Sparse Table

B.S. Rodolfo Mercado Gonzales

Sparse Table

- Estructura de datos que nos permite responder consultas sobre un rango, pero que solo puede ser usada con datos inmutables (no updates).

- Cada consulta en realidad es calcular el valor de una función **asociativa** sobre un rango.

$$f(A_L, A_{L+1}, \dots, A_R)$$

Idea base

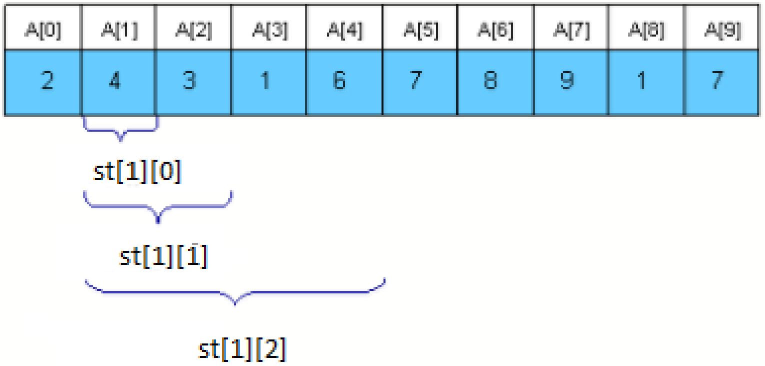
Si precalculamos todas las respuestas para rangos de tamaño igual a una potencia de 2, luego será posible responder sobre un rango de cualquier tamaño, ya que siempre podremos representar dicho rango en base a los precalculados.

$$[5,17] = [5,12] + [13,16] + [17,17]$$

Precálculo

Usaremos una matriz para precalcular respuestas sobre rangos de tamaño igual a una potencia de 2.

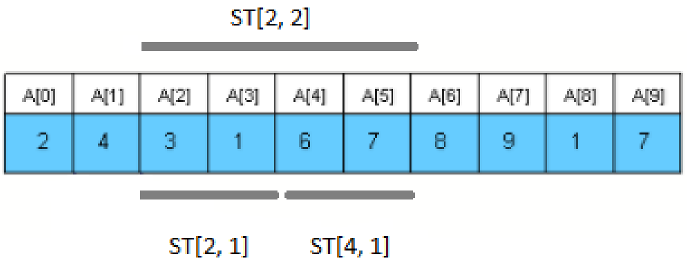
$$\boldsymbol{ST}[\,\boldsymbol{i}\,][\,\boldsymbol{j}\,] = respuesta\ sobre\ rango\ [i, i + 2^j - 1]$$



Precálculo

Para realizar el precálculo, observemos que un rango $[i, i+2^j-1]$ de tamaño 2^j puede ser dividido en dos rangos de tamaño 2^{j-1}, que son $[i, i+2^{j-1}-1]$ y $[i+2^{j-1},\ i+2^j-1]$.

$$ST[\,i\,][\,j\,] = f(\,ST[\,i\,][\,j-1\,], ST\left[\,i+2^{j-1}\right][\,j-1]\,)$$



Precálculo

Ahora podemos completar nuestro sparse table, usando programación dinámica.

```
int ST[ N ][ log2(N) + 1 ]

void precalculo( int A[ ], int n ){

    for( int i = 0; i < n; ++i )
        ST[ i ][ 0 ] = f( A[ i ] ) ;

    for( int j = 1; ( 1 << j ) <= n; ++j )
        for( int i = 0; i + ( 1 << j ) <= n; ++i )
            ST[ i ][ j ] = f( ST[ i ][ j - 1], ST[ i + ( 1 << (j - 1) ) ][ j - 1 ] );
}
```

$$O(n \log n)$$

Consultas

Para formar un rango $[\boldsymbol{L}, \boldsymbol{R}]$ tenemos que expresar su tamaño $\mathrm{T} = R - L + 1$ como suma de potencias de 2, esto se puede hacer recorriendo las potencias de mayor a menor.

Sea $\boldsymbol{T} = \mathbf{2}^{\boldsymbol{j}} + \mathbf{2}^{\boldsymbol{k}} + \cdots + \mathbf{2}^{\boldsymbol{x}}$, tal que $j > k > \cdots > x$

$$\boldsymbol{f}(\,[\boldsymbol{L}, \boldsymbol{R}]\,) = \boldsymbol{f}(\,\boldsymbol{ST}[\boldsymbol{L}][\boldsymbol{j}], \boldsymbol{ST}[\boldsymbol{L} + \mathbf{2}^{\boldsymbol{j}}][\boldsymbol{k}]\,, \ldots)$$

 # intervalos necesarios = $\boldsymbol{log_2^{(T)}}$

Consultas

```
int query( int L, int R ){
    int ans = ?, T = R - L + 1;
    int lg = 31 - ( __builtin_clz ( T ) ) ;
    for ( int j = lg; j >= 0; j--) {
        if ( (1 << j) <= T ) {
            ans = f( ans, ST[ L ][ j ] );
            L += ( 1 << j );
            T -= ( 1 << j );
        }
    }
    return ans;
}
```

$$O(\log n)$$

Range Sum Query

Cada consulta en este problema nos pide hallar la suma de los elementos de un arreglo $\boldsymbol{A}$, en un rango $[\boldsymbol{L}, \boldsymbol{R}]$, es decir : $\boldsymbol{A_L + A_{L+1} + \cdots + A_R}$

Solución usando sparse table

Ahora nuestra **función** representará la suma $f(a, b) = a + b$, solo tenemos que editarlo en nuestras funciones “**precálculo**” $O(n \log n)$ y “**query**” $O(\log n)$ del sparse table.

Range Minimum Query (RMQ)

Cada consulta en este problema nos pide hallar el menor elemento de un arreglo A en un rango $[L, R]$, es decir : $\boldsymbol{min\ (A_L, A_{L+1}, \dots, A_R)}$

Solución 1

Ahora nuestra **función** representará el mínimo $f(a, b) = \min(a, b)$, si solo reemplazamos esto en nuestras funciones del sparse table, el precálculo seguirá siendo $O(n \log n)$ y la consulta $O(\log n)$.

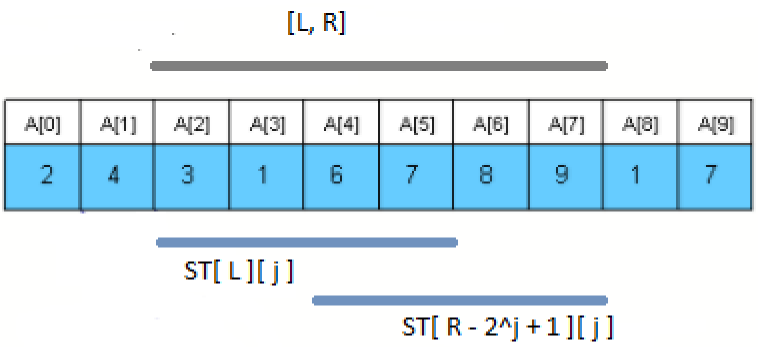
¿Se puede mejorar?

Range Minimum Query (RMQ)

En lugar de dividir en múltiples rangos, para este caso nos es suficiente dividirlos en solo 2 rangos que probablemente se sobrelapen (igual el mínimo no variará al repetir elementos) pero que cubran todo el rango original.

```
int query( int L, int R ){
    int T = R - L + 1;
    int lg = 31 - ( __builtin_clz ( T ) ) ;
    return min( ST[ L ][ lg ] , ST[ R - ( 1 << lg ) + 1 ][ lg ] );
}
```

$O(1)$



Problemas

SPOJ – Catapult that ball

Referencias

- E-maxx, Sparse Table
- Hackerearth, Sparse Table

¡ Good luck and have fun !