Universidad de los Andes

Ingeniería de Sistemas y Computación
ISIS1206 – Estructura de Datos
Taller 4

Taller 4: Árboles Binarios Ordenados

Objetivos

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

Fecha Límite de Entrega

14 de Octubre, 11:59 p.m.

Lectura Previa

Estudiar la teoría de árboles binarios ordenados. Consultar la sección 3.2 *Binary Search Trees* del libro guía *Algorithms* de Sedgewick y Wayne.

Las Fuentes de Datos

Se van a usar los datos a utilizar en el Reto 3. Estos datos corresponden al Proyecto US Accidents (Version 3) (URL: https://www.kaggle.com/sobhanmoosavi/us-accidents/version/3) en formato CSV. Se cuenta con conjuntos de accidentes reportados de los años 2016 al 2019. Estos datos los encuentran en el aula Sicua+ Unificada del curso, sección Datos, sección USA Accidents.
Se adjunta un documento (Excel) con los principales campos de información reportados en cada accidente.

Lo que su grupo debe hacer

Parte 1 – Configuración Repositorio

1. Cree en GitHub un repositorio llamado T4_202020.
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 que acaba de crear.
4. Descargue los datos descritos en la sección fuentes de datos y cópielos en la carpeta **data** del repositorio local. Por limitación de espacio, los datos NO hay que copiarlos en su repositorio remoto en Github.

Parte 2 – Desarrollo

1. Definir el TAD TablaSimbolosOrdenada<K extends Comparable<K>,V> (Interface)

El tipo K representa la clase que define las llaves de la Tabla y debe ser Comparable<K>.
El tipo V representa la clase que define el valor asociado a cada llave. V puede ser o no Comparable<V>.
Las operaciones básicas del TAD Tabla de Símbolos Ordenada son:

Operación	Descripción
`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. Si la llave ya existe se reemplaza el valor.
`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
`Lista<K> keySet()`	Retorna las llaves del árbol. Para su implementación en BST o RBT deben retornarse usando un recorrido en Inorden.
`Lista<K> keysInRange(K init, K end)`	Retorna todas las llaves K en el árbol que se encuentran en el rango de llaves dado. Las llaves en el rango deben retornarse en orden ascendente. Por eficiencia, debe intentarse No recorrer todo el árbol.
`Lista<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.

2. Implementar la Estructura de Datos BinarySearchTree<K,V> o BST<K,V> como una Tabla de Símbolos Ordenada

Esta estructura representa un árbol binario ordenado donde K representa el tipo de las llaves a agregar y V representa el tipo del valor asociado a cada llave.

La declaración en Java de la clase BST debe ser de la forma:
class BST<K extends Comparable<K>, V> implements TablaSimbolosOrdenada<K, V>

3. **Pruebas JUnit para el BinarySearchTree / BST**

Construya casos de prueba en JUnit para validar la correcta implementación de todos los métodos de la Estructura de Datos BST.

4. **Cargar datos por Fecha Inicial de los Accidentes**

   Cargar los accidentes a un Árbol Binario Ordenado como tuplas (Llave, Valor) donde la ***Llave*** corresponde a la Fecha Inicial (Start_Time) del accidente y el ***Valor*** al conjunto de accidentes en esa fecha. Las fechas deben representarse en formato AAAA-MM-DD con AAAA el año, MM el número del mes y DD el número del dia. Para el taller hay que cargar los datos de un año.
   Se debe poder cargar cualquiera de los años reportados.

   Una vez realizada la carga debe informarse:
   - El número total de accidentes en el año cargado
   - El número de llaves ingresadas en el BST
   - La altura del árbol BST
   - El valor mínimo de la llave y el valor máximo de la llave ingresada al BST

5. **Resolver el Requerimiento 1 del Reto 3**

   Reportar el total de accidentes para una fecha dada en formato AAAA-MM-DD. Adicionalmente hay que detallar el número de accidentes en dicha fecha por gravedad (severity).
   Si no existe la fecha de consulta debe informarse que No se encontró la fecha.

Entrega

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