**LABORATORIO NO. 7: TABLAS DE SÍMBOLOS ORDENADAS**

# Objetivos

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

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

# Desarrollo

En los grupos previamente definidos sigan los siguientes pasos para el laboratorio de hoy.

## Cargar el ejemplo ISIS1225-SampleTree

Asegúrense que todos los miembros del equipo tienen acceso al ejemplo. Para ello utilicen el URL <https://github.com/ISIS1225DEVS/ISIS1225-SampleTree> y sigan el proceso acostumbrado: Fork del proyecto a su espacio en GitHub y clone a su máquina local.

Del directorio de Datos en la sección unificada de Sicuaplus, descargue los datos necesarios para que funcione el ejemplo, en particular el archivo *“crime-utf8.csv”* de la sección Boston-Crimes. Copie dicho archivo en el directorio Data del proyecto de ejemplo en su máquina local.

Este ejemplo se basa en el conjunto de datos de Kaggle, llamado Crimes in Boston (<https://www.kaggle.com/AnalyzeBoston/crimes-in-boston>) que tiene datos de crímenes en la ciudad de Boston, entre los años 2015 a 2018.

## Ejecutar el Ejemplo

Cuando tenga los datos de prueba, ejecute el programa, desde el archivo view.py y úselo para familiarizarse con su funcionamiento. Ejecute primero la opción 1 que inicializa las estructuras de datos, luego la opción 2, para cargar la información.

Al cargar la información, verá el número de registros cargados, la menor llave encontrada, la mayor llave encontrada, la altura del árbol *(BST)* y el número de nodos en el árbol *(BST).* El árbol guarda un índice con parejas llave-valor, donde la llave es la fecha del crimen *(YYYY-MM-DD)* y el valor, los crímenes ocurridos ese día.

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ahora, ejecuten la opción 3, que retorna todos los crímenes encontrados en un rango de fechas. Utilice fechas de inicio y final en formato YYYY-MM-DD y dentro de las fechas del archivo (2015 a 2018).

Por ejemplo, como fecha de inicio puede dar: 2017-01-01 y como fecha final 2018-01-01

Verá como resultado el total de crímenes en ese rango de fechas.

**Pregunta 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?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ahora, ejecuten la opción 4, que retorna todos los crímenes de un tipo determinado y en una fecha específica.

Por ejemplo, como fecha de puede dar: 2017-01-01 y como tipo de crimen: “*Vandalism”*.

Deberá ver como resultado el total de crímenes en esa fecha y de ese tipo.

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Inspeccionar el código de ejemplo

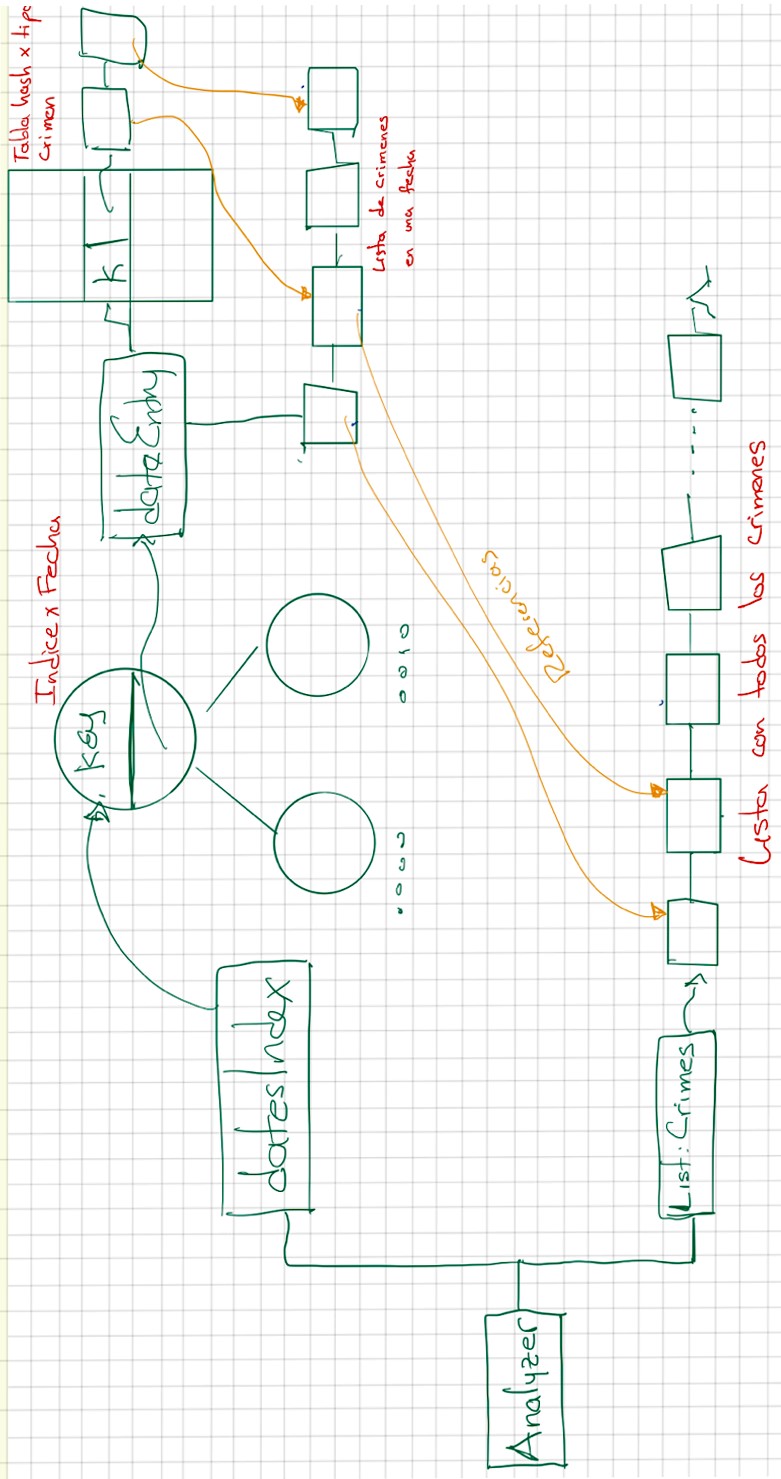
En el archivo *model.py* del ejemplo inspeccione el código para entender los TADs y estructuras de datos utilizados.

Lo primero que deben estudiar, es cómo está representado un analizador de crímenes. En el ejemplo el analizador tiene dos elementos: Una lista con todos los crímenes cargados y un índice *(dateIndex)* representado con un mapa ordenado, que en este caso es un *BST.*

Cada nodo del árbol tiene una pareja llave-valor. La llave es una fecha y el valor, a su vez, es de nuevo una pareja de elementos: a) La lista de todos los crímenes ocurridos en dicha fecha y una tabla de hash, con una lista de los crímenes en esa fecha, agrupados por tipo de crimen (la llave para la tabla de hash es el tipo de crimen).

Revisen el código hasta entender cómo se construye el árbol y cómo se usan el árbol y la tabla de hash para solucionar el problema.

La siguiente imagen ilustra las estructuras de datos utilizadas.



## Implementar el requerimiento 1 del Reto 3

Una vez que han entendido el ejemplo, descarguen el esqueleto del reto 3 en el siguiente URL (fork y clone).

### Elaborar el menú de opciones para el reto 3

En la vista *(view.py)* complete las opciones del menú para cumplir con los requerimientos del reto.

### Implementar el requerimiento 1

Implementen la lectura del archivo del reto 3 y utilicen las estructuras de datos necesarias para almacenar los accidentes. Se debe utilizar un árbol *(BST)* para construir un índice sobre la información de accidentes. Decidan qué otras estructuras de datos pueden necesitar para responder el requerimiento 1.

Se debe conservar el esquema Modelo-Vista-Controlador en el desarrollo del reto. Utilice el esqueleto provisto en este enlace (URL: <https://github.com/ISIS1225DEVS/Reto3-202020-Template>) como base de su implementación.

## Compartir el producto de la practica con los evaluadores

El resultado de este laboratorio es la implementación del requerimiento 1 del reto 3. Para entregar exitosamente sus resultados de este laboratorio, por favor recuerde las siguientes indicaciones:

1. Crear un nuevo repositorio nombrándolo con el formato ***EDA-2020-20-Lab-07-SEC-<<Número de la sección>>-GRUPO-<<Número del grupo>>.***
2. Invitar a los monitores del laboratorio asignados.
3. Incluir en el **README** del repositorio los datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
4. Incluir en la carpeta *Docs* un documento en formato PDF que indique lo siguiente:
   1. Datos completos de los integrantes del grupo (nombre completo, correo Uniandes y código de estudiante).
   2. Las tres preguntas del laboratorio con sus respectivas respuestas, en un documento PDF llamado “*respuestas-lab7.pdf*”, marcado con el nombre de los integrantes del grupo. Dentro del documento la respuesta a las preguntas:

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

La relación que existe es que la altura ideal de un árbol balanceado es , siendo n el número de elementos en el árbol.

En el ejemplo podemos observar que el árbol tiene una altura de 13 con 1177 elementos, en este caso podemos observar que el árbol no está del todo balanceado, pero se acerca a la altura que se esperaría ya un árbol RBT no siempre está perfectamente balanceado.

**Pregunta 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 sería mayor ya que a diferencia de las tablas de hash, los RBT están ordenados y permiten consultar valores en un rango, en un tiempo máximo de respuesta de O(log(n).

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

Para retornar una lista con la información encontrada en un rango de fechas se utiliza la función recursiva values la cual Retorna todas los valores del árbol que se encuentren entre [keylo, keyhi] siendo keylo el límite inferior y keyhi el límite superior del rango.

Recuerden que cualquier documento solicitado durante las actividades debe incluirse en el repositorio GIT y que solo se calificará hasta el último **COMMIT** realizado dentro de la fecha límite del miércoles 30 de septiembre de 2020, antes de la media noche (11:59 pm).