Estructuras de dator y álgoritmos II
Tarea 1

Rubén Pérez Palacios Lic. Computación Matemática
Profesor: Dr. Carlos Segura González

13 de febrero de 2022

1. Segment Tree

El ST se implemento con una clase y en forma de arreglo.

Diferencias

Puesto que entre dos STs las únicas diferencias son el *tipo de dato* que almacenan los nodos y como se calcula el *valor* de un nodo a partir de sus hijos, estas dos deberían ser la únicas modificadas, lo primero se implemento con un *template* el cual representa el *tipo de dato* que almacenan los nodos, el segundo se explicará mas adelante. Ahora cuando los STs tiene propagación vaga la única diferencia entre sus propagaciones vagas son el *tipo de dato* de la actualización por propagar, como se actualiza el *valor* de un nodo según el *valor* de la actualización por propagar en ese nodo y como se propaga el valor de la actualización de un nodo a sus hijos, el primero de nuevo se implemento con un *template* el cual representa el *tipo de dato* de la propagación vaga, lo últimos se explicarán mas adelante.

Implementación

Campos

Se implemento una clase abstracta de un ST que define la base de uno, la cual es un $template class$ que recibe dos clases que representan *tipo de dato* que almacenan los nodos y las actualizaciones respectivamente, además contiene los siguientes datos:

- Un *vector node* : El conjunto de nodos.
- Un *vector lazy* : El conjunto de actualizaciones por propagar.
- Un *vector vec* : Los valores iniciales de los nodos(Los cuales se pueden ir actulizando si y sólo si las actualizaciones son puntuales).
- Un *vector pend* : El conjunto de indicadores de las actualizaciones pendientes de los nodos.
- $dflt\_node$: El valor *predeterminado* de un nodo, que además se usara como valor neutro al obtener el valor de un *nodo* a partir de sus hijos.
- $dflt\_lazy$: El valor *predeterminado* del lazy, que además se usara como valor neutro al actualizar el valor de un nodo y al propagar una actualización.
- $tval$: El valor por actualizar.

- tl, tr : Extremo izquierdo y derecho respectivamente de un rango, el cual se usa cuando se actualiza un rango, o se hace una consulta de un rango.
- N : El tamaño del arreglo que representa el ST.
- $init$: Indica si se proporciono valores iniciales para el arreglo que representa el ST, en caso de no ser así entonces se incializa con el valor $dflt\_node$.

Funciones

Funciones principales de la clase:

- $\_build$: Se encarga de construir e inicializar recursivamente el ST.
- $\_update$: Se encarga de actualizar el rango tl, tr recursivamente del ST.
- $\_query$: Se encarga de hacer una consulta del rango tl, tr recursivamente del ST.

Ahora si es momento de explicar como se implementaran las diferencias que quedaron pendientes. Para implementar como se calcula el *valor* de un nodo a partir de sus hijos es con una función que reciba como argumentos el valor de sus hijos y devuelva el valor resultado del nodo. Para implementar como se actualiza el *valor* de un nodo según el *valor* de la actualización por propagar en ese nodo es con una función que reciba como argumentos un nodo, el rango que representa y el *valor* de la actualización por propagar y devuelva el *valor* final del nodo. Por último para implementar como se propaga el valor de la actualización de un nodo a sus hijos es con una función que reciba como argumentos el *valor* de la actualización por propagar y el valor de la actualización pendiente del hijo, y devuelva el *valor* de la actualización por propagar del hijo. Todas las anteriores funciones son implementadas puramente virtuales (lo cual hace a la clase abstracta), por lo que el usuario debera heredar una clase de esta y proporcionar la implementación de estas, recuerde que no se puede hacer una instancia de una clase abstracta. Para poder usarla deberá hacer lo anterior.

Funciones de apoyo:

- $merge$: calcula el *valor* de un nodo a partir de sus hijos.
- $merge\_range$: actualiza el *valor* de un nodo según el *valor* de la actualización por propagar en ese nodo.
- $merge\_lazy$: propaga el valor de la actualización de un nodo a sus hijos.
- out_range : verifica si un rango l, r esta fuera del rango tl, tr.
- in_range : verifica si un rango l, r esta dentro del rango tl, tr.

Funciones de interface entre la clase y el usuario, las cuales son virtuales por si el usuario quisiera sobreescribirlas, de ser así es altamente recomendable que se llame a las funciones originales dentro de las nuevas:

- $build$: Recibe como parametros el tamaño N del arreglo que representa el ST, el, $dflt\_node$, $dflt\_lazy$, y opcionalmente el valor inicial del arreglo que representa el ST. Construye e inicializa el ST y los campos de este.
- $update$: Recibe como parametros el rango l, r por actualizar y val el *valor* de la actualización. Actualiza el rango l, r con el valor val.
- $query$: Recibe como parametros el rango l, r de la consulta. Devuelve el valor del rango l, r del ST.

Problemas

Editor-Codeforces

La solución de este problema es con un ST que almacena la suma de un rango, la mínima y máxima suma de los prefijos, donde (vale 1,) vale -1 y el resto vale 0 con propagación vaga. Ahora notemos que si todos las sumas de los prefijos son no negativas entonces significa que siempre huba más parentesis abiertos que cerrados y si la suma de todo es 0 entonces la cantidad de parentesis abiertos y cerrados es la misma por lo que es un *texto correcto*, por último notemos que la máxima suma de los prefijos es la mayor cantidad de parentesis abiertos anidados sin tener un parentesis cerrado correspondiente, por lo que la cantidad de colores el la máxima suma de los prefijos. Por lo tanto nuestro ST resuelve el problema.

SegmentTreeLazy-Hacerrank

La solución de este problema es con un ST que almacena la suma de un rango con propagación vaga. Para actualizar un rango l, r tenemos 4 casos:

1. $F(1)(E(0))$: el *valor* final del nodo es $r - l + 1(0)$, puesto que es la suma de $r - l + 1, 1's(0's)$.
2. $I(-1)$: el *valor* final del nodo es $r - l + 1 - node$(node es el valor del nodo), puesto que si la suma actual es *node* entonces al invertir todos los valores sería el complemento de esa suma.
3. (-2) : No se actualiza.

Los valores de las actualizaciones por propagar son:

- 1 : Fijar todos con 1.
- 0 : Fijar todos con 0.
- -1 : Intercambiar valores.
- -2 : No hacer nada(predeterminado).

Para la propagación de las actualizaciones tenemos -2 casos según el valor de la actualización:

1. $1(0)$: puesto que todos los valores serán reemplazados por $1's(0's)$ no importa que valor tenía antes la actualización pendiente del hijo, por lo que la actualización final del hijo es $1(0)$.
2. -1 : tenemos 4 casos según el valor de la actualización pendiente del hijo.
   a) $1(0)$: se intercambia por $0(1)$, por lo que la actualización final del hijo es $0(1)$.
   b) -1: como doble invertir es no hacer nada entonces la actualización final del hijo es -2.
   c) -1: como no hacía nada entonces la actualización final del hijo es -1.