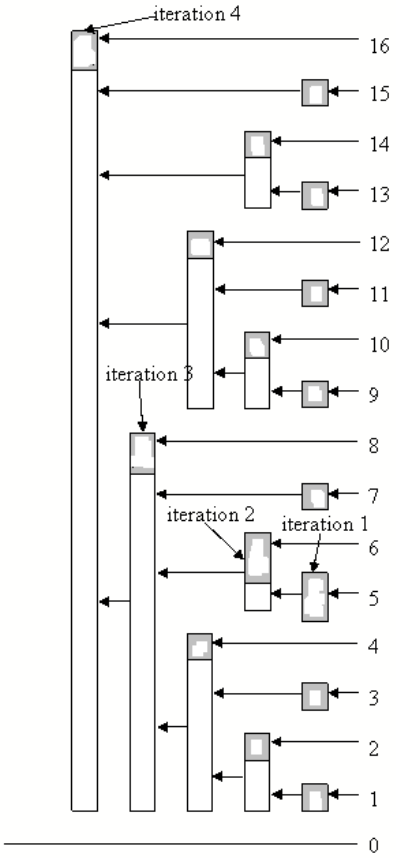
Frecuencia acumulada

```
int query( int i ){
    int sum = 0;
    while( i > 0 ){
        sum += tree[ i ];
        i -= ( i & -i );
    }
    return sum;
}
```

Complejidad: $O(\log n)$

Actualizar frecuencia

Para actualizar el valor de una frecuencia f[ ], debemos actualizar el arreglo $tree[\,]$ en aquellos índices que son responsables de la posición a actualizar.



Si queremos actualizar $f[5]$ entonces debemos actualizar:

$$tree[5], \qquad tree[6], \qquad tree[8], \qquad tree[16], \dots$$

Actualizar frecuencia

Dado un índice $\boldsymbol{i}$, ¿cómo hallamos $\boldsymbol{j}$, el menor responsable de $\boldsymbol{i}$?

Podemos hallar j como:

$$\boldsymbol{j} = \boldsymbol{i} + (\boldsymbol{i} \,\&\, -\boldsymbol{i})$$

Para $i = 10$, entonces $j = 12$

Para $i = 5$, entonces $j = 6$

Actualizar frecuencia

```
void update( int i, int delta ){
    while( i <= n ){
        tree[ i ] += delta;
        i += ( i & -i );
    }
}
```

Complejidad: $O(\log n)$

Binary Indexed Tree

```
struct FenwickTree{
    int tree[ n + 1 ];

    FenwickTree(){
        for( int i = 0; i <= n; ++i ) tree[ i ] = 0;
    }

    int query( int i ){
        int sum = 0;
        while( i > 0 ){
            sum += tree[ i ];
            i -= ( i & -i );
        }
        return sum;
    }

    void update( int i, int delta ){
        while( i <= n ){
            tree[ i ] += delta;
            i += ( i & -i);
        }
    }
} ft;
```

Actualizar rango – consultar frecuencia

Supongamos que queremos actualizar un rango de frecuencias (incrementar o disminuir su valor en un delta) y que también queremos hacer consultas sobre el valor actual que tiene una frecuencia. ¿ Podemos hacerlo eficientemente ?

Sea nuestro arreglo de frecuencias: $f[\,] = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

Si incrementamos en 3 al rango $[4{,}7]$:

índice	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{f}[\,]$	0	0	0	3	3	3	3	0	0	0
$\boldsymbol{ac}[\,]$	0	0	0	3	6	9	12	12	12	12

Actualizar rango – consultar frecuencia

Actualizar rango

En general para incrementar $\boldsymbol{delta}$ en el rango $[\boldsymbol{L}, \boldsymbol{R}]$, podemos hacer:

update($\boldsymbol{L}, \boldsymbol{delta}$ **)** y **update(** $\boldsymbol{R} + \boldsymbol{1}, -\boldsymbol{delta}$ **)**

Consultar frecuencia

El valor de $\boldsymbol{f}[\ \boldsymbol{i}\ ]$ estaría dado por $\boldsymbol{query}(\boldsymbol{i})$

índice	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{f}[\,]$	0	0	0	3	0	0	0	-3	0	0
$\boldsymbol{ac}[\,]$	0	0	0	3	3	3	3	0	0	0

Problemas

SPOJ – Inversion Count

Codeforces – Little Girl and Maximum Sum

Actualizar rango – consultar en rango

Si queremos actualizar un rango de frecuencias (incrementar o disminuir su valor en un delta) y también hacer consultas sobre la suma de frecuencias en un rango. ¿Podemos hacerlo eficientemente?

Sea nuestro arreglo de frecuencias: $f[\,] = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

Si incrementamos en 3 al rango $[4{,}7]$

índice	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{f}[\,]$	0	0	0	3	3	3	3	0	0	0
$\boldsymbol{ac}[\,]$	0	0	0	3	6	9	12	12	12	12

Si queremos saber la suma de frecuencias en el rango [5, 6] = ac[6] – ac[4]

Actualizar rango – consultar en rango

Supongamos que haremos un update de $\boldsymbol{V}$ en el rango $[\boldsymbol{L}, \boldsymbol{R}]$ y que luego necesitamos hallar el valor del acumulado para un índice $\boldsymbol{K}$. (todas las demás frecuencias están en 0).

- **Caso 1:** $1 \leq K < L$, su acumulado no varía.
- **Caso 2:** $L \leq K \leq R$, su acumulado incrementa en $\mathrm{V}(K) - V(L-1)$
- **Caso 3:** $R \leq K \leq N$, su acumulado incrementa en $\mathrm{V}(R) - V(L-1)$

Actualizar rango – consultar en rango

CASO 1: Para todo K en $[1, L >$ su acumulado no varía.

Hacer un update en $[\boldsymbol{L}, \boldsymbol{R}]$ no afecta en el acumulado de ningún índice K en $[\mathbf{1}, \boldsymbol{L} >$

Actualizar rango – consultar en rango

CASO 2: Para todo K en $[L, R]$, su acumulado incrementa en $\mathbf{V}(\boldsymbol{K}) - \boldsymbol{V}(\boldsymbol{L}-\mathbf{1})$

Ya que depende el primer término del valor de K ….

- Tendremos un primer FT que nos dirá el valor del K-ésimo elemento (**V**)

$\boldsymbol{ft1.update(L,\ V)}$ $\ \boldsymbol{y}\ \boldsymbol{ft1.update(R+1,\ -V)}$

- Un segundo FT similar nos dirá cuánto falta restar para tener el incremento exacto.

$\boldsymbol{ft2.update(\ L,\ V(L-1)\ )}$ $\ \boldsymbol{y}\ \boldsymbol{ft2.update(\ R+1,\ -V(L-1))}$

Actualizar rango – consultar en rango

CASO 3: Para todo $\boldsymbol{K}$ en $[\boldsymbol{R}, \boldsymbol{N}]$, su acumulado incrementa en $\mathbf{V}(\boldsymbol{R}) - \boldsymbol{V}(\boldsymbol{L} - \mathbf{1})$

No depende del valor de K, o lo podemos expresar como : $0(\boldsymbol{K}) - \left(-\mathbf{V}(\boldsymbol{R}) + \boldsymbol{V}(\boldsymbol{L} - \mathbf{1})\right)$

- Solo necesitamos actualizar nuestro segundo FT.

$$\boldsymbol{ft2.update}(\boldsymbol{R} + \mathbf{1}, -\boldsymbol{V}(\boldsymbol{R}) + \boldsymbol{V}(\boldsymbol{L} - \mathbf{1}))$$

Actualizar rango – consultar en rango

- Actualizar rango **[L,R]**
- $\boldsymbol{ft1.update(L,V)}$ $\boldsymbol{y}$ $\boldsymbol{ft1.update(R+1,-V)}$
- $\boldsymbol{ft2.update(\ L,V(L-1)\ )}$ $\boldsymbol{y}$ $\boldsymbol{ft2.update(\ R+1,-V(R))}$
- Consultar acumulado de un índice **K**
- $\boldsymbol{query(K) = \ ft1.query(K) * K - ft2.query(K)}$
- Consultar rango **[L, R]**
- $\boldsymbol{query(L,R) = query(R) - query(L-1)}$

Problemas

SPOJ – Horrible Queries

Binary Indexed Tree en 2D

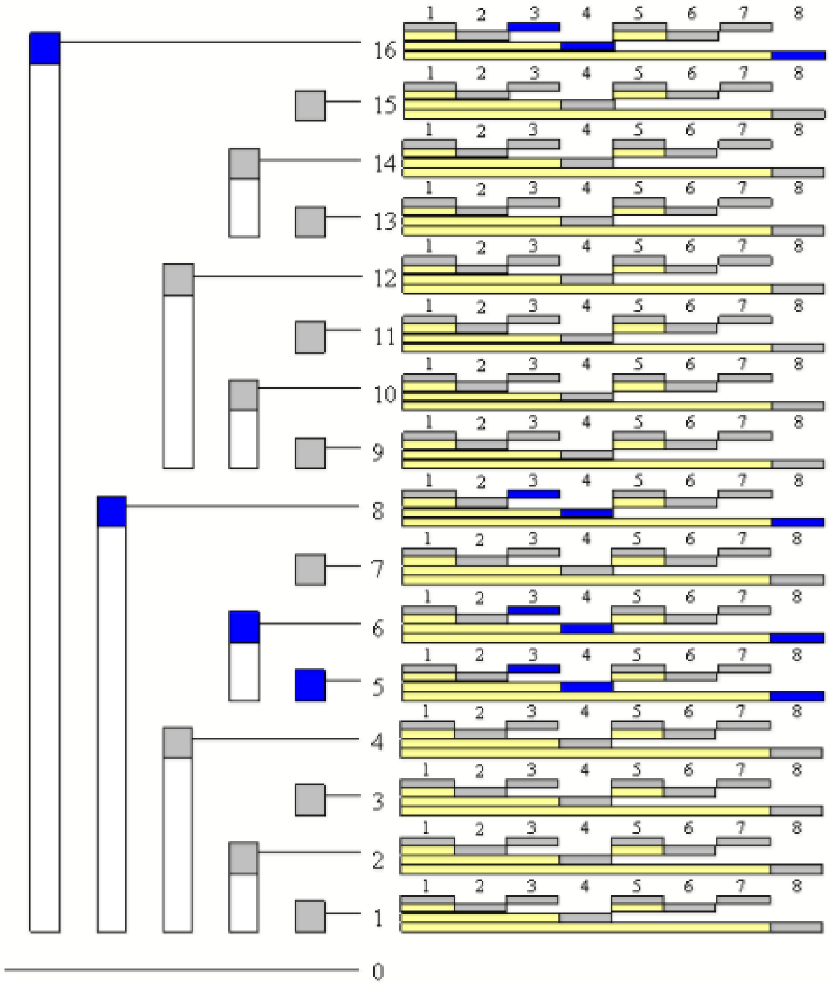
Dado un plano con puntos (celdas) con coordenadas positivas, deseamos :

- ❑ Aumentar el valor de la celda (i, j) en delta
- ❑ Consultar el acumulado del rectángulo desde (1,1) hasta (i, j).

Binary Indexed Tree 2D

Ahora cada elemento de nuestro FT también será un FT, por ello tendremos $\boldsymbol{tree[maxX][maxY]}$

Blue fields are fields which we should update when we are updating index (5 , 3).



Binary Indexed Tree 2D

Actualizando el valor de la celda (x, y) :

```
void update(int x , int y , int val){
    while (x <= max_x){
        updatey(x , y , val);
        // this function should update array tree[x]
        x += (x & -x);
    }
}
```

```
void updatey(int x , int y , int val){
    while (y <= max_y){
        tree[x][y] += val;
        y += (y & -y);
    }
}
```

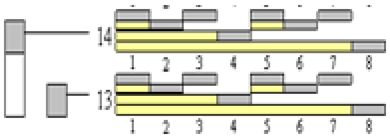
Binary Indexed Tree 2D

Actualizando el valor de la celda (i, j) :

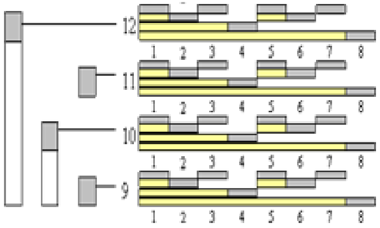
```
void update( int i, int j, int delta ){
    while( i <= maxX ){
        int _j = j;
        while( _j <= maxY ){
            tree[ i ][ _j ] += delta;
            _j += ( _j & -_j );
        }
        i += ( i & -i );
    }
}
```

Binary Indexed Tree 2D

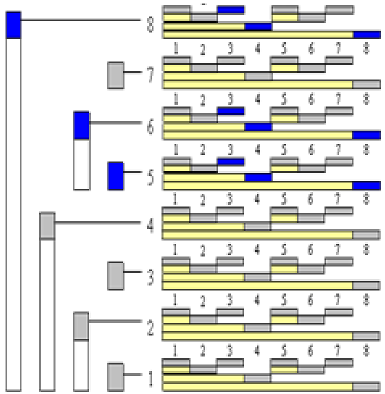
Obtener acumulado desde (1,1) al (14,8) .



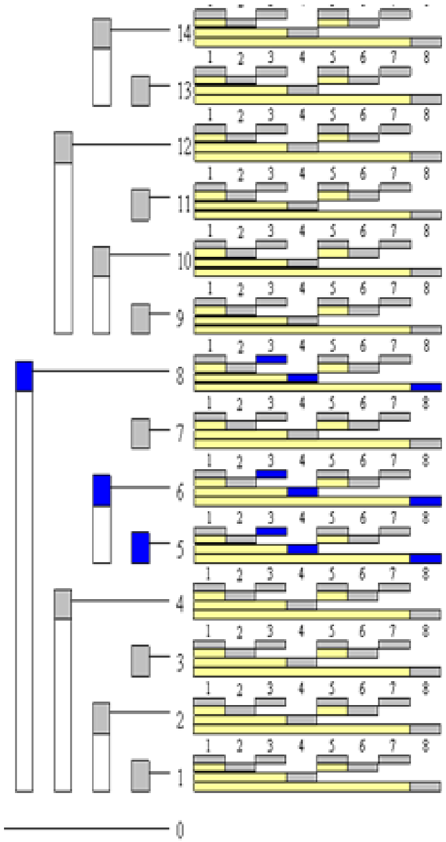
Primer bloque



Segundo bloque



Tercer bloque



Binary Indexed Tree 2D

Consultando el acumulado desde (1,1) hasta (i, j) :

```
ll query( int i, int j ){
    ll sum = 0LL;
    while( i > 0 ){
        int _j = j;
        while( _j > 0 ){
            sum += tree[ i ][ _j ];
            _j -= ( _j & -_j );
        }
        i -= ( i & -i );
    }
    return sum;
}
```

Problemas

SPOJ – Matrix Summation

Referencias

- Halim, Steven et al. *Competitive Programming 3*
- Topcoder, *Binary Indexed Tree*
- Hackerearth, *Binary Indexed Tree or Fenwick Tree*

¡ Good luck and have fun !