Estructura de Datos: Segment Tree

Por Lautaro Lasorsa, para el curso OIA - UNLaM 2020

Motivación

Tenemos un vector *V* y una operación asociativa *Op* (ej: suma, producto, mínimo, máximo, máximo común divisor, concatenar strings, etc.) y queremos poder realizar las siguientes acciones de forma RÁPIDA:

- Cambiar el valor de un elemento de *V*
- Dado un rango [L;R], saber el valor de aplicar la operación *Op* en dicho rango. Es decir, saber $Op(V_L, V_{L+1}, \ldots, V_{R-1}, V_R)$.

¿Cómo lo hacemos?

Primero, supongamos que *V* tiene una longitud *N* que es potencia de 2 (1,2,4…), para mayor comodidad. Si no la tiene, podemos rellenarlo en el final con elementos neutros a nuestra operación. (es decir, elementos *u* tales que *Op(u,x)* = *x* para todo *x*).

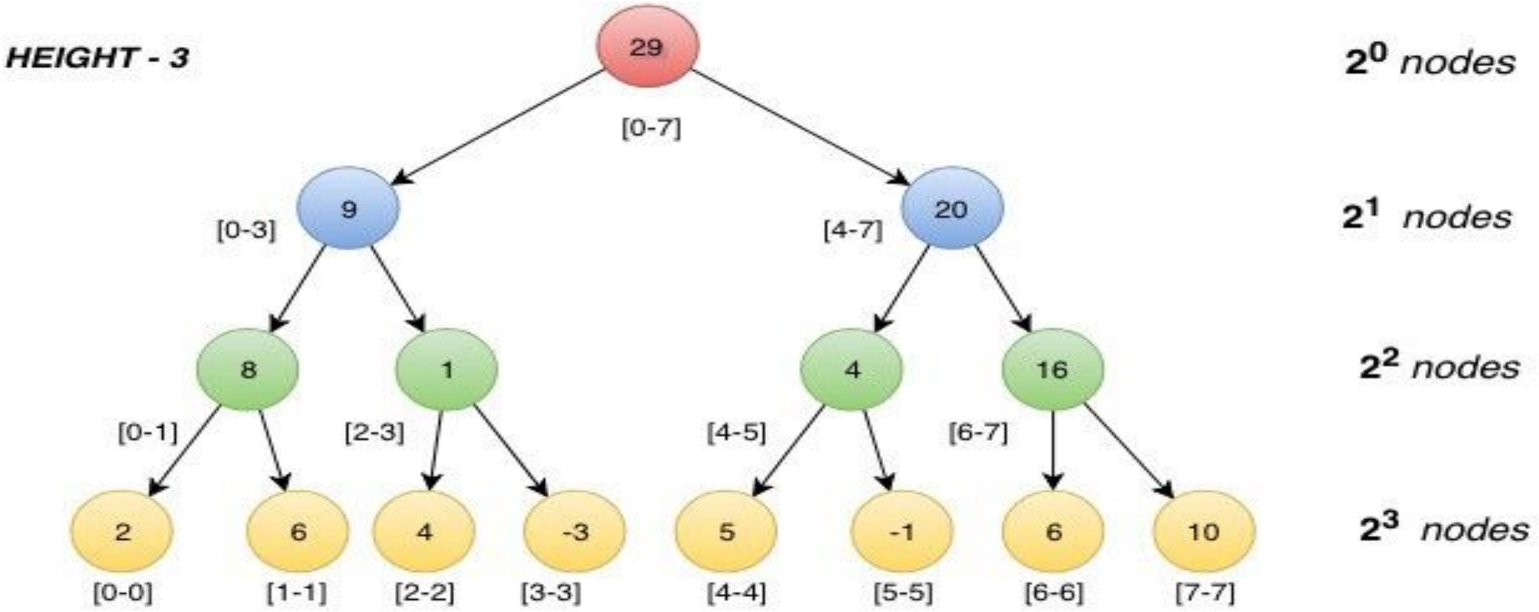
Vemos que podemos dividir a *V* en 2 rangos de tamaño *N/2*, 4 de tamaño *N/4*, etc. Esto será muy útil para armar la siguiente estructura.

¿Cómo lo hacemos?

Armamos un árbol binario completo, es decir un árbol jerarquizado donde cada nodo o es una hoja o tiene exactamente 2 hijos, y todas las hojas tienen la misma profundidad. (en este caso, $log_2(N)$)

Cada nodo representará un rango, es decir, contendrá la respuesta de aplicar *Op* en ese rango. La raíz representa al vector completo, uno de sus hijos representa al intervalo [*1;N/2*] y otro al intervalo [*N/2+1;N*].(Indexando en base 1), y así sucesivamente hasta las hojas que representan intervalos de tamaño 1, es decir elementos del vector.

¿Cómo lo hacemos?



Consultas (Queries)

Para hacer una consulta lo que haremos es ver los nodos empezando por la raíz. Al ver un nodo tenemos 3 posibilidades:

- Que el rango que el nodo representa esté totalmente contenido en la consulta, caso en el que devuelve su valor.
- Que el rango esté parcialmente contenido por la consulta, caso en el que preguntara a sus hijos y aplicará *Op* al resultado de estos.
- Que el rango no coincida en nada con la consulta, caso en el que devolverá el elemento neutro de *Op*

Consultas (Queries)

Notar que en 2 de los 3 casos la consulta se resuelve mirando solo ese nodo. Al ser la consulta un rango continuo, en cada nivel solo es posible que 2 nodos estén parcialmente contenidos en ella y propagan la pregunta a sus hijos (2 por cada nodo).

Por lo anterior, en cada nivel miraremos a lo sumo 4 nodos, y como la cantidad de niveles es $log_2(N)$, la complejidad de cada consulta queda $O(log(N)*O(Op))$.

Actualizaciones (Updates)

Para hacer la actualización o cambio, el camino será el inverso:

- Iremos a la hoja que representa el nodo que queremos actualizar y cambiaremos su valor.
- Al cambiar este elemento, solo cambian los valores de los rangos que lo contienen. Es decir, solo cambian los valores de los ancestros del nodo en nuestro árbol.
- Actualizamos los ancestros empezando por su padre, luego el padre de éste, y así hasta llegar a la raíz.

Actualizaciones (Updates)

Como la operación es asociativa, para saber su valor un nodo sólo necesita saber el valor de sus 2 hijos, y aplicar la operación a ellos. Así, actualizar cada nodo tiene complejidad $O(Op)$.

Como hay $log_2(N)$ niveles, nuestra hoja sólo tiene $log_2(N)$ ancestros que debemos actualizar. Por eso, la complejidad de una actualización queda $O(log(N) * O(Op))$

Inicializar la Estructura

Si bien puede inicializarse la estructura haciendo un Update en cada posición, con complejidad *O(N*log(N)*O(Op))*, también puede hacerse de forma más rápida en *O(N*O(Op))*.

Primero ponemos en las hojas el valor inicial de la posición que representan. Luego simplemente actualizamos los nodos empezando por el anteúltimo nivel, luego el nivel anterior, y así hasta llegar a la raíz. En ese caso, habremos actualizado una vez cada nodo, y como la cantidad total de nodos es $1+2+...+N/2+N = 2*N-1$, la complejidad resulta *O(N*O(Op))*

Implementación

Una implementación muy compacta del Segment Tree, que suelo usar mucho, utiliza los siguientes vectores, todos de tamaño *2*N* (la posición 0 no se utiliza)

- Dos vectores, L y R, indicando L_i y R_i el inicio y el fin del intervalo representado por el nodo i respectivamente.
- Un vector *st*, indicando st_i el valor guardado en el nodo i.

Implementación

En esta representación, el nodo 1 representa a la raíz del árbol. A su vez, los nodos *2*i* y *2*i+1* representan a los hijos izquierdo y derecho del nodo *i* respectivamente.

Las hojas están ubicadas en el intervalo [*N*;*2*N*-1], siendo que el valor de V_i está contenido en st_{N+i}

Implementación

El vector *st* ya sabemos cómo inicializarlo. Ahora, sabemos que el vector *R* contiene el índice del elemento más a la derecha (mayor) en el intervalo, y el *L* el índice del elemento más a la izquierda (menor).

Pero, lógicamente, el elemento más a la derecha de un intervalo es el elemento más a la derecha de su mitad derecha. Así, $R_i=R_{i*2+1}$. Análogamente, el más a la izquierda es el más a la izquierda de su mitad izquierda, por lo que $L_i=L_{i*2}$.

Implementación

A su vez, las hojas, ubicadas en $N+i$, representan el rango $[i;i]$. En resumen:

- El hijo izquierdo del nodo i es el nodo $i*2$ y el derecho es el $i*2+1$.
- Si $i>=N$, $L_i = R_i = i-N$ y $st_i = V_{i-N}$.
- Si $i<N$, $L_i = L_{i*2}$, $R_i = R_{i*2+1}$ y $st_i = Op(st_{i*2}, st_{i*2+1})$.

A su vez, si el nodo i no es la raíz, su padre es el piso de $i/2$, por lo que es muy fácil de acceder durante el Update.

Implementación

El siguiente código muestra cómo implementar un Segment Tree que de la suma en rango, con $N<=2^{18}$ ($2^{18} = 262.144$) : https://pastebin.com/vj2Yj7KP

Notar que la implementación siempre actúa como si $N = 2^{18}$, agregando todos los 0 a la derecha que sean necesarios. En este ejemplo se muestra una versión generalizada a cualquier operación *Op* que reciba 2 enteros y devuelva un entero, dejando la función *Op* sin completar: https://pastebin.com/hpFT2WZM. Notar lo poco que cambia de un ejemplo al otro.

Problemas

Algunos problemas que salen con Segment Tree:

- Los primeros 6 problemas de la sección “Range Queries” de CSES (https://cses.fi/problemset/list/)
- Problema “Dominando Operaciones” del Certamen Nacional de OIA 2019(http://www.oia.unsam.edu.ar/wp-content/uploads/2020/01/c3a19n3p1.pdf)
- En esta entrada en Code Forces se enlistan varios problemas agrupados por categoria https://codeforces.com/blog/entry/22616

Operaciones Asociativas

Como Anexo, se agregan ejemplos de operaciones que son asociativas. Se recuerda que *Op* es asociativa si $Op(Op(a,b),c) = Op(a,Op(b,c))$.

- Suma y producto de números, polinomios y matrices. (recordar que restar es sumar el inverso, y dividir multiplicar por el inverso)
- Concatenar strings/vectores
- Tomar el máximo común divisor / mínimo común múltiplo entre enteros.

Operaciones Asociativas

- Las operaciones de máximo y mínimo.
- Las operaciones bit a bit AND, OR y XOR. (operadores &, | y ^ respectivamente)
- Tomar el prefijo/sufijo común más largo entre strings/vectores.
- Unión e Intersección de conjuntos (notar la relación con las operaciones OR y AND respectivamente)
- Tomar el ancestro común menor entre nodos de un árbol (tema que veremos en profundidad más adelante)

Gracias por ver!

Por Lautaro Lasorsa, para el curso OIA - UNLaM 2020