UNIVERSIDAD SAN CARLOS DE GUATEMALA

CENTRO UNIVERSITARIO DE OCCIDENTE

DIVISIÓN CIENCIAS DE LA INGENIERÍA

ESTRUCTURA DE DATOS

ING. OLIVER ERNESTO SIERRA PAC

PRIMER SEMESTRE DE 2022

"Proyecto 2 - Pyramid"

Por

Luis Alejandro Méndez Rivera, 202030627

Lunes 09 de Mayo del 2,022

Documentación Técnica

El presente informe tiene como finalidad explicar los aspectos importantes sobre el desarrollo de la API Rest “Pyramid”, la cual fue elaborada como proyecto del curso Estructura de Datos del primer semestre del año 2022 con la finalidad de profundizar y cumplir con el desarrollo de un servicio en el que se emplee el uso de árboles AVL y desarrollo web; simulando así un juego de cartas (naipes) el cual consiste en armar una baraja en forma de pirámide e ir eliminando cartas cuyos valores sumados sean 13. Cumpliendo además con la funcionalidad de insertar cartas, pedir estado del árbol, pedir nodos de cierto nivel del árbol, pedir nodos ordenados en determinado ordenamiento AVL, etc.

Árboles AVL

Este árbol siempre está siempre equilibrado, de manera que todos los nodos, la diferencia de altura del lado izquierdo con la del lado derecho siempre estará entre -1 y 1. Es un árbol de búsqueda binaria que está completamente balanceado, fue creado por Adelson Velskii y Landis, de ahí el nombre de AVL.

Movimientos o rotaciones

A la hora de insertar un nodo a nuestro árbol, siempre es necesario verificar si esta inserción causará algún desbalance, de manera que si la causa se debe de realizar ciertas rotaciones para mantener el equilibrio del árbol AVL.

Las rotaciones son:

- ### Rotación simple hacia la izquierda

De un árbol de raíz (r) y de hijos izquierdo (i) y derecho (d), consiste en formar un nuevo árbol cuya raíz sea la raíz del hijo derecho, como hijo derecho colocamos el hijo derecho de d (nuestro d’) y como hijo izquierdo construimos un nuevo árbol que tendrá como raíz la raíz del árbol (r), el hijo izquierdo de d será el hijo derecho (i’) de r y el hijo izquierdo será el hijo izquierdo del árbol (i).

Requisito: Tiene que tener hijo derecho no vacío.

- ### Rotación simple hacia la derecha

De un árbol de raíz (r) y de hijos izquierdo (i) y derecho (d), lo que haremos será formar un nuevo árbol cuya raíz sea la raíz del hijo izquierdo, como hijo izquierdo colocamos el hijo izquierdo de i (nuestro i’) y como hijo derecho construimos un nuevo árbol que tendrá como raíz, la raíz del árbol (r), el hijo derecho de i (d’) será el hijo izquierdo y el hijo derecho será el hijo derecho del árbol (d).

- Rotación doble a la izquierda:

La Rotación doble a la Izquierda son dos rotaciones simples, primero rotación simple derecha y luego rotación simple izquierda en sencilla explicación.

- Rotación doble a la derecha:

La Rotación doble a la Derecha son dos rotaciones simples, primero rotación simple izquierda y luego rotación simple derecha. En sencilla explicación

API REST y su funcionamiento general

Es un servicio programado (backend) cuya función en es ser el intermediario entre un cliente y un servidor para cumplir con ciertos mandatos, manejarlos y devolver objetos, ya que esta, es la encargada de enviar al servidor datos que reciba del cliente y responderle al cliente cosas que el servidor le envía.

Esta herramienta no contiene más que funciones asignadas a un directorio/rutas en especial y mediante peticiones HTTP.

Las requisiciones HTTP que más se usan en una API son: GET, POST, DELETE, PUT. Sin ellas no se podría comunicar el cliente con el servidor

- POST: crea datos en el servidor;
- GET: lectura de datos en el host;
- DELETE: borra la información;
- PUT: registro de actualizaciones.

Estructura de Pyramid

La Api fue desarrollada con el FrameWork para desarrollo de API Rest, “Spring boot”, con el objetivo de contener una estructura clara, organizada y de facil manejo en cuanto a peticiones HTTP se refiere.

Se utilizo el lenguaje de programación Java en su versión OpenJDK11 (default de Spring boot).

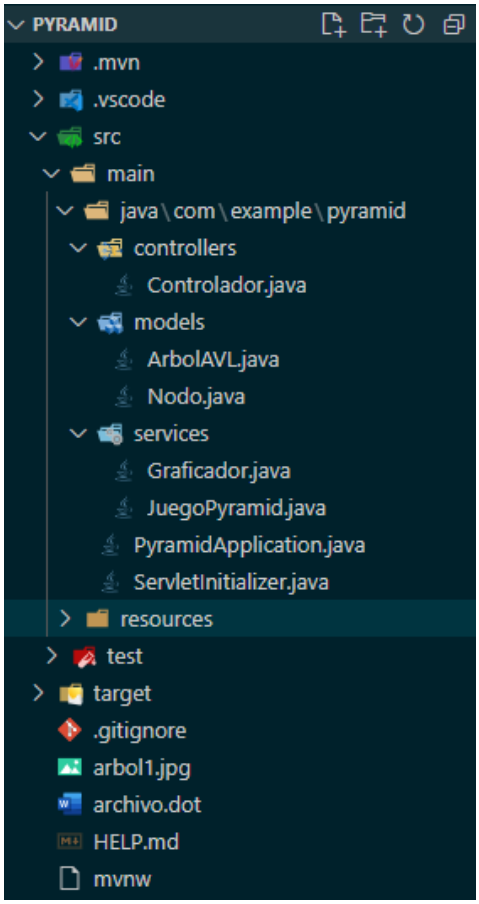
Se utilizó la herramienta de diagramas programables Graphiz (Diagraph) para la implementación de reporte de estados de Arboles AVL graficados.

Requisitos para Pyramid:

- Instalación del OpenJDK 11 en el computador.
- Instalación de la herramienta Graphiz en su versión 3.0.0.
- Instalación de la herramienta de peticiones web Postman.
- Conexión a internet estable.

Organización de directorios

La API se conforma por un directorio raíz llamado main, el cual contiene 3 secciones principales para su correcto funcionamiento:



- Controllers:
  Este directorio contiene el archivo principal, ya que contiene las rutas HTTP e invocación de métodos (del directorio Models) para el funcionamiento de Pyramid.

  Peticiones “GET” de clase Controlador.java:

```java
@RestController
public class Controlador {

    JuegoPyramid juego = new JuegoPyramid();

    /*
     * Peticiones: GET
     */
    @GetMapping("/")
    public String bienvenida() {

        return " Hola, Bienvenido a Pyramid (API)!";
    }

    @GetMapping("Game/get-level") // ("/Game/get-Level?")
    public String obtenerCartasDeNivel(@RequestParam(value = "level", defaultValue = "0") int nivel) {
        int nivelConsultado = nivel;
        juego.buscarCartaPorNivel(nivelConsultado); //Metodo que accede al nivel del arbol pedido
        String response = juego.cartasObtenidasPorNivel(); //Devolver nodos izq-der del nivel pedido en formato JSON

        return response; //System.out.println(" Imprimir cartas del nivel pedido: " + nivel);
    }

    @GetMapping("Game/status-avltree")
    public String obtenerEstadoArbol(@RequestParam(value = "status", defaultValue = "") String orden) {
        String ruta = "{\n" + "\"path\":\"" + juego.generarGrafica() + "\"\n}";

        return ruta; //System.out.println(" Devolver path");
    }
```

```java
    @GetMapping("Game/avltree") // ("Game/avltree?")
    public String obtenerOrdenArbol(@RequestParam(value = "transversal", defaultValue = "inOrder") String orden) {
        String response = "";

        if (orden.equals(anObject: "inOrder")) {
            response = juego.enOrden(); //Devolver nodos de orden pedido en formato JSON
            //System.out.println(" Imprimir cartas del arbol InOrden");
        }
        if (orden.equals(anObject: "preOrder")) {
            response = juego.preOrden(); //Devolver nodos de orden pedido en formato JSON
            //System.out.println(" Imprimir cartas del arbol preOrden");
        }
        if (orden.equals(anObject: "postOrder")) {
            response = juego.postOrden(); //Devolver nodos de orden pedido en formato JSON
            //System.out.println(" Imprimir cartas del arbol PostOrden");
        }

        return response; //System.out.println(" Imprimir cartas  en el orden pedido: " + orden);
    }
```

Peticiones “POST” de clase Controlador.java:

```java
@PostMapping("Game/start")
public ResponseEntity<String> iniciarJuego(@RequestBody String json) {
    int estado = juego.iniciarPyramid(json); //Inserta cartas (nodos) 1 por 1 dentro de un arbol AVL

    if(estado == 400){
        return new ResponseEntity<>(body: " Al menos 1 carta a insertar, tiene valor inaceptable", HttpStatus.BAD_REQUEST);
    }
    if(estado == 406){
        return new ResponseEntity<>(body: " Al menos 1 carta a insertar, esta duplicada", HttpStatus.NOT_ACCEPTABLE);
    }

    juego.actualizarNivelCarta();

    return new ResponseEntity<>(body: " Iniciando Pyramid ... Inserción de carta(s) exitosa!", HttpStatus.OK);
}

@PostMapping("Game/add")
public ResponseEntity<String> insertarArbol(@RequestBody String json) {
    int estado = juego.insertarCarta(json);

    if(estado == 406){
        return new ResponseEntity<>(body: " La carta a insertar, esta duplicada", HttpStatus.NOT_ACCEPTABLE);
    }
    if(estado == 400){
        return new ResponseEntity<>(body: " La carta a insertar, tiene valor inaceptable", HttpStatus.BAD_REQUEST);
    }

    juego.actualizarNivelCarta();

    return new ResponseEntity<>(body: " Inserción de carta(s) exitosa!", HttpStatus.OK);
}
```

Peticiones “DELETE” de clase Controlador.java:

```java
@DeleteMapping("Game/delete")
public ResponseEntity<String> eliminarArbol(@RequestBody String json) {
    int estado = juego.eliminar(json);

    if(estado == 400){
        return new ResponseEntity<>(body: " Al menos 1 carta, tiene valor inaceptable", HttpStatus.BAD_REQUEST);
    }
    if (estado == 404) {
        return new ResponseEntity<>(body: " Al menos 1 carta, no se encuentra en el árbol avl (eliminar)", HttpStatus.NOT_FOUND);
    }

    if (estado == 406) {
        return new ResponseEntity<>(body: " Los valores de las cartas no suman 13 (ni es K)", HttpStatus.NOT_ACCEPTABLE);
    }

    if (estado == 409) {
        return new ResponseEntity<>(body: " Las cartas no se puede eliminar ya que al menos 1, cuenta con hijo(s)", HttpStatus.CONFLICT);
    }

    return new ResponseEntity<>(body: " Eliminacion de carta(s) exitosa!", HttpStatus.OK);
}
}
```

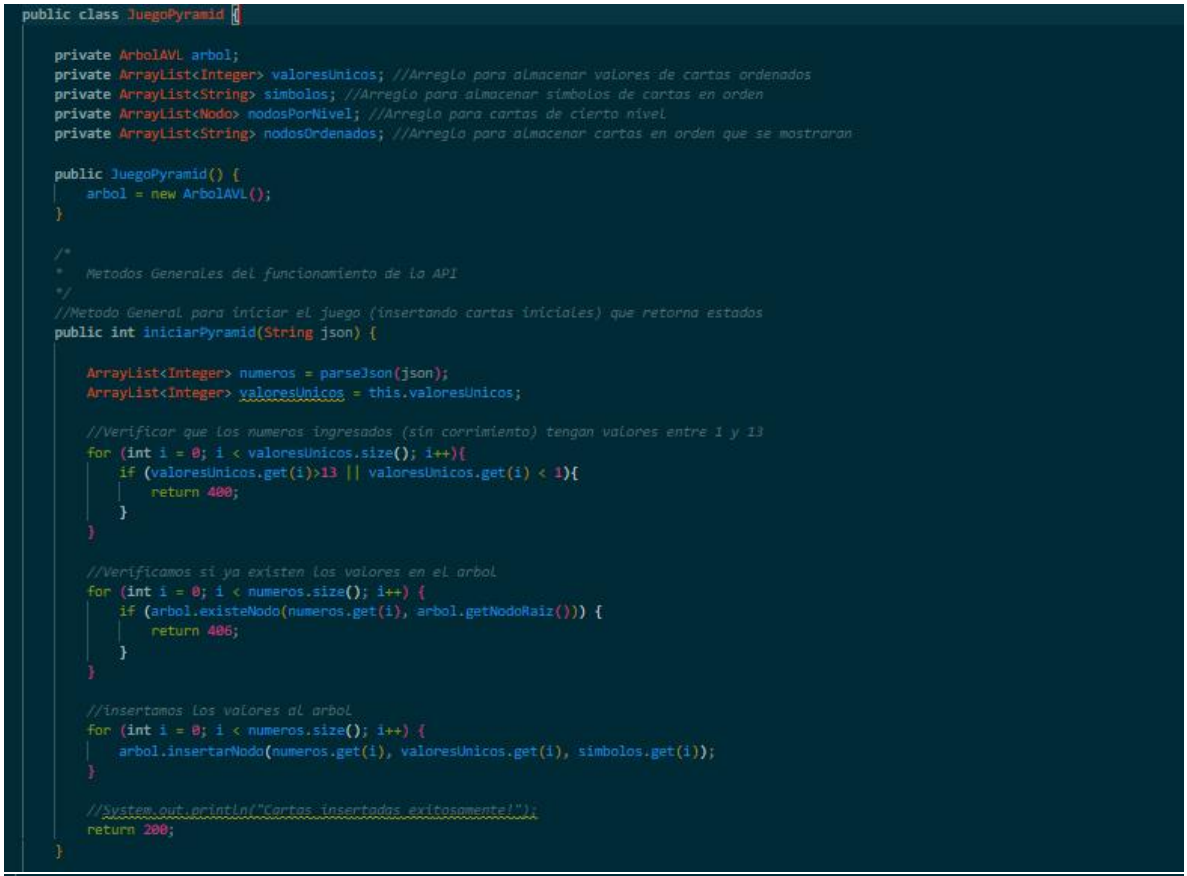
- Models: Este directorio contiene llos objetos que componen al Arbol AVL, por ende, Models es la base de la API, y es aquí donde se encuentran todos los atributos del objeto Nodo.java que es utilizado como parámetro en la mayoría de funciones de la clase ArbolAVL.java

Atributos esenciales de la clase Nodo.java: (Explicación en los comentarios del codigo)

```java
public class Nodo {

    private int valorUnico; //Valor de carta
    private int valor; //ValorUnico + ValorCorrimiento
    private String simbolo; //Caracter que se usara para sumarle ValorCorrimiento a Valor
    private int altura; //Cuantos niveles de hijos tiene
    private int nivel; //Nivel del arbol al que pertenece
    private int factEquilibrio; //hijoIzquierdo.getAltura() - hijoDerecho.getAltura()

    private boolean esHoja; //Indicador para saber si se puede eliminar
    private Nodo hijoIzq; //Enlace a carta menor izquierda
    private Nodo hijoDer; //Enlace a carta superior derecha

    public Nodo() {
    }

    //Constructor de Cartas
    public Nodo(int valor) {
        this.valor = valor;
        this.altura = 0;
        this.nivel = 0;
        this.esHoja = false;
        this.hijoIzq = null;
        this.hijoDer = null;
        this.factEquilibrio = 0;
    }
```

```java
public int getValor() {
    return valor;
}

public void setValor(int valor) {
    this.valor = valor;
}

public Nodo getHijoIzq() {
    return hijoIzq;
}

public void setHijoIzq(Nodo hijoIzq) {
    this.hijoIzq = hijoIzq;
}

public Nodo getHijoDer() {
    return hijoDer;
}

public void setHijoDer(Nodo hijoDer) {
    this.hijoDer = hijoDer;
}

public int getAltura() {
    return altura;
}

public void setAltura(int altura) {
    this.altura = altura;
}
```

- Services: Este directorio contiene toda la logica del funcionamiento del juego que simula la API, es en JuegoPyramid.java donde se adjuntan todos los metodos para el funcionamiento completo del servicio (se explican a detalle en comentario del código)



```java
public class JuegoPyramid {

    private ArbolAVL arbol;
    private ArrayList<Integer> valoresUnicos; //Arreglo para almacenar valores de cartas ordenados
    private ArrayList<String> simbolos; //Arreglo para almacenar símbolos de cartas en orden
    private ArrayList<Nodo> nodosPorNivel; //Arreglo para cartas de cierto nivel
    private ArrayList<String> nodosOrdenados; //Arreglo para almacenar cartas en orden que se mostraran

    public JuegoPyramid() {
        arbol = new ArbolAVL();
    }

    /*
    *   Metodos Generales del funcionamiento de la API
    */
    //Metodo General para iniciar el juego (insertando cartas iniciales) que retorna estados
    public int iniciarPyramid(String json) {

        ArrayList<Integer> numeros = parseJson(json);
        ArrayList<Integer> valoresUnicos = this.valoresUnicos;

        //Verificar que los numeros ingresados (sin corrimiento) tengan valores entre 1 y 13
        for (int i = 0; i < valoresUnicos.size(); i++){
            if (valoresUnicos.get(i)>13 || valoresUnicos.get(i) < 1){
                return 400;
            }
        }

        //Verificamos si ya existen los valores en el arbol
        for (int i = 0; i < numeros.size(); i++) {
            if (arbol.existeNodo(numeros.get(i), arbol.getNodoRaiz())) {
                return 406;
            }
        }

        //insertamos los valores al arbol
        for (int i = 0; i < numeros.size(); i++) {
            arbol.insertarNodo(numeros.get(i), valoresUnicos.get(i), simbolos.get(i));
        }

        //System.out.println("Cartas insertadas exitosamente!");
        return 200;
    }
```

```java
    //Metodo General para parsear datos de cartas del JSON a valores
    public ArrayList<Integer> parseJson(String json) {

        valoresUnicos = new ArrayList<>();
        simbolos = new ArrayList<>();

        //Separar body del JSON por comas
        String dataSinComas[] = json.split(regex: ",");

        //Aqui almaceno numero + text
        ArrayList<String> dataCartas = new ArrayList<>();

        //Agregar inserciones al ArrayList valoresCartas que almacenan los numeros en formato String
        for (int i = 0; i < dataSinComas.length; i++) {
            String[] dataSinPuntos = dataSinComas[i].split(regex: ":");
            dataCartas.add(dataSinPuntos[1]); //la posicion 0 es la camilla, y la posicion 1 el numero
        }
```

Los metodos mas importantes, son los que son llamados directamente de la clase JuegoPyramid.java debido a que aquí hay metodos que usan a metodos recursivos, vacíos y arreglos compuestos que han sido inicializados y manejados en la clase Nodo.java, ArbolAVL.java y quie pide la resolución o quien controla las funciones que necesita es la clase Controlador.java

JuegoPyramid.InsertarCarta():

```java
//Metodo General para insertar cartas indicadas en el JSON, a un arbol AVL que retorna estados
public int insertarCarta(String json) {

    ArrayList<Integer> numeros = parseJson(json);
    ArrayList<Integer> valoresUnicos = this.valoresUnicos;

    //Verificamos que los valores a insertar sean de valores entre 1 y 13
    for (int i = 0; i < valoresUnicos.size(); i++){
        if (valoresUnicos.get(i)>13 || valoresUnicos.get(i) < 1){
            return 400;
        }
    }
    //Verificamos si existen los valores en el arbol
    for (int i = 0; i < numeros.size(); i++) {
        if (arbol.existeNodo(numeros.get(i), arbol.getNodoRaiz())) {
            return 406;
        }
    }
    //insertamos los valores al arbol
    for (int i = 0; i < numeros.size(); i++) {
        arbol.insertarNodo(numeros.get(i), valoresUnicos.get(i), simbolos.get(i));
    }

    //System.out.println("Cartas insertadas exitosamente!");
    return 200;
}
```

JuegoPyramid.EliminarCarta():

```java
//Metodo General para eliminar cartas indicadas en el JSON, del arbol AVL que retorna estados
public int eliminar(String json) {
    ArrayList<Integer> numeros = parseJson(json);
    ArrayList<Integer> numerosSinCorrimiento = this.valoresUnicos;

    //Verificar que los numeros ingresados (sin corrimiento) tengan valores entre 1 y 13
    for (int i = 0; i < numerosSinCorrimiento.size(); i++){
        if (numerosSinCorrimiento.get(i) > 13 || numerosSinCorrimiento.get(i) < 0){
            return 400;
        }
    }
    //Verificar que los numeros ingresados (sin corrimiento) sumen 13
    if (sumaValoresEsTrece(numerosSinCorrimiento)) {
        for (int i = 0; i < numeros.size(); i++) {
            if (!arbol.existeNodo(numeros.get(i), arbol.getNodoRaiz())) {
                return 404;
            }
            Nodo nodo = arbol.buscarNodo(numeros.get(i), arbol.getNodoRaiz());
            if (!arbol.esHoja(nodo)) {
                return 409;
            }
            int eliminado = arbol.eliminarNodo(numeros.get(i), arbol.getNodoRaiz(), arbol.getNodoRaiz());
        }
        return 200;
    } else {
        return 406;
    }
}
```

Ordenamientos del arbol:

```java
public String enOrden() {
    String texto = "";
    nodosOrdenados = new ArrayList<>();
    arbol.enOrden(nodosOrdenados);
    texto += "{\n";

    for (int i = 0; i < nodosOrdenados.size(); i++) {
        texto += "\"" + i + "\":" + "\"" + nodosOrdenados.get(i) + "\"\n";
    }
    texto += "}";
    return texto;
}

public String preOrden() {
    String texto = "";
    nodosOrdenados = new ArrayList<>();
    arbol.preOrden(nodosOrdenados);
    texto += "{\n";
    for (int i = 0; i < nodosOrdenados.size(); i++) {
        texto += "\"" + i + "\":" + "\"" + nodosOrdenados.get(i) + "\"\n";
    }
    texto += "}";

    return texto;
}

public String postOrden() {
    String texto = "";
    nodosOrdenados = new ArrayList<>();
    arbol.postOrden(nodosOrdenados);
    texto += "{\n";
    for (int i = 0; i < nodosOrdenados.size(); i++) {
        texto += "\"" + i + "\":" + "\"" + nodosOrdenados.get(i) + "\"\n";
    }
    texto += "}";

    return texto;
}
```