Universidad San Carlos de Guatemala
Estructura de datos
Ing. Oliver Sirra
Axiliar Odri
David Enrique Lux Barrera 201931344

Manual de Tecnico:

Introducción:

El siguiente programa demuestra la funcionalidad de los conocimientos aplicados acerca de los temas vistos en clase, sobre un programa de carrera de caballos en donde el usuario puede ingresar las apuestas de las personas y muestra como estas son procesadas por varios metodos y generar los resultados por medio de las tablas correspondientes donde indica un juego de un puzzle 15

Requisito del sistema:

1. Windows 7 o superior
2. Ubunto u otras distribuciones superior a la 16,5 LTS
3. Porcesador Pentium o más
4. Java JDK
5. Visual Studio Code

Arboles AVL:

Como se mostró en este objeto de aprendizaje, los árboles binarios de búsqueda (ABB) son una estructura de datos que intenta conseguir mejor tiempo de acceso a los datos en las operaciones de búsqueda/recuperación, inserción o eliminación comparado con los tiempos en estructuras lineales como arreglos y listas. El acceso a un dato es proporcional a la altura del árbol, ya que su ubicación podría ser, en el peor de los casos, en una hoja. La forma del ABB dependerá de la secuencia en que los datos se fueron insertando en el momento de la creación. De esta forma, en el peor de los casos, la búsqueda puede llegar a tener orden de complejidad O(n).

Por otro lado, los árboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio (o balanceo), la búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log2 n).

Cuando se tiene un problema donde el tiempo de búsqueda es un requisito importante para la solución se debe optar por los árboles AVL ya que su forma no depende del orden en que los datos son insertados y siempre se mantiene la complejidad en O(log2 n).

Si el tiempo de búsqueda no es un requisito importante, la simplicidad de los algoritmos de Árboles Binarios de Búsqueda puede ser suficiente para la resolución del problema.

Árbol binario de búsqueda que cumple con la condición de que la diferencia entre las alturas de los subárboles de cada uno de sus nodos es, como mucho 1. La denominación de árbol AVL viene dada por los creadores de tal estructura (Adelson-Velskii y Landis)

Codigo

Recorrido Post, pre e in Orden:

```
private void inOrden(Nodo nodo, ArrayList<String> cartas) {
    if (nodo != null) {
        inOrden(nodo.getHijoIzquierdo(), cartas);
        cartas.add(nodo.getCarta().toString());
        inOrden(nodo.getHijoDerecho(), cartas);
    }
}

private void preOrden(Nodo nodo, ArrayList<String> cartas) {
    if (nodo != null) {
        cartas.add(nodo.getCarta().toString());
        preOrden(nodo.getHijoIzquierdo(), cartas);
        preOrden(nodo.getHijoDerecho(), cartas);
    }
}

private void postOrden(Nodo nodo, ArrayList<String> cartas) {
    if (nodo != null) {
        postOrden(nodo.getHijoIzquierdo(), cartas);
        postOrden(nodo.getHijoDerecho(), cartas);
        cartas.add(nodo.getCarta().toString());
    }
}
```

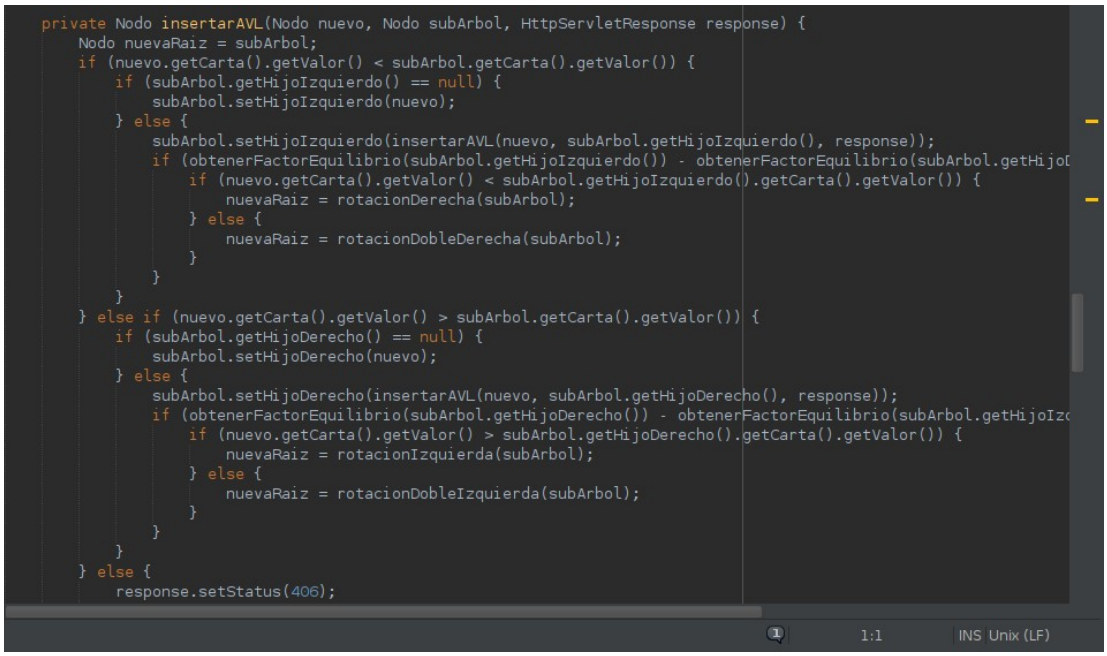
Método para devolver el recorrido:

```
//Metodos para devolver el orden
public ArrayList<String> inOrden() {
    ArrayList<String> cartas = new ArrayList<>();
    this.inOrden(this.raiz, cartas);
    return cartas;
}

public ArrayList<String> preOrden() {
    ArrayList<String> cartas = new ArrayList<>();
    this.preOrden(this.raiz, cartas);
    return cartas;
}

public ArrayList<String> postOrden() {
    ArrayList<String> cartas = new ArrayList<>();
    this.postOrden(this.raiz, cartas);
    return cartas;
}
```

Método para obtener y retornar un Nivel

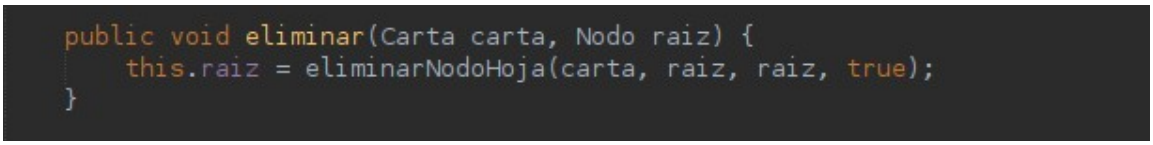
```
public ArrayList<String> getNivel(int nivel) {
    ArrayList<String> cartas = new ArrayList<>();
    this.getNivel(this.raiz, cartas, nivel);
    return cartas;
}

private void getNivel(Nodo nodo, ArrayList<String> cartas, int nivel) {
    if (nodo != null) {
        if (nivel == 0) {
            cartas.add(nodo.getCarta().toString());
        }
        getNivel(nodo.getHijoIzquierdo(), cartas, nivel-1);
        getNivel(nodo.getHijoDerecho(), cartas, nivel-1);
    }
}
```

Método para Insertar un nodo



```
private Nodo insertarAVL(Nodo nuevo, Nodo subArbol, HttpServletResponse response) {
    Nodo nuevaRaiz = subArbol;
    if (nuevo.getCarta().getValor() < subArbol.getCarta().getValor()) {
        if (subArbol.getHijoIzquierdo() == null) {
            subArbol.setHijoIzquierdo(nuevo);
        } else {
            subArbol.setHijoIzquierdo(insertarAVL(nuevo, subArbol.getHijoIzquierdo(), response));
            if (obtenerFactorEquilibrio(subArbol.getHijoIzquierdo()) - obtenerFactorEquilibrio(subArbol.getHijoD
                if (nuevo.getCarta().getValor() < subArbol.getHijoIzquierdo().getCarta().getValor()) {
                    nuevaRaiz = rotacionDerecha(subArbol);
                } else {
                    nuevaRaiz = rotacionDobleDerecha(subArbol);
                }
            }
        }
    } else if (nuevo.getCarta().getValor() > subArbol.getCarta().getValor()) {
        if (subArbol.getHijoDerecho() == null) {
            subArbol.setHijoDerecho(nuevo);
        } else {
            subArbol.setHijoDerecho(insertarAVL(nuevo, subArbol.getHijoDerecho(), response));
            if (obtenerFactorEquilibrio(subArbol.getHijoDerecho()) - obtenerFactorEquilibrio(subArbol.getHijoIzq
                if (nuevo.getCarta().getValor() > subArbol.getHijoDerecho().getCarta().getValor()) {
                    nuevaRaiz = rotacionIzquierda(subArbol);
                } else {
                    nuevaRaiz = rotacionDobleIzquierda(subArbol);
                }
            }
        }
    } else {
        response.setStatus(406);
```

Método para eliminar un nodo:



```
public void eliminar(Carta carta, Nodo raiz) {
    this.raiz = eliminarNodoHoja(carta, raiz, raiz, true);
}
```

Método para Verificar una eliminación de un nodo de una hoja

```
private Nodo eliminarNodoHoja(Carta carta, Nodo raiz, Nodo padre, boolean derecha) {
    Nodo nuevaRaiz = raiz;
    if (raiz == null) {
        //Arbol vacio
    } else if (raiz.getCarta().getValor() == carta.getValor()) {
        if (padre == null) {
            this.raiz = null;
        } else {
            if (derecha) {
                padre.setHijoDerecho(null);
                nuevaRaiz = null;
            } else {
                padre.setHijoIzquierdo(null);
                nuevaRaiz = null;
            }
        }
    } else if (raiz.getCarta().getValor() < carta.getValor()) { //Derecha
        raiz.setHijoDerecho(eliminarNodoHoja(carta, raiz.getHijoDerecho(), raiz, true));
        actualizarFe(raiz);
    } else {
        raiz.setHijoIzquierdo(eliminarNodoHoja(carta, raiz.getHijoIzquierdo(), raiz, false)); //Izquierda
        actualizarFe(raiz);
    }

    if (obtenerFactorEquilibrio(raiz.getHijoDerecho()) - obtenerFactorEquilibrio(raiz.getHijoIzquierdo()) == 2)
        Nodo derecho = raiz.getHijoDerecho();
        if (derecho.getHijoIzquierdo() != null && derecho.getHijoDerecho() == null) {
            nuevaRaiz = rotacionDobleIzquierda(raiz);
        } else {
            nuevaRaiz = rotacionIzquierda(raiz);
        }
    } else if (obtenerFactorEquilibrio(raiz.getHijoIzquierdo()) - obtenerFactorEquilibrio(raiz.getHijoDerecho()
        Nodo izquierdo = raiz.getHijoIzquierdo();
```

Método para Buscar algún nodo:

```java
//Buscar
public Nodo buscar(Carta carta, Nodo raiz) {
    if (raiz == null) {
        return null;
    } else if (raiz.getCarta().getValor() == carta.getValor()) {
        return raiz;
    } else if (raiz.getCarta().getValor() < carta.getValor()) {
        return buscar(carta, raiz.getHijoDerecho());
    } else {
        return buscar(carta, raiz.getHijoIzquierdo());
    }
}
```

Método para matarme del frio:

```java
//Elimina las cartas en el orden que les corresponde
public void eliminarCartas() {
    Arbol arbol = LlamadasArbol.getArbol();
    try {
        String jsonDelete = Simbolos.remplazarSimbolos(Simbolos.obtenerContenido(request.getReader()));
        StringReader sr = new StringReader(jsonDelete);
        lexerJsonDelete lexer = new lexerJsonDelete(sr);
        parser par = new parser(lexer);

        try {
            par.parse();
            if (!par.isErrores()) {
                ArrayList<Carta> cartas = par.getCartas();

                eliminarCartas(cartas, arbol);

                ArrayList<String> cartasS = arbol.preOrden();
                for (String c : cartasS) {
                    System.out.println(c);
                }

            } else {
                response.setStatus(400);
            }
        } catch (Exception ex) {
            ex.printStackTrace(System.out);
            response.setStatus(400);
        }

    } catch (IOException ex) {
        response.setStatus(400);
    }
}
```

34:57 INS Unix (LF)

Para la visualización de los Servetls a todos:

```java
 */
package Servelts;

import Clases.Agregar;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author david
 */

//Servet que agrega la insercion de los nodos
@WebServlet(name = "add", urlPatterns = {"/Game/add"})
public class AgregarServelt extends HttpServlet {

    @Override
    public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOExcep
        System.out.println("Peticion post /Game/add");
        new Agregar(request, response).insertarCarta();
    }
```

```java
import Clases.Eliminar;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *
 * @author david
 */

//Servet que elimina la insercion de los nodos
@WebServlet(name = "delete", urlPatterns = {"/Game/delete"})
public class EliminarServelt extends HttpServlet {

    @Override
    protected void doDelete(HttpServletRequest request, HttpServletResponse response) throws ServletException, 
        new Eliminar(request, response).eliminarCartas();
    }

}
```

```java
package Servelts;

import Clases.Niveles;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


/**
 *
 * @author david
 */

//Servet que Obtiene el nivel de los nodos
@WebServlet(name = "get-level", urlPatterns = {"/Game/get-level"})
public class NivelesServelt extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException 
        new Niveles(request, response).getNivel();
    }

}
```