OBSERVACIONES DEL LA PRACTICA

Ana Sofía Castellanos 202114167
Martín Santiago Galván Castro 201911013

Preguntas de análisis

1) ¿Qué relación encuentra entre el número de elementos en el árbol y la altura del árbol?

Normalmente, se puede encontrar la relación de número de elementos y la altura de un árbol binario bajo la siguiente ecuación:

$$2^h = n$$
$$floor(\log_2(n)) = h$$

Donde h se refiere al nivel del árbol y n se refiere al número de elementos dentro del árbol. Sin embargo, esta ecuación asume que el árbol esta balanceado. Aplicando esta expresión al caso de este laboratorio, se encuentra que no coincide.

$$2^{29} = 536870912 \neq 1177$$
$$floor(\log_2(1177)) = 10 \neq 29$$

De modo que, se afirma que el árbol no está balanceado ni completo debido a que es un BST y no un RBT. Dado esto, lo único que podemos afirmar acerca de la relación entre el número de elementos y los niveles, haciendo uso de las anteriores expresiones, son las siguientes inecuaciones.

$$2^h \geq n$$
$$floor(\log_2(n)) \leq h$$

Por otro lado, considerando que un árbol binario tiene un comportamiento logarítmico que relaciona la altura y la cantidad de elementos con la expresión mencionada arriba, se puede generalizar en tanto que la base del logaritmo expresa en realidad la cantidad de hijos de cada padre (2 en el caso de binario) pero que al generalizar quedaría una expresión del tipo:

$$\log_x(n) = h$$

Donde x representa la cantidad de hijos que tiene cada padre, n la cantidad de nodos total y h la altura del árbol. Para los datos que brinda el laboratorio se tiene que n = 1117 y h = 29.

$$\sqrt[29]{1117} = x$$

De lo cual se obtiene un valor de x de aproximadamente 1.27, lo cual indica que en promedio cada padre tiene 1.27 hijos. Esto, sin embargo, no es posible en tanto que los nodos son cantidades discretas, por lo cual este resultado de x nos indica que algunos de los nodos cuentan con

solamente 1 hijo y algunos otros nodos con 2 hijos, lo que hace que el árbol no este balanceado y cuente con una altura mayor a la obtenida con el logaritmo en base 2 de la cantidad de nodos.

2) ¿Si tuviera que responder esa misma consulta y la información estuviera en tablas de hash y no en un BST, cree que el tiempo de respuesta sería mayor o menor? ¿Por qué?

   El tiempo de respuesta para consultar crímenes en un rango de fechas sería mayor en una tabla de Hash puesto que en esta los elementos no se encuentran ordenados y se debería buscar entre todos los elementos para verificar que el crimen ocurrió en ese rango de fechas, es decir que tendría una complejidad de O(N).

   Caso contrario al BST el cual al estar ordenado al ingresar a la raíz puede descartar hasta la mitad de los elementos dependiendo de si el valor de la llave es menor al rango descarta los de la derecha o si el valor de la llave es mayor a la fecha mayor descarta los de la izquierda. En el peor caso, los rangos de fechas corresponden al menor valor del árbol binario y al mayor valor, allí la complejidad sería de O(N) puesto que tendría que añadir todos los valores del árbol a la lista de elementos que se encuentran en ese rango. Sin embargo, si el caso no es el peor descartará una cantidad significativa de elementos según el valor de la raíz en cada subárbol del árbol original lo cual hará que en el BST la búsqueda de elementos sea mucho mejor que en el Tabla de Hash.

3) ¿Qué operación del TAD se utiliza para retornar una lista con la información encontrada en un rango de fechas?

   La función que se usa en el modelo, cuando se crean listas de los valores que se encuentran es values(…) de la librería ADT orderedmapstructure.py. Esta función tiene como entrada un mapa, keylo, que es el límite inferior de las llaves que se buscan, y keyhi, que es el límite superior de las llaves que se buscan. La forma en que funciona esta función depende de otra librería, la cual se encuentra en el directorio DataStructures, orderedmapstructure.py, en donde se encuentra una función con el mismo nombre y opera de manera distinta dependiendo del tipo de árbol que se use, ya sea un Binary Search Tree (BST) o un árbol rojo-negro (RBT)

   Como el tipo de árbol que se utiliza en el laboratorio es un BST, declarado en el modelo en la función newAnalyzer, se analizará cómo funciona la función de este tipo. Esta función, a su vez, envía a otra librería de funciones dentro del mismo directorio. Esta vez, envía al archivo bst.py. En esta librería, se encuentra otra función con el mismo nombre de values(…) con las mismas entradas de la primera función mencionada. Esta función crea una lista simplemente encadenada y opera con una función llamada valuesRange(…)

   valuesRange(…) tiene como entrada una raíz, límite inferior de llaves, límite superior de llaves, una lista y una función de comparación. Esta función es una función recursiva que empieza comparando los límites con la raíz. Primero revisa si el límite inferior es estrictamente menor a la raíz, si lo es, se aplica valuesRange(…) al hijo izquierdo de la raíz. Luego, revisa si la raíz se encuentra en el rango keylo - keyhi, de ser así añade el valor de dicha raíz a la lista con la operación

lt.addLast(lstvalues,root[values]). Por último, se revisa si el límite superior es estrictamente mayor a la raíz, si lo es, aplica valuesRange(…) al hijo derecho. Al finalizar esta función sobre un árbol, se tendrá una lista de valores. El orden de estos valores viene heredado de las llaves. Por lo que, la lista de valores empezara con los valores cuyas llaves eran menores y terminaran por las mayores. Esta misma lista es la que devuelve la función values(…).