Universidad Nacional Autónoma de México

Facultad de Ingeniería
División de Ciencias básicas
Estructura de datos y algoritmos II

Proyecto final:
Binary Search Tree

Integrantes:
Gaytán Arteaga Julio
López Santibáñez Jiménez Luis Gerardo
Arrieta Ocampo Braulio
Varela Garcia Alan Paul

Profesor: Catana Salazar Juan Carlos

Grupo: 08
Semestre: 2017-1
Fecha de entrega: 20 - noviembre - 2016

SECCIÓN 1
Introducción y presentación del problema

Para organizar datos de una forma particular, las estructuras de datos son la opción para ello. A través de sus diferentes tipos permiten hacer mencionada tarea de administración haciendo uso de diferentes algoritmos y técnicas de programación.

Una de las estructuras de datos más importantes son los árboles, que simulan una estructura arbórea jerárquica, con un valor raíz y subárboles (izquierdo y derecho) con un padre, conteniendo un conjunto de nodos ligados.

Como el resto de las estructuras de datos, los árboles engloban múltiples conceptos de construcción y aplicación, así como diversos problemas con respecto al tratamiento de datos, pudiendo involucrar la cantidad de memoria en un ordenador a usar o el tiempo de ejecución del programa que implementa a la estructura.

En esta ocasión el problema que se nos presenta, y el cual tenemos que resolver, es balancear un árbol de forma automática, (auto balanceo), como medio para emplear la propiedad de búsqueda de los BST en el tiempo más óptimo; para ello, utilizaremos Python para el código base, ya que este proyecto tiene la finalidad de hacer que el usuario pueda ver lo que está pasando en forma gráfica o visual. Nos referimos a forma gráfica o visual cuando el usuario es capaz de ver cómo se está formando el árbol, con todos los elementos que lo componen como nodos, aristas, padres e hijos; en este caso también se puede interactuar con él, ya que se tiene un menú el cual realiza ciertas funciones como insertar un nodo, eliminar un nodo, obtener el máximo y mínimo, cargar de archivo, etc. La parte gráfica fue realizada con ayuda de frameworks como Django de Python y jQuery de Javascript.

Conceptos básicos

Árbol binario (BST)

Un árbol de búsqueda binaria, o BST por sus siglas en inglés (Binary Search Tree), es una estructura de datos que soporta varias operaciones dinámicas tales como la búsqueda de un elemento significativamente más rápida que en el caso de una lista ligada (sea máximo, mínimo u otros como el predecesor o sucesor de un nodo), así como inserción y eliminación.
Características

- No debe tener valores duplicados en los nodos y además, tiene la característica de que:

- Los valores en cualquier subárbol izquierdo son menores que el valor en su nodo padre.

- Los valores en cualquier subárbol derecho son mayores que el valor en su nodo padre

AVL Tree
Un árbol AVL es un tipo de BST. Su nombre proviene de sus inventores: Adelson-Velskii y Landis. La característica principal de los árboles AVL es su autobalanceo, por lo que sus propiedades se enlistan de la siguiente manera:

- La altura de todos los subárboles de cada nodo difiere, a lo más, de 1.

- Cada subárbol es del tipo AVL.

Rotación
La rotación es una operación en un árbol binario que cambia la estructura sin interferir con el orden de los elementos.

- Altura o profundidad

La profundidad o altura de un árbol binario es el máximo nivel de cualquier hoja en el árbol. El nivel de cualquier otro nodo en el árbol es uno más que el nivel de su padre.

- Árbol balanceado

El nivel de cualquier nodo no difiere por una diferencia mayor a dos con la del nodo con el que tiene en común al nodo padre.

Problema
Balancear de forma automática un árbol de búsqueda binaria dado.
El problema consiste en que se tiene un árbol binario con altura o profundidad, el cual puede esta balanceado o no; cuando nos referimos a desbalanceado es porque el nodo padre, que tiene uno o dos nodos sucesores (hijos), este nodo padre en sus hijos respectivos tienen una diferencia de altura o profundidad mayor o igual a dos.

El programa que implementamos se encarga de verificar si dicho árbol está desbalanceado o no, en cada caso se presentan situaciones diferentes:

- Caso I – El árbol está desbalanceado.

Este puede tener más de un nodo desbalanceado, es decir que si un hijo está desbalanceado, y tiene ancestros, estos tienen una probabilidad muy alta de estar desbalanceados, en este caso se aplica la rotación a la izquierda o derecha, dependiendo de donde se encuentre este nodo, n veces hasta que se logre el balanceo, estos balanceos n veces los podemos ver como doble rotación.

Este puede tener más de un nodo desbalanceado, es decir si más de un nodo está desbalanceado consecutivamente: en este caso se aplica la rotación a la izquierda o derecha, dependiendo de donde se encuentre este nodo, n veces hasta que se logre el balanceo, estos n balanceos los podemos ver como doble rotación.

- Caso II – El árbol está balanceado

Cuando esto sucede quiere decir que nuestro árbol no tiene ningún nodo desbalanceado, en este caso no le aplicamos ninguna rotación; este es el árbol que se prende tener después de que se ejecute este programa.

Algunas acciones que realiza el programa

Cabe mencionar que cada que se realiza un cambio como: agregar un nodo, eliminar un nodo, se verifica si el árbol está o no desbalanceado.

Cuando leemos de archivos lo único que hacemos es que se genera un árbol con datos ya establecidos, y cuando guardamos se crea un archivo con los datos que tenemos, en este caso es el árbol en pantalla.

Cuando obtenemos el máximo y el mínimo, nos fijamos en los extremos del árbol, porque, como ya se mencionó esa es la estructura del árbol.

Sección 2

Definición intuitiva (ejemplo didáctico) y definición formal.

- **Definición intuitiva (ejemplo didáctico)**

Supongamos que tenemos un árbol binario balanceado, y usamos la función para insertar un nodo, al hacer esto puede que el árbol resultante este o no balanceado. Podemos comprobar si el árbol esta balanceado si y solo si el nodo recién insertado es un descendiente inmediato de un nodo con altura mínima (hoja) en el balance previo a la inserción. Si esto no sucede tenemos que hacer uso de nuestro código para que este árbol se mantenga balanceado.

Para que el árbol se mantenga balanceado es necesario hacer una rotación:

- El recorrido en orden del árbol rotado tiene que ser el mismo que para el árbol original, es decir, tiene que seguir cumpliendo las características de un árbol binario.

- El árbol rotado debe de estar balanceado

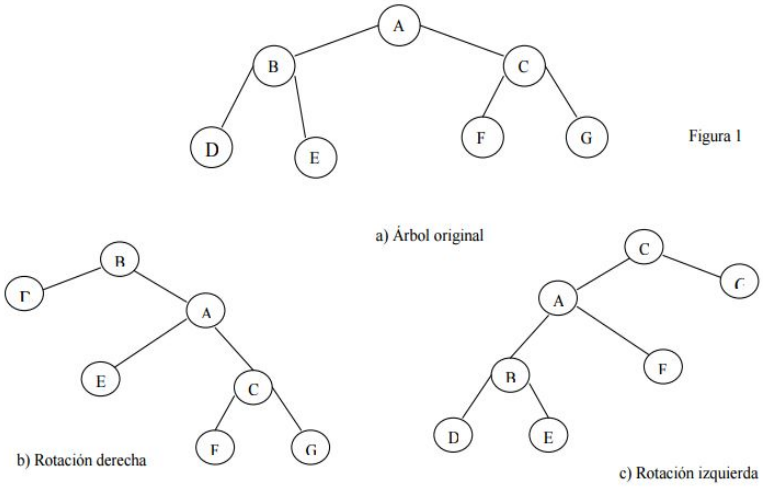

Figura 1

En la figura 1 se muestra un árbol alanceado, en la figura 1.b se muestra la rotación a la derecha y el la figura 1.c se muestra la rotación a la izquierda.

Ejemplo didáctico

Tenemos un árbol, claramente desbalanceado.

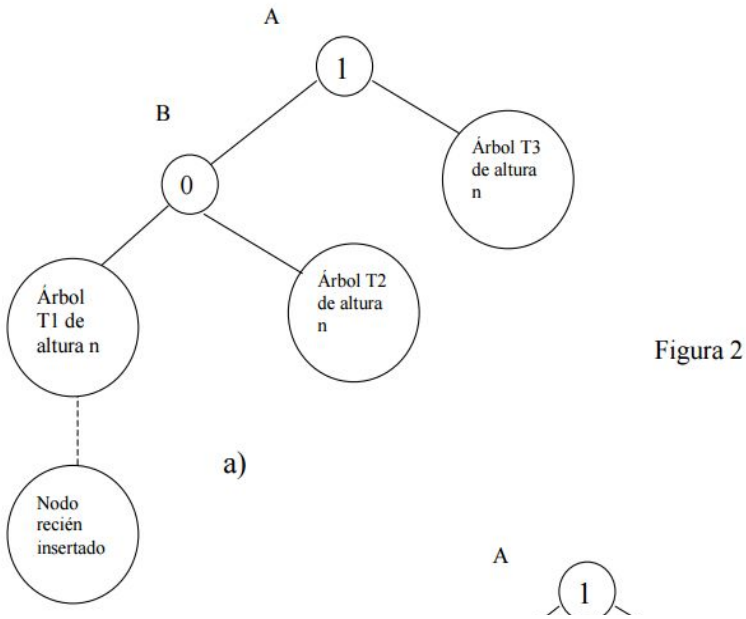

Figura 2

Le aplicamos una rotación, esperemos que el lector pueda identificar de qué tipo de rotación se trata.

(*Rotación a la derecha*)

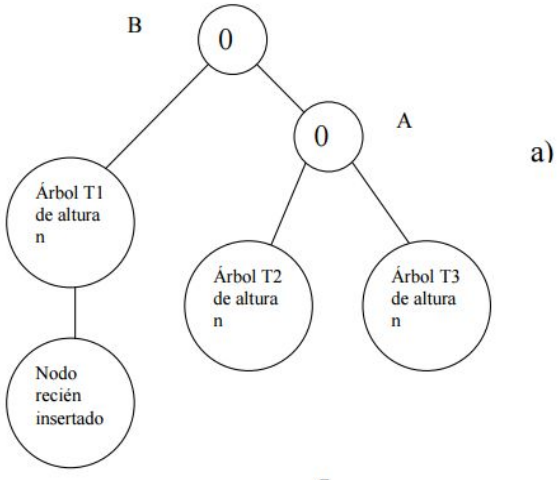

Figura 3

Tenemos el árbol balanceado

Formalmente, un BST se puede definir como:

"*(…)un árbol binario en el cual cada nodo tiene una llave comparable (y un valor asociado) y satisface la restricción en la cual la llave en cualquier nodo es más grande que las llaves en todos los nodos del subárbol izquierdo a este y más pequeña que las llaves en todos los nodos de su subárbol derecho."(Sedgewick,1983).*

Sección 3*.*

Descripción de él o los algoritmos implementados en el proyecto. Cabe mencionar que toda esta sección está basado en python

Class vertex

Creamos una clase en Python la cual recibe un entero el cual lo inicializamos como nodo, con algunos atributos como: identificador, padre, hijo izquierdo, hijo derecho, altura y FE, los valores de los atributos se inicializan en None ó 0 respectivamente, lo único que tenemos que destacar es que el identificador del nodo será el valor del entero que se recibe.

Class árbol

Inicializamos una raíz, la parte más importante para nuestro árbol, la cual será asignada al primer nodo que entre en el árbol.

- Función*:* **crearArbol**

- Función: **insertarVértice**

- Función: **agregar**

Estas tres funciones se ejecutan en forma secuencial ya que se ejecutan en un cierto orden. La función **crearArbol** lee los datos de entrada en este caso un conjunto de datos en forma de lista, esta función toma cada uno de los elementos de la lista y hace uso de la función **insertarVértice** *en la cual se inicializan los*

nodos, como ya se menciona anteriormente, y se asigna la raíz al primer nodo que se pasa por esa función, esta función hace uso de **agregar** *en la cual se van insertando los nodos de tal forma que se crea la estructura* **ARBOL BINARIO***. Se termina la ejecución de estas funciones cuando la función* **crearArbol** termina, es decir cuando todos los datos de entrada han sido utilizados.

- Función: **imprimir**

Lo único que hacemos en esta función es imprimir los datos de entrada, que en este caso son los nodos del árbol los imprimimos en orden INFIX, claro, aplicando las reglas de este tipo de impresión de un árbol.

- Función*:* **altura**

En esta función, que es una de las más importantes de nuestro programa, calcula la altura o nivel de nuestro árbol, esto lo hacemos, recorriendo desde los nodos extremos, hojas, hasta la raíz. Esta función es recursiva, la cual recorre todo el árbol hasta llegar a los nodos extremos y desde ahí comienza a hacer el cálculo de la altura, el nivel inicial para los nodos extremos es 0, y conforme se avanza hacia la raíz se le aumenta uno, cabe mencionar que el nodo más próximo a la raíz toma como altura el nivel más alto de sus hijos y le aumenta uno. Al final se tiene el árbol con una profundidad establecida.

- Función:**Función *calcFE*** *(Factor de Equilibrio)*

Por este momento pensemos que nuestros nodos se están insertando en un plano XY, en el cual nuestro árbol se graficara, representando a cada nodo como un punto sobre este plano. La coordenada “y” corresponde a la altura de cada nodo, ”hasta este momento los nodos se graficarían de tal forma que se podrían observar como en forma de línea recta paralela al eje de las ordenadas, o eje de las y’s, ya que solo tienen una altura, lo cual causa que solo se desplacen sobre este eje. ”. Pero con la esta función de **calcFE**, haremos que también se desplacen sobre el eje “x”, para que de esta forma se pueda ver, ahora sí, como un árbol.
Factor de equilibrio. Esta función calcula la distancia que tiene que estar alejado un nodo hijo del padre, esta distancia está medida hacia los extremos y se puede ver como un caso similar a la profundidad, pero ahora sobre el eje de las x’s.

- Función: **obtenerOBJDesv**

La función **obtenerOBJDesv** *se encargara de pasar por todos los nodos de nuestra gráfica, verificando que este no se encuentre desnivelado, en cada iteración por un nodo se hace llamada a las funciones* **altura** *y* **calcFE** *en esta función solo se irán guardando los [nodos, posición] en forma de arreglo, que a la vez están contenidos en un arreglo. Al final de la recursividad se tendrá un arreglo con los [nodos, lado desnivelado] desnivelados.*

- Función: **RR** (rotación a la derecha)

- Función: **LR** (rotación a la izquierda)

En estas dos funciones no hay mucho que decir pues solo hacemos la rotación a la izquierda o la derecha, dependiendo de qué es lo deseamos realizar. Tomamos al nodo desbalanceado, sacado del arreglo que regresa la función **obtenerOBJDesv** y le aplicamos la rotación, tomando en cuenta el nodo y el lado desnivelado. Nos apoyándonos en un nodo auxiliar, pues cuando hacemos la rotación tenemos que hacer que uno de los hijos del nodo desbalanceado sea el padre, quitamos los enlaces que existen entre los nodos y le asignamos las nuevas, de tal forma que se pueda hacer la rotación sin perder las propiedades del árbol binario y de que se siga teniendo la estructura de árbol, se hacen un par de verificaciones para no caer en errores, como que el padre no existe y cosas del estilo.

- Función: **RDR** (doble rotación a la derecha)

- Función: **LDR** (doble rotación a la izquierda)

Estas dos funciones lo único que hacen es aplicar las dobles rotaciones, como el nombre lo indica, pues se tiene que hacer una doble rotación cuando el nodo desbalanceado, después de aplicar una rotación queda desnivelado. En el código solo se hace llamada a las funciones una vez intercalada, como ejemplo: “si se tiene que hacer una doble rotación a la derecha, lo primero que hacemos es una rotación a la izquierda en torno al nodo desnivelado y luego una la derecha en torno al nodo desnivelado”. En estas funciones lo único que se hace es realizar estas dos rotaciones en orden, ósea, se hacen llamadas a las funciones **LR, RR.**

- Función: **buscarVertice**

En esta función sólo se recorre el árbol de forma inteligente, búsqueda binaria, y se imprime “Identificador del nodo y su profundidad correspondiente”, cuando se encuentra, en caso contrario se imprime un mensaje de “Nodo no encontrado”.

- Función: **obtenerMin**

- Función: **obtenerMax**

Recorre el árbol, de una forma parecida a la búsqueda binaria, y cuando se llega al extremo, hoja, se imprime, pues en los extremos, ya sea derecho o izquierdo es en donde se encuentran los valores máximos y mínimos, esto por que cumple con las propiedades de un árbol binario.

- Función: **autobalanceo**

La función más importante del código, se dejó hasta el final para que en esta explicación se tenga una idea de que es lo que realizan las funciones mencionadas anteriormente, pues esta función **autobalanceo** implementara la mayoría de ellas,

ya que hasta el momento las funciones que no sean para crear el árbol no habían sido implementadas, esta función **autobalanceo** es la que hace uso de ellas.
Se comienza haciendo uso del arreglo obtenido de la función **obtenerOBJDesv** y por cada elemento de este arreglo se hacen unas comparaciones para poder decidir qué tipo de rotación se realizarán, las comparaciones son, solo por mencionar algunas, "¿el nodo desnivelado tiene hijo derecho?, ¿el nodo desnivelado tiene hijo izquierdo?, ¿el nodo desnivelado tiene una altura mayor al de las hojas? Usamos la función **altura**", la comparación que decide qué tipo de **rotación (RR, LR)** usar es la de **FE (factor de equilibrio),** pues esta nos dice hacia qué lado, está desplazado y por consecuente que tipo de rotación usar. La comparación que nos permite decidir si se aplica una **doble rotación (RDR, LDR)** es, "si las alturas entre nodos hijos del desbalanceado es mayor a dos", ósea utilizamos la función **altura**.

Interfaz gráfica utilizada
Lo usado para fungir como interfaz gráfica fue **Django**, un framework de desarrollo web escrito en Python.
La implementación realizada en Django del código para el autobalanceo de un BTS dado involucró a los siguientes conceptos:

- **API REST -** Es una librería de funciones (API) a la cual se accede por el protocolo HTTP. Una API REST define un conjunto de funciones con las cuales los desarrolladores pueden ejecutar solicitudes y recibir respuestas a través del protocolo HTTP, tales como GET y POST.

- **Métodos GET y POST –** Métodos de envío de datos a una página web. De manera general, el método GET envía una solicitud al servidor y, para este caso, recibe de él como respuesta la aplicación web. Por otro lado, POST envía información tal como los nodos que conformarán el árbol BST desde el cliente hacía el servidor y éste último responde con el BST esperado.

- **JSON -** (JavaScript Object Notation) es un formato ligero para el intercambio de datos y representa una alternativa a XML. Puede ser leído por cualquier lenguaje de programación y, por lo tanto, puede ser usado para el intercambio de información entre distintas tecnologías.

- **AJAX –** Acrónimo de Asynchronous JavaScript And XML; es una técnica que permite, mediante programas escritos en JavaScript, que un servidor y un navegador intercambien información, posiblemente en XML u otro tipo de datos como JSON, de forma asíncrona. La ejecución de un programa con AJAX puede realizarse sin necesidad de recibir respuesta de operaciones hechas con anterioridad.

Un ejemplo de su funcionamiento es un reloj puesto en una página web; sin AJAX para saber la hora es necesario refrescar la página, mientras que con AJAX el reloj va actualizando la hora sin necesidad de recargar la página web, manteniendo estático el contenido que no se encuentra involucrado con dicho reloj.

Para el BST, AJAX es el que se encarga de construir y modificar el árbol sin necesidad de tener que solicitar al servidor la aplicación web otra vez, actualizando su estructura de forma asíncrona al resto de la página.

- **JQuery –** Biblioteca de JavaScript que simplifica la programación de páginas web usando dicho lenguaje de programación, reduciendo al mínimo el código a utilizar.

- **DOM -** (Document Object Model**),** es una interfaz de programación de aplicaciones para documentos HTML y XML. El Modelo de Objetos del Documento establece cómo se representan los documentos HTML y XML como objetos, para poder ser utilizados por programas orientados a objetos.

- **D3.js –** Librería de JavaScript para la representación gráfica de información, utilizando HTML, SVG y CSS. Además, permite ligar datos a un DOM, y luego transformarlos a un documento.

Para el programa del autobalanceo, D3.js es la herramienta que permite visualizar el BST como el conjunto de nodos establecidos visualmente como círculos unidos entre sí mediante rectas, que son sus aristas.

Conclusiones

Un BST que se balancea por si mismo sin importar los nodos que lo fuesen a conformar antes de su construcción formal resulta en un AVL Tree, un tipo de BST cuya función principal es precisamente la de autobalancearse.

Si los árboles de búsqueda binaria se encuentran desbalanceados provocan que el tiempo que le tome al programa que aplica el algoritmo buscar un elemento dentro del árbol no sea el más óptimo, por lo que el balanceo es importante para refinar la ejecución del programa.

Por otro lado, la implementación de la estructura de datos que son los BST contempla distintos conceptos de programación orientada a objetos tales como el construir al árbol manejando a cada uno de sus nodos como un objeto con atributos tales como el poseer un valor, un padre, un hijo izquierdo, un hijo derecho y altura, facilitando tanto la creación de la estructura como el manejo de sus datos,

empleando otras funciones que se encargan de aplicar las propiedades más importantes de este tipo de estructura como la principal que es la búsqueda de elementos.

Referencias bibliográficas

- CORMEN, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2009), *Introduction to Algorithms* (Third Edition)*,* EE.UU. MIT Press and McGraw -Hill,v págs(286-309).

- DEVADAS Srini
  "*Lecture 5: Binary Search Trees, BST Sort*", MIT OpenCourseWare, <https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-5-binary-search-trees-bst-sort/>, (recuperado el 15 de noviembre del 2016.)

- SEDGEWICK, Robert
  "3.2 BINARY SEARCH TREES", ALGORITHMS, 4H EDITION. Princeton University, <http://algs4.cs.princeton.edu/32bst/>, (recuperado el 15 de noviembre de 2016).

- Django
  "*Django documentation*", django,<https://docs.djangoproject.com/en/1.10/>, (recuperado el 15 de noviembre de 2016)