Sparse Table

Oscar Gauss Carvajal Yucra

1 Introducción

En este escrito veremos el funcionamiento de la estructura *Sparse Table*, esta estructura nos permitirá responder preguntas en rangos de un vector con una complejidad de $O(1)$ para aquellas operaciones que sean asociativas [1] e idempotentes [2]. Con un preproceso de $O(Nlog_2N)$ donde N es el tamaño del vector.

2 Preproceso

Para que la estructura se entienda trabajaremos con la operación *mínimo* ya que es asociativo e idempotente, para trabajar con otra operación que sea asociativa e idempotente se trabaja de forma análoga.

Este algoritmo se basa en la idea de Divide y Vencerás memorizando los resultados en una tabla de N x log_2N.

Llamemos a la tabla de memorización $'ST'$, la posición $ST[i][j]$ nos guardara el índice del mínimo elemento del rango que empieza en j y tiene una longitud de 2^i.

En el siguiente vector de longitud 9:

índice:	0	1	2	3	4	5	6	7	8
A[9]:	9	20	-8	5	13	2	5	4	11

la posición $i = 2$, $j = 3$ de ST alojaría el índice 5 ($A[5] = 2$) ya que es el índice del mínimo elemento del rango que empieza en $j = 3$ y tiene longitud $2^2 = 4$ en otras palabras es el índice del menor elemento de entre los elementos $A[3] = 5$, $A[4] = 13$, $A[5] = 2$ y $A[6] = 5$.
La posición $i = 3$, $j = 1$ de ST alojaría el índice 2 ($A[2] = -8$) ya que es el índice del mínimo elemento del rango que empieza en $j = 1$ y tiene longitud $2^3 = 8$ en otras palabras es el índice del menor elemento de entre los elementos $A[1] = 20$, $A[2] = -8$, $A[3] = 5$, $A[4] = 13$, $A[5] = 2$, $A[6] = 5$, $A[7] = 4$ y $A[8] = 11$.

[1] La asociatividad indica que, cuando existen tres o más cifras en estas operaciones, el resultado no depende de la manera en la que se agrupan los términos.

[2] La idempotencia es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado que se obtendría si se realizase una sola vez.

La posición $i = 3$, $j = 4$ de ST no alojaría nada por que el rango que empieza en $j = 4$ y tiene longitud $2^3 = 8$ no existe.

La tabla ST completa asociado al vector de arriba es:

ST[4][9]:

i\j	0	1	2	3	4	5	6	7	8
0	0	1	2	3	4	5	6	7	8
1	0	2	2	3	5	5	7	7	-
2	2	2	2	5	5	5	-	-	-
3	2	2	-	-	-	-	-	-	-

Los valores de la primera fila cuando $i = 0$ (donde se alojan los rangos que empiezan en j y tiene longitud $2^0 = 1$) es sencillo de calcular solo es j por que el rango solo tiene un elemento.

$$ST[0][j] = j$$

Para calcular los valores de la segunda fila cuando $i = 1$ (donde se alojan los rangos que empiezan en j y tienen longitud $2^1 = 2$) debemos observar la fila anterior y preguntar por el elemento de índice ST[0][j] y por el elemento de índice ST[0][j+1] y elegir el menor.

$$ST[1][j] = \begin{cases} ST[0][j] & \text{si } A[ST[0][j]] \leq A[ST[0][j+1]] \\ ST[0][j+1] & \text{si } A[ST[0][j+1]] < A[ST[0][j]] \end{cases}$$

Para calcular los valores de la tercera fila cuando $i = 2$ (donde se alojan los rangos que empiezan en j y tienen longitud $2^2 = 4$) debemos observar la fila anterior y preguntar por el elemento de índice ST[1][j] y por el elemento de índice ST[1][j+2] y elegir el menor.

$$ST[2][j] = \begin{cases} ST[1][j] & \text{si } A[ST[1][j]] \leq A[ST[1][j+2]] \\ ST[1][j+2] & \text{si } A[ST[1][j+2]] < A[ST[1][j]] \end{cases}$$

Para calcular los valores de la cuarta fila cuando $i = 3$ (donde se alojan los rangos que empiezan en j y tienen longitud $2^3 = 8$) debemos observar la fila anterior y preguntar por el elemento de índice ST[2][j] y por el elemento de índice ST[2][j+4] y elegir el menor.

$$ST[3][j] = \begin{cases} ST[2][j] & \text{si } A[ST[2][j]] \leq A[ST[2][j+4]] \\ ST[2][j+4] & \text{si } A[ST[2][j+4]] < A[ST[2][j]] \end{cases}$$

Una vez entendido como construir las columnas $i > 0$ podemos generalizar la construcción de la tabla de la siguiente manera.

$$ST[i][j] = \begin{cases} j & \text{si } i = 0 \\ ST[i-1][j] & \text{si } A[ST[i-1][j]] \leq A[ST[i-1][j+2^{i-1}]] \\ ST[i-1][j+2^{i-1}] & \text{en otro caso} \end{cases}$$

Ahora veamos una implementación en $C++$, y recordar que para calcular 2^x de forma rápida en $C++$ debemos usar el desplazamiento de bits $1 << x$.

```cpp
#define N 500050
#define LN 20

long long A[N];
int ST[LN][N];

void STInit(int n){
  for(int j=0; j<n; j++) ST[0][j]=j;
  for(int i=1; (1<<i )<=n; i++)
    for(int j=0; j+(1<<i)<=n; j++){
      int a=ST[i-1][j]; // primera mitad
      int b=ST[i-1][j+(1<<(i-1))]; // segunda mitad
      if( A[a]<A[b]) ST[i][j]=a;
      else ST[i][j]=b;
    }
}
```

Donde LN es un valor techo aproximado de $log_2 N$, A es el vector donde están los elementos y ST que es la tabla que vimos atrás.

En lugar de guardar índices también se puede guardar el valor de la operación realizada.

3 Query

Ahora con la tabla creada ST debemos ser capaces de responder en $O(1)$ cual es el elemento mínimo dado un rango del vector.

Dado un rango $[I, J]$ del vector podemos decir que una parte o toda la respuesta esta en algún lugar de la columna I por que sabemos que la columna I guarda los resultados de los rangos que empiezan en I. También podemos decir que en la posición $i = 0, j = J$ esta parte o toda la respuesta ya que el rango por lo menos incluye un elemento osea que $J-I+1 >= 1$, si $J-I+1 \geq 2$ podemos decir que en la posición $i = 1, j = I$ esta parte o toda la respuesta, si $J - I + 1 \geq 4$ podemos decir que en la posición $i = 2, j = I$ esta parte o toda la respuesta, si $J - I + 1 \geq 8$ podemos decir que en la posición $i = 3, j = I$ esta parte o toda la repuesta, y así hasta que para algún x no cumpla que $J - I + 1 \geq 2^x$.

Debemos encontrar el máximo x que cumpla $J - I + 1 \geq 2^x$ despejando tenemos $x = log_2(J - I + 1)$ entonces estamos seguros que se cumple $J - I + 1 \geq 2^x$ y podemos estar seguros que $ST[x][I]$ esta parte o toda la respuesta por que guarda la respuesta del prefijo de longitud 2^x del rango, también podemos preguntar por $ST[x][J - 2^x + 1]$ por que guarda la respuesta del sufijo de longitud 2^x del rango.
Entonces tenemos:

$$Query(I, J) = \begin{cases} ST[x][I] & \text{si } A[ST[x][I]] \leq A[ST[x][J - 2^x + 1]] \\ ST[x][J - 2^x + 1] & \text{en otro caso} \end{cases}$$

$$\text{donde } x \text{ es } log_2(J - I + 1)$$

Veamos un ejemplo, si queremos preguntar por el rango $[2, 6]$ nuestro x seria 2, cumpliendo con $5 \geq 2^2$ $(J - I + 1 \geq 2^x)$ y debemos preguntar por las casillas

$i = x = 2, j = I = 2$ y por $i = x = 2, j = J - 2^x + 1 = 3$.

indice	9	20	-8	5	13	2	5	4	11
A:	0	1	2	3	4	5	6	7	8

ST[4][9]:	i,j	0	1	2	3	4	5	6	7	8
	0	0	1	2	3	4	5	6	7	8
	1	0	2	2	3	5	5	7	7	-
	2	2	2	2	5	5	5	-	-	-
	3	2	2	-	-	-	-	-	-	-

indice del minimo del rango:

indice del minimo del rango:

indice	-8	5	13	2
A[2-5]	2	3	4	5

indice	5	13	2	5
A[3-6]	3	4	5	6

Siendo el resultado 2 lo que significa que del rango $[2, 6]$ el mínimo elemento se encuentra en la posición 2 que es el valor $A[2] = -8$ siendo el elemento buscado.

Ahora veamos una implementación en $C++$:

```
int STQuery(int I, int J){
  int x=32-__builtin_clz(J-I+1)-1;
  if( A[ ST[x][I] ] < A[ ST[x][ J-(1<<x)+1 ] ] )
    return ST[x][I];
  return ST[x][J-(1<<x)+1];
}
```

Para hallar $log_2(M)$ en $C++$ se puede hacer de una forma muy ingeniosa: Restamos 32 con la cantidad de ceros a la izquierda que tiene el entero M con $\_\_builtin\_clz(M)$ así tendremos la cantidad de bits que se usa para representar M en binario sin ceros a la izquierda. Si M fuera 8 entonces $32 - \_\_builtin\_clz(M)$ llegaría a ser 4, si M fuera 9 entonces $32 - \_\_builtin\_clz(M)$ llegaría a ser 5. Entonces x llegaría a ser el resultado anterior menos 1. Al final tenemos que $log_2(M) = 32 - \_\_builtin\_clz(M) - 1$

4 Bibliografía

https://www.topcoder.com/community/data-science/data-science-tutorials/