OBSERVACIONES DEL LA PRACTICA

Carlos Arturo Holguín Cárdenas - 202012385
Daniel Hernández Pineda – 202013995

Preguntas de análisis

1) **¿Qué relación encuentra entre el número de elementos en el árbol y la altura del árbol?**
   Sí puede existir relación entre el número de elementos del árbol y la altura, de modo que, incluso, el número de elementos permitiría determinar la altura del árbol. Esto sucede en el caso ideal, en donde el árbol se encuentre totalmente balanceado; no obstante, el árbol con el que se está trabajando no está balanceado. A continuación, el porqué:

   Por cómo está definida la función size() en bst.py, si un árbol tiene tamaño *k,* significa que sus niveles pueden contarse desde el 0 hasta *k-1*, debido a que en el tamaño también se tiene en cuenta la raíz. De esta manera, la siguiente fórmula corresponde al máximo de elementos que puede almacenar un árbol *n*-ario de altura *k*:

$$\#\max de\ elementos = \sum_{i=0}^{k-1} n^i$$

   Para el árbol de este laboratorio, se tendría entonces:

$$\#\max de\ elementos = \sum_{i=0}^{28} 2^i = 536870911$$

   Podemos ver que, por tanto, el árbol no se encuentra balanceado, pues contiene 319073 elementos de 536890911 posibles.

   El mínimo de elementos que puede contener un árbol de altura *k* es también calculable. Para esto, el árbol tendría que encontrarse de manera totalmente desbalanceada, donde la raíz correspondiera al mínimo o al máximo elemento que se almacena. Su estructura tendría que encontrarse dispuesta de manera secuencial, semejante a una lista encadenada ordenada cuyo primer o último elemento sería la raíz del árbol. En este caso, el árbol podría almacenar *k* elementos, un número significativamente menor al que realmente almacena el árbol de este laboratorio.

   De lo anterior es posible concluir que, si bien puede existir relación entre la altura de un árbol y el número de elementos que almacena, es difícil determinar la altura a partir del número de elementos (y viceversa) debido a la gran cantidad de maneras en que se puede construir un árbol. Para este ejemplo, encontramos un árbol que almacena alrededor del 0.06% del máximo de elementos que podría almacenar con la altura que tiene, pero, a su vez, almacena alrededor de 11000 veces el mínimo de elementos que podría almacenar con su altura.

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?**
   Al implementar esta función con tablas de hash, el tiempo de respuesta sería mayor. Esto se debe a que las llaves de una tabla de hash no están ordenadas, provocando que, para encontrar los valores de un rango de fechas, habría que evaluar todos los elementos del mapa para determinar cuáles están en el rango y cuáles no. Por otro lado, los BST tienen sus datos organizados, permitiendo encontrar la información en un rango de fechas sin tener que evaluar todos los elementos del mapa. Esto reduce considerablemente el tiempo de ejecución.

3) **¿Qué operación del TAD se utiliza para retornar una lista con la información encontrada en un rango de fechas?**
   Para poder retornar una lista con la información en un rango de fechas se recurre a la función **values()** del módulo orderedmap.py, invocando, en última instancia, la función **valuesRange()** del módulo bst.py.

   Esta función retorna todos los valores del árbol que se encuentren entre *KeyLow y KeyHigh* por medio de llamados recursivos a sí misma, aprovechando la virtud de que, para cualquier nodo de un bst, en su subárbol izquierdo hay valores menores y en su derecho hay valores mayores. Explicada brevemente, lo que hace *valuesRange()* es recorrer el mapa en búsqueda de algún elemento que pertenezca al rango, descartando todo subárbol derecho para el cual se sepa que *KeyHigh* no pertenece (es decir, que comphi<0). Una vez encontrado algún elemento del rango (complo<=0 y comphi>=0), se implementa la función sobre sus subárboles izquierdo y derecho con el fin de encontrar todos los valores que pertenezcan al rango, repitiendo recursivamente hasta llegar a *KeyLow* y *KeyHigh*.