Proyecto #1: Análisis temporal de un árbol AVL.

Fecha de Entrega: 19 de setiembre, 2018, a las 11:59pm.

Instrucciones Generales:

- Se debe entregar un enlace dirigido al repositorio en GitHub.
- El proyecto debe ser programado usando C++ 14.
- El proyecto debe tener un “Readme” con información relevante para ejecutar el programa final.
- El proyecto debe tener una estructura de archivos específica, donde se incluya una carpeta include, src, test, misc, y las que usted considere necesarias.
- Un “Makefile” debe ser implementado de la siguiente manera:
    - Make: Compilación general del proyecto.
    - Make unittest: Compilación de las pruebas unitarias.
    - Make run: Ejecución del programa principal.
    - Make run_unittest: Ejecución de las pruebas unitarias.

Descripción: Los AVL trees son estructuras de datos de tipo árbol, binarios y balanceados. Estos se usan cuando los tiempos de ejecución de funciones como “Insert” y “Remove” deben tener complejidad logarítmica. La estructura cumple además una característica especial llamada auto-balanceo, o sea que siempre se mantendrá balanceada no importa el tamaño ni las funciones que se usen sobre ella.

El propósito de este proyecto es poder implementar una base de datos muy simplificada, donde cada nodo del árbol sea un objeto que contenga los atributos: "Nombre", y “Cédula”. Estos atributos serán del tipo "string" y “unsigned int (32 bits)” respectivamente. El atributo que definirá el orden en el árbol será la cédula.

La base de datos se construirá a partir de un archivo con alrededor de 10000 líneas (luego de 5000, luego de 1000, luego de 100, y luego de 10), donde cada línea tendrá un nombre y una cédula en el siguiente formato:

Fernando Fernández Fernández, 812341234

Marco Antonio Marquez Marquez, 199990000

Se debe construir el árbol AVL a partir de estas listas.

Pasos:

1- Investigación: Se debe leer documentación acerca de la estructura de datos llamada árbol AVL (AVL tree en inglés). Se debe estudiar específicamente las funciones para:

Crear el árbol (avl_tree_create)

Insertar un nuevo nodo (avl_tree_insert)

Remover un nodo (avl_tree_remove)

2- Implementación de la estructura de datos: Se debe crear un "header file" (.h o .hpp) o encabezado, donde vengan las declaraciones de las funciones para crear, insertar, y remover antes mencionadas; así como funciones misceláneas como obtención de la cantidad de nodos en el árbol, y otras que usted considere necesarias. También deben venir los “structs” o “classes” que usted considere necesario.

avl_tree_create

avl_tree_insert

avl_tree_remove

avl_tree_get_size

avl_tree_get_max_height

Este header file debe estar acompañado de los archivos ".cpp" donde estén programadas cada una de las funciones anteriormente mencionadas. Se recomienda utilizar un struct para crear la plantilla genérica de los nodos del árbol.

El header file debe poder ser incluido en cualquier otro programa mediante la línea:

#include "include/data_structures/avl_tree.hpp "

3- Implementación de la infraestructura de pruebas.

Se debe probar la implementación del árbol AVL mediante pruebas unitarias (unit tests). Se debe probar tanto positivamente como negativamente (dar entradas inválidas a las funciones y recibir los errores correspondientes). Mínimo deberá haber 4 pruebas unitarias del código del árbol AVL.

Ejemplo de prueba:

Crear un árbol con una lista vacía de números, esperar error ningún error.

Crear un árbol con la misma raíz (root node), esperar error.

Insertar diez nodos, esperar ningún error.

Obtener tamaño, esperar que sea 10.

Remover diez nodos, obtener tamaño y revisar que sea 0.

Remover un nodo, esperar error.

4- Implementación del programa principal.

Se debe leer un archivo (/misc/input/name_id_list.txt) con los datos de todas las personas que deben ser incluidas en la base de datos. Posteriormente se debe crear el árbol AVL e incluir todos los nodos. Se debe obtener el dato de la mayor cédula y de la menor cédula, e imprimirlos en un archivo (/misc/output/max_and_min_id.txt) .Luego se debe eliminar todos los nodos.

Esto se debe repetir para 5 archivos (10000 líneas, 5000, 1000, 100, y 10), y se debe obtener el tiempo de ejecución del programa para cada uno (escŕibalos en un archivo llamado /misc/data/running_times.txt). Cree una gráfica con estos datos y obtenga una expresión para el tiempo de ejecución total del programa. La gráfica puede construirla de la manera que usted considere óptima. Los datos de tiempo obtenidos deben ser demostrados en clase.

Desglose de puntos:

Implementación de la estructura de datos: 50%

Implementación de las pruebas unitarias: 20%

Implementación del programa principal: 15%

Documentación, formato de entrega, y legibilidad: 15%