Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 7

Taller 7: 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.

Carga de Información

Descargue la información del archivo de infracciones para el mes de Enero, Febrero, Marzo, Abril, Mayo y Junio de 2018 (formato JSON) que se encuentra en el archivo data.zip. Para responder a los requerimientos presentados más adelante, usted deberá cargar la información de todos los archivos .JSON correspondientes al primer semestre del año 2018.
Solo es permitido leer una vez la información de los archivos.

Lo que su grupo debe hacer (parejas)

Parte 1 – Trabajo en casa

1. Cree en bitbucket un repositorio llamado T7_201910. 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/T7_201910.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 T7_201910.
   El trabajo debe repartirse entre ambos estudiantes del grupo.

4. 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.

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

Parte 2 – Trabajo en clase

Construir la aplicación a partir de los siguientes requerimientos funcionales:

1. Inicialmente cargar los archivos JSON de los meses del primer semestre 2018. Los archivos los encuentra en el archivo data.zip incluida en el material del taller.
   Cada infracción debe agregarse directamente a un Árbol Balanceado RedBlackBST como una tupla (Llave, Valor) donde la ***Llave*** corresponde al ObjectId de la infracción y el ***Valor*** al resto de la información de la infracción.
   Una vez realizada la carga de las infracciones debe informarse:
   - El total de infracciones del semestre.
   - El número de infracciones de cada mes cargado.

2. Definir un menú que permita resolver los siguientes requerimientos:
   - Consultar la información asociada a un valor ObjectId. El usuario ingresa el ObjectId.
     Como resultado se debe mostrar el Location, AddressId, StreetSegId, XCoord, YCoord y TicketIssueDate de la infracción respectiva.
   - Consultar los ObjectIds de las infracciones que se encuentren registrados en un rango dado por [ObjectId menor, ObjectId Mayor]. El usuario ingresa los valores del rango de consulta con ObjectId menor <= ObjectId Mayor. Por cada ObjectId existente en el rango se debe mostrar su ObjectId, Location, AddressId, StreetSegId, XCoord, YCoord y TicketIssueDate.
     Se espera hacer una búsqueda eficiente de los ObjectIds de las infracciones en el rango (Debe intentarse No recorrer todo el árbol).
   - Terminar la ejecución de la aplicación. Permite salir del menú de las opciones de la aplicación.
3. Preparar un documento de texto Analisis.txt donde se incluya la siguiente información del árbol Red-Black construido:
   a. Total de nodos (infracciones) en el árbol
   b. Altura (real) del árbol
   c. Altura promedio para buscar una infracción existente en el árbol
      Ayuda: Calcular el promedio de las alturas de los ObjectIds en el árbol construido
   d. Altura teórica del árbol Red-Black más alto que se podría construir con el mismo número de infracciones
   e. Altura teórica del árbol Red-Black más bajo que se podría construir con el mismo número de infracciones
   f. Altura teórica del árbol 2-3 más alto que se podría construir con el mismo número de infracciones
   g. Altura teórica del árbol 2-3 más bajo que se podría construir con el mismo número de infracciones
   h. Comentario de análisis de la altura de su árbol (real) Red-Black (2.b) con respecto a las alturas de los árboles 2.d, 2.e, 2.f. y 2.g. Es menor? Es mayor? Es igual?

i. Comentario de como es el promedio de la altura de su árbol Red-Black (2.c) con respecto a las alturas de los árboles 2.d, 2.e, 2.f. y 2.g

Entrega

1. Para hacer la entrega del taller usted debe agregar a su repositorio los usuarios de los monitores y su profesor, siguiendo las instrucciones del documento “Guía Creación de Repositorios para Talleres y Proyectos.docx”.
2. Entregue su taller por medio de BitBucket. Recuerde, si su repositorio no tiene los accesos o si está vacío, su taller no será calificado por más de que lo haya desarrollado.