**LABORATORIO NO. 8: TABLAS DE SÍMBOLOS ORDENADAS y BALANCEADAS**

# Objetivos

Incluir en el diseño de soluciones el uso de *Maps* o Tablas de símbolos ordenadas y utilizar los árboles binarios de búsqueda balanceados *(RBTs)* para su implementación

1. Analizar los órdenes de crecimiento y el desempeño de los *RBTs* como índices ordenados para la consulta de datos
2. Integrar los *RBTs* con las otras estructuras de datos vistas en el curso

# Desarrollo

## Cargar el ejemplo ISIS1225-SampleTree

## Ejecutar el Ejemplo

Tome nota del total de elementos en el árbol y la altura del BST reportada. Recuerde a que conclusión llegaron en el laboratorio anterior sobre estos valores.

**Número de elementos (BST):**

* Crímenes cargados: 319073
* Elementos en el árbol: 1177
* Menor Llave: 2015-06-15
* Mayor Llave: 2018-09-03

**Altura del árbol (BST):**

* Altura del árbol: 19

## Ejecutar el Ejemplo con un RBT

Salgan del programa y ahora editen el archivo model.py, específicamente donde se crea el BST.

    analyzer = {'crimes': None,

                'dateIndex': None

                }

    analyzer['crimes'] = lt.newList('SINGLE\_LINKED', compareIds)

    #analyzer['dateIndex'] = om.newMap(omaptype='BST',

    analyzer['dateIndex'] = om.newMap(omaptype='RBT',

                                      comparefunction=compareDates)

    return analyzer

**Número de elementos (RBT):**

* Crímenes cargados: 319073
* Elementos en el arbol: 1177
* Menor Llave: 2015-06-15
* Mayor Llave: 2018-09-03

**Altura del árbol (RBT):**

* Altura del arbol: 13

**Pregunta 1:** ¿Qué diferencia existe entre las alturas de los dos árboles (BST y RBT) ?, ¿por qué pasa esto?

Lo que se observa es que la altura cambia de 19 a 13, esto nos permite concluir, que el grado de complejidad de estos árboles mejora, toda vez que o(h), donde el h en los arboles RBT es 13, mucho menor que BST que es 19, por ende, es más eficiente el BST. Lo anterior, se debe a que RBT es considerado como un “self balancing tree”, y esto nos deja entender que es mas eficiente en el cargue de los datos en el árbol, mientras que el BST el cargue de los datos se hace de manera secuencial, y no cuida de ir haciendo balanceo del árbol, en este sentido, podemos terminar con complejidades mas granes que los RBTs.

## Modificar el requerimiento 1 del Reto 3

Utilicen la implementación del requerimiento 1 del reto 3 del laboratorio anterior (Lab 7) en el que utilizaron un BST, para averiguar la altura del árbol. Hagan la prueba con los datos del año 2016.

**Pregunta 2:** ¿Cuántos elementos tiene el árbol (size)? ¿Qué altura tiene el árbol (height)?

analyzer['dateIndex']=om.newMap(omaptype='BST',comparefunction=compareDates)

Accidentes cargados: 131254

Altura del arbol: 14

Elementos en el arbol: 344

Menor Llave: 2016-02-08

Mayor Llave: 2017-01-26

Tiempo de ejecución 7.9375 segundos

analyzer['dateIndex']=om.newMap(omaptype='RBT',comparefunction=compareDates)

Accidentes cargados: 131254

Altura del arbol: 11

Elementos en el arbol: 344

Menor Llave: 2016-02-08

Mayor Llave: 2017-01-26

Tiempo de ejecución 8.21875 segundos

## Modificar el requerimiento 1 del Reto 3

Cambien el requerimiento 1 del reto 3, desarrollado en el laboratorio anterior para que ahora utilice un RBT.

def newAnalyzer():

    # creo la lista para almacenar todos los accidentes, esto es cada fila del excel con sus 49 campos

    # crea un Cataolo de Analyzer, una lista para los accidentes y una Mapa Ordenado para las fechas

    analyzer={ 'accidents':None,

               'dateIdndex':None

            }

    analyzer['accidents']=lt.newList('SINGLE\_LINKED',compareIds)

    #analyzer['dateIndex']=om.newMap(omaptype='BST',comparefunction=compareDates)

    analyzer['dateIndex']=om.newMap(omaptype='RBT',comparefunction=compareDates)

    return analyzer

**Pregunta 3:** Qué tan difícil fue hacer el cambio de una estructura de datos por otra? ¿Cuántas líneas de código tuvieron que modificar para hacer el cambio?

La dificultad bajo las estructuras definidas en el código, realmente es cambiar solo el parámetro de omaptype= 'BST' por omaptype= 'RBT'.

**Pregunta 4:** Cuántos elementos tiene el árbol? ¿Qué altura tiene el árbol? ¿Qué puede concluir sobre las alturas de los árboles cuando se usa un BST o un RBT?

Ahora prueben con el mismo archivo de datos (año 2016) la nueva estructura de datos (RBT)

analyzer['dateIndex']=om.newMap(omaptype='RBT',comparefunction=compareDates)

Accidentes cargados: 131254

Altura del arbol: 11

Elementos en el arbol: 344

Menor Llave: 2016-02-08

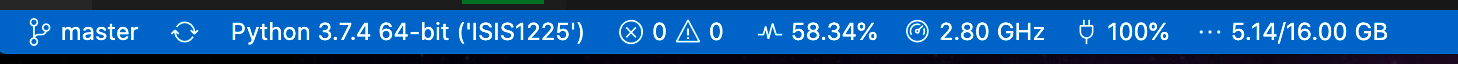
Mayor Llave: 2017-01-26

Tiempo de ejecución 8.21875 segundos

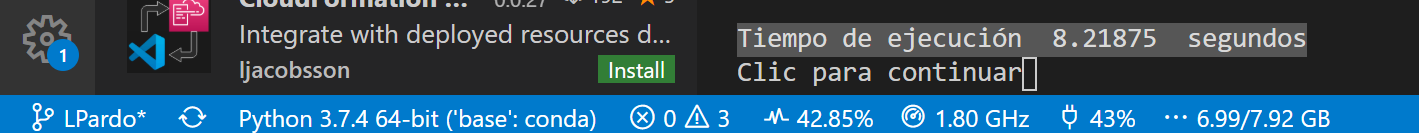
Lo que se observa es que la altura cambia de 14 a 11, esto nos permite concluir, que el grado de complejidad de estos árboles mejora, toda vez que o(h), donde el h en los arboles RBT es 11, mucho menor que BST que es 14, por ende, es más eficiente el BST. Lo anterior, se debe a que RBT es considerado como un “self balancing tree”, y esto nos deja entender que es más eficiente en el cargue de los datos en el árbol, mientras que el BST el cargue de los datos se hace de manera secuencial, y no cuida de ir haciendo balanceo del árbol, en este sentido, podemos terminar con complejidades más granes que los RBTs. De otro lado, se tomo el “time proccess”, observando que el RBT consume mas tiempo en razón a los ciclos de proceso que consume para hacer el balanceo del árbol, no obstante, una vez cargado si reduce la complejidad de una manera importante.

## Utilizar el conjunto completo de datos con el requerimiento 1 del Reto 3

En la barra inferior de VSCode verán la información de los recursos de su computador:











Accidentes cargados: 131254

Altura del arbol: 14

Elementos en el arbol: 344

Menor Llave: 2016-02-08

Mayor Llave: 2017-01-26

Tiempo de ejecución 8.109375 segundos

Ahora vamos a realizar las corridas con RBTÑ



Accidentes cargados: 131254

Altura del arbol: 11

Elementos en el arbol: 344

Menor Llave: 2016-02-08

Mayor Llave: 2017-01-26

Tiempo de ejecución 10.984375 segundos

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Type | Tiempo | | Memoria | Procesador |
| 1 | RBT | 8,9 | | 6,79 | 22,00% |
| 2 | RBT | 8,42 | | 6,34 | 23,00% |
| 3 | RBT | 8,31 | | 6,63 | 21,00% |
| 4 | RBT | 8,46 | | 6,37 | 20,17% |
| 5 | RBT | 8,39 | | 6,69 | 18,56% |
|  | |  |  |  |  |
| Average -> | |  | 8,496 | 6,564 | 21% |

**Pregunta 5: Existe diferencia en el consumo de memoria? ¿Pueden proponer una** **relación entre el total de datos cargados y la memoria utilizada?**

A continuación, se presentan los resultados, con el archivo del 2016.



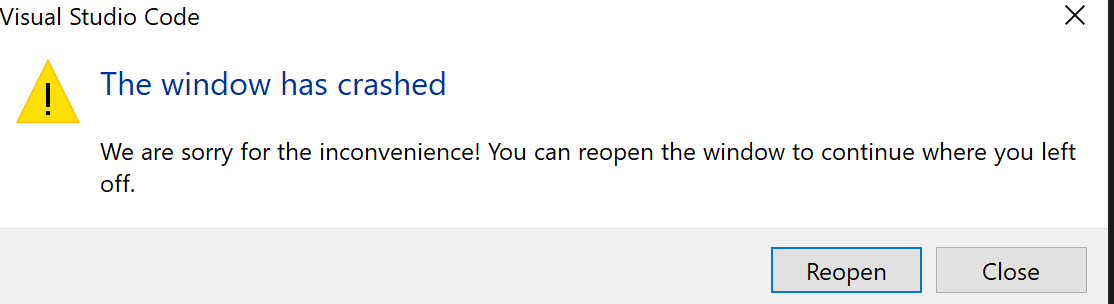
A continuación, se presentan los resultados, con el archivo de todos.











|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Type | Volumen de  Registros (CSV) | Tiempo Segundos | Memoria  (GB) | Memoria  Base (GB) | % Memoria  Usada GB | Procesador | % Memoria Usada GB / No. de Registros | Tiempo Seg / No. de Registros |
| 1 | RBT |  |  |  |  |  |  |  |  |
| 2 | RBT |  |  |  |  |  |  |  |  |
| 3 | RBT |  |  |  |  |  |  |  |  |
| 4 | RBT |  |  |  |  |  |  |  |  |
| 5 | RBT |  |  |  |  |  |  |  |  |
| Average -> | | |  |  |  |  |  |  |  |