Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 6

Taller 6: Árboles Rojo-Negro

Objetivos

- Estudiar, implementar, probar y documentar una estructura de árbol binario balanceado
- Utilizar adecuadamente herramientas para el desarrollo de software en equipos

Lectura Previa

Estudiar la teoría de árboles balanceados 2-3 y Rojo-Negro. Consultar la sección 3.3 *Balanced Search Trees* del libro guía *Algorithms* de Sedgewick y Wayne.

Las Fuentes de Datos

bogota_cadastral.json
Archivo JSON con todas las zonas que define Uber.
Formato:

```
{
"type":"FeatureCollection",
"features":[
{
"type":"Feature",
"geometry":
       {"type":"MultiPolygon","coordinates":[[[[-74.200295,4.617249],[-74.200285,4.617248],[-
       74.200277,4.617248],[-74.200257,4.617246] …]]],
"properties":{
       "cartodb_id":12,
       "scacodigo":"004575",
       "scatipo":0,
       "scanombre":"LOS LAURELES",
       "shape_leng":0.02774133557,
       "shape_area":0.00003682838,
       "MOVEMENT_ID":"1",
       "DISPLAY_NAME":"LOS LAURELES, 004575 (1)"}
},{
...
},
...
```

]
}

En la colección de features, existe un atributo geometry con los puntos (cada uno representado como un arreglo con longitud y latitud) que forman la frontera de la zona, y un atributo properties con las propiedades de la zona. Cada zona tiene:

- MOVEMENT_ID: Id de la zona.
- scanombre: Nombre de la zona.
- shape_leng: perímetro de la zona. Si se multiplica por 100 da en kilómetros.
- shape_area: área de la zona. Si se multiplica por 10,000 da en kilómetros cuadrados.

IMPORTANTE: La propiedad MOVEMENT_ID de una zona corresponde a los atributos sourceid y dstid en los viajes.

Para la lectura de archivos JSON se recomienda usar el API (librería) GSON. Consultar https://github.com/google/gson Para descargas del API (librería extensión .jar) consulte: https://maven-badges.herokuapp.com/maven-central/com.google.code.gson/gson Para documentación del API consulte: http://www.javadoc.io/doc/com.google.code.gson/gson

Lo que su grupo debe hacer (parejas)

Parte 1 – Configuración Repositorio

1. Cree en Bitbucket/GitHub un repositorio llamado T6_201920. Al momento de crearlo recuerde la URL que se muestra en la parte superior derecha de la página de bitbucket: Por ejemplo

   Repository_url = https://login-usuario@bitbucket.org/login-usuario/T6_201920.git

   donde login-usuario corresponde a su login en Bitbucket.

2. Cree el README del repositorio donde aparezcan los nombres y códigos de los miembros del grupo de trabajo.
3. Realice el procedimiento para crear el directorio en su computador de trabajo para que relacione este directorio con el repositorio remoto T6_201920.
4. Descargar los archivos descritos en la fuente de datos y agregarlos a la carpeta data.

Parte 2 – Desarrollo

5. Implemente la estructura de datos RedBlackBST<K, V>, la cual representa un árbol binario ordenado, balanceado Rojo-Negro (con enlaces rojos a la izquierda) donde K representa el tipo de las llaves a agregar y V representa el tipo de los valores asociados a cada llave. La estructura de datos a implementar está compuesta por las siguientes operaciones:

Operación	Descripción
`RedBlackBST()`	Crea un árbol vacío
`int size()`	Retornar el número de parejas [Llave,Valor] del árbol
`boolean isEmpty ()`	Informa si el árbol es vacío
`V get(K key)`	Retorna el valor V asociado a la llave key dada. Si la llave no se encuentra se retorna el valor *null*.
`int getHeight(K key)`	Retorna la altura del camino desde la raiz para llegar a la llave key (si la llave existe). Retorna valor –1 si la llave No existe.
`boolean contains(K key)`	Indica si la llave key se encuentra en el árbol
`void put(K key, V val)`	Inserta la pareja [key, val] en el árbol respetando el balanceo RedBlack. Si la llave ya existe se reemplaza el valor. Si la llave key o el valor val es *null* se debe lanzar una Exception.
`int height()`	Retorna la altura del árbol definida como la altura de la rama más alta (aquella que tenga mayor número de enlaces desde la raíz a una hoja).
`K min()`	Retorna la llave más pequeña del árbol. Valor *null* si árbol vacío
`K max()`	Retorna la llave más grande del árbol. Valor *null* si árbol vacío
`boolean check()`	Valida si el árbol es Binario Ordenado y está balanceado Rojo-Negro a la izquierda. Hay que validar que: (a) la llave de cada nodo sea mayor que cualquiera de su sub-árbol izquierdo, (b) la llave de cada nodo sea menor que cualquiera de su sub-árbol derecho, (c) un nodo NO puede tener enlace rojo a su hijo derecho, (d) No puede haber dos enlaces rojos consecutivos a la izquierda. Es decir, un nodo NO puede tener un enlace rojo a su hijo izquierdo y su hijo izquierdo NO puede tener enlace rojo a su hijo izquierdo, (e) todas las ramas tienen el mismo número de enlaces negros.
`Iterator <K> keys()`	Retorna todas llaves del árbol como un iterador
`Iterator<V> valuesInRange(K init, K end)`	Retorna todos los valores V en el árbol que estén asociados al rango de llaves dado. Por eficiencia, debe intentarse No recorrer todo el árbol.
`Iterator<K> keysInRange(K init, K end)`	Retorna todas las llaves K en el árbol que se encuentran en el rango de llaves dado. Por eficiencia, debe intentarse No recorrer todo el árbol.

6. Diseñe escenarios de prueba para probar los diferentes métodos de la estructura.
7. Construya casos de prueba en JUnit para verificar y validar todos los métodos del API.

8. **Requerimiento 1: Cargar datos.**

Cargar las zonas directamente a un Árbol Balanceado RedBlackBST como una tupla (Llave, Valor) donde la ***Llave*** corresponde al MOVEMENT_ID de la zona y el ***Valor*** el resto de la información de la zona: scanombre, shape_leng, shape_area, puntos (número de puntos geográficos).

Una vez realizada la carga de las zonas debe informarse:

- El número total de zonas
- El valor mínimo y máximo de MOVEMENT_ID

9. **Requerimiento 2: Consultar una zona por id.**
   Dado un MOVEMENT_ID de una zona que ingresa el usuario se busca su información asociada en el Árbol Balanceado RedBlackBST construido. Mostrar el nombre de la zona, su perímetro, su área y el número de puntos (coordenadas geográficas) que definen el perímetro de la zona.

10. **Requerimiento 3: Consultar las zonas con un id en un rango específico.**
    Dado un rango de MOVEMENT_IDs (mov_id_inferior y mov_id_maximo) que ingresa el usuario se buscan todas las zonas en el rango. De cada zona se muestra su MOVEMENT_ID, nombre, perímetro, área y el número de puntos (coordenadas geográficas) que conforman el perímetro de la zona. Las zonas deben mostrarse de menor a mayor MOVEMENT_ID.

11. **Documento análisis**
    Incluir el documento en la carpeta docs de su proyecto Eclipse.

    El documento debe incluir:

    Tabla

a. Total nodos en el árbol Red-Black	
b. Altura (real) del árbol Red-Black	
c. Altura promedio de las hojas del árbol Red-Black	
d. Altura Teórica mínima de un árbol Red-Black con el número de nodos	
e. Altura Teórica máxima de un árbol Red-Black con el número de nodos	
f. Altura Teórica mínima de un árbol 2-3 con el número de nodos	
g. Altura teórica máxima de un árbol 2-3 con el número de nodos	

Comentarios Comparativos

a. Comentario de análisis de la altura de su árbol (real) Red-Black (11.b.) con respecto a las alturas de los árboles 11.d., 11.e., 11.f. y 11.g. ¿Es menor? ¿Es mayor? ¿Es igual?

b. Comentario de cómo es el promedio de la altura de su árbol Red-Black (11.c) con respecto a las alturas de los árboles 11.d., 11.e., 11.f. y 11.g.

Entrega

1. Para hacer la entrega del taller usted debe agregar los usuarios de los monitores y su profesor (modo Read) a su repositorio Bitbucket/Github, siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
2. Si No da acceso a su repositorio a los monitores y al profesor, el taller No podrá ser calificado.