Manual Tecnico

Para un optimo funcionamiendo del programa, se necesita de:

- 500mb de memoria ram
- Tener instalado compilador de Graphviz y java

Para este programa se usaron estructura de datos de tipo grafo y arbol B
la estructuras son las siguientes:

Grafo

Para el grafo se usaron dos objetos, uno de tipo origen y otro de tipo destino. El objeto de tipo origen contiene el nombre del nodo y un arreglo de objetos del tipo destino, esta clase destino, almacena el nombre de origen, nombre de destino, los consumos de llegada. Las clases se ven de la siguiente manera.

Origen

```
public class origen {
    private String origen;
    private destino [] destino;
    private List <String> nombresDestinos;
    public origen(String origen, destino[] destino) {
        this.origen = origen;
        this.destino = destino;
        this.nombresDestinos = new ArrayList<>();
    }
    public void agregarNombreDestino (String nombre){
        this.nombresDestinos.add(nombre);
    }
    public List <String> getNombresDestinos(){
        return nombresDestinos;
    }
    public String getOrigen() {
        return origen;
    }

    public destino[] getDestino() {
        return destino;
    }


}
```

Destino

```java
public class destino {
    private origen origen;
    private String destino;
    private double tiempo_Vehiculo;
    private double tiempo_Pie;
    private double consumo_Gas;
    private double consumo_Persona;

    public destino(String datos [],origen origenEntrada) {
        this.origen = origenEntrada;
        this.destino = datos[1];
        this.tiempo_Vehiculo = Double.parseDouble(datos[2]);
        this.tiempo_Pie = Double.parseDouble(datos[3]);;
        this.consumo_Gas = Double.parseDouble(datos[4]);;
        this.consumo_Persona = Double.parseDouble(datos[5]);;
    }
```

Arbol B

Se uso una clase que almacena el dato, los lugares o rutas, un hijo izquiero y derecho. Asi mismo un anterior que sirve para poder ir subiendo de niveles según se complete un nivel.

```java
public class estructura {
    public String [] lugar;
    public double dato;
    public int espaciosMaximosPermitidos = 5,espaciosOcuapados;
    public estructura raiz,padre,hijoI[],hijoD[],anterior[];
    public estructura getRaiz() {
        return raiz;
    }

}
```

Metodos principales de funcionamiento en arbol b

Para insertar se uso un objeto con un metodo estatico recibe una pagina de 5 espacios. Tiene tres posibles caminos, el primero que inserte en un lugar nulo, el segundo que inserte de un lugar menor al que va recorriendo, y el tercero es que inserte si este es mayor.

Primer opcion:

```
public static estructura [] insetarDato(double valor,estructura indices[],boolean
    for (int i = 0; i < indices.length; i++) {
        //System.out.println("Analizando "+indices.datos[i]+" en posicion "+i);
        if(indices[i] == null ){
            indices[i] = new estructura();
            indices[i].dato = valor;
            indices[i].lugar = lugar;
            indices[i].anterior = indices[0].anterior;
            break;
        }else
```

Segunda opcion:

```
if(valor < indices[i].dato){
    if(expancion || indices[i].hijoI == null){
        int limite = 0;
        for (int j = i; j < indices[i].espaciosMaximosPermitidos; j++) {
            if(indices[j] == null)break;
            else limite++;
        }
        System.out.println("tiene que guardar datos de "+limite);
        double datos [] = new double[limite];
        String lugares [] = null ;
        int a = 0;
        for (int j = 0; j < limite; j++) {
            datos [a] = indices[i+j].dato;
            lugares = indices[i+j].lugar;
            System.out.println("guardo "+datos[a]);
            a++;
        }
        a = 0;
        int finale = i + limite;
        for (int j = i; j <= finale; j++) {
            if(j == i){
                indices[j].dato = valor;
                indices[j].lugar = lugar;
                indices[j].anterior = indices[0].anterior;
            }else
            {
             if(indices[j]==null)indices[j] = new estructura();
             indices[j].dato = datos[a];
             indices[j].lugar = lugares;
             if(indices[0].padre!= null)indices[j].padre = indices[0].padre;
             indices[j].raiz = obtenerRaiz(indices[0]);
```

para esta opcion, si inserta en un lugar que es menor, hace un recorrido hasta encontrar un espacio nulo, desde donde tiene que insertar el lugar, luego de esto guarda los datos de la posciones donde tiene que correr los espacios. Luego de esto, inserta en el lugar donde corresponde, y va guardando los datos guardados anteriores, si encuentra un nulo entonces instancia uno nuevo.

Tercer opcion:

```
}else if(valor > indices[i].dato){
    if((i+1 < 5) && indices[i+1] == null &&indices[i].hijoD != null){
        insetarDato(valor,indices[i].hijoD,false,lugar);
        return indices;
    }
}
```

Esta opcion manda a es recursiva, hasta que haga las dos opciones anteriores de insertar.

Despues de insertar los datos, este verifica si ya tiene que hacer una expancion de pagina y utiliza el siguiente algoritmo

```
if(indices[indices.length-1] != null){
    System.out.println("va a expandir");
    double datos [] = recuperarDatos(indices);
    String lugares [] = recuperarLugares(indices);
    estructura nuevaEstructura = new estructura();

    nuevaEstructura.hijoD = new estructura[5];
    nuevaEstructura.hijoI = new estructura[5];
    nuevaEstructura.dato = datos[2];
    nuevaEstructura.lugar = lugares;
    nuevaEstructura.raiz = obtenerRaiz(indices[2]);
    int nuevoContador = 0;
    if(indices[0].anterior == null || indices[0].anterior[3] != null){
        estructura nuevoAnterior [] = new estructura[5];
        nuevoAnterior [0] = nuevaEstructura;
        nuevoAnterior [0].anterior = nuevoAnterior;
        if(indices[0].anterior == null)indices = nuevoAnterior;
        else{
            for (int i = 0; i < indices[0].anterior.length; i++) {
                if(i<4){
                    if(nuevaEstructura.dato < indices[0].anterior[i].dato){
                        indices[0].anterior[i].hijoI = nuevoAnterior;
                        if(i>0)indices[0].anterior[i-1].hijoD = nuevoAnterior;
                        break;
                    }
                }if (i==3){
                    indices[0].anterior[i].hijoD = nuevoAnterior;
                    break;
                }
            }
```

crea un nueva pagina, con sus dos hijos, despues de esto verifica si en la pagina anterior puede subir o si tiene que hacer una nueva pagina, dependiendo de estas dos opciones entonces elige y tiene los siguientes algoritmos.

```
}else {
    int limite2 = 0;
    int datosInsertar = 0;
    for (int i = 0; i < indices[0].anterior.length; i++) {
        if(indices[0].anterior[i]==null){
            indices[0].anterior[i] = nuevaEstructura;
            indices[0].anterior[i-1].hijoD = nuevaEstructura.hijoI;
            datosInsertar = 5;
            break;
        }
        if(nuevaEstructura.dato > indices[0].anterior[i].dato){
            datosInsertar++;
        }else if(nuevaEstructura.dato < indices[0].anterior[i].dato)break;
    }
    limite2 = nuevaEstructura.espaciosMaximosPermitidos - datosInsertar;
    estructura aux [] = new estructura[limite2];
    for (int i = 0; i < aux.length; i++) {
        if(indices[0].anterior[i+datosInsertar]!=null){
            aux[i] = indices[0].anterior[i+datosInsertar];
        }else break;
    }
    for (int i = 0; i < nuevaEstructura.espaciosMaximosPermitidos; i++) {
        if(i == 0){
            if(datosInsertar == 5)break;
            indices[0].anterior[i+datosInsertar] = nuevaEstructura;
            if(limite2 > 0)aux[i].hijoI = nuevaEstructura.hijoD;
            if(datosInsertar ==0 && limite2>0 &&aux[i+1]!=null)aux[i+1].hijoD = n
            else if(datosInsertar>0 && limite2>0 && aux[i]!=null){
                aux[i].hijoI = nuevaEstructura.hijoD;
                if(datosInsertar>0)indices[0].anterior[datosInsertar-1].hijoD = n
            }
```

El siguiente algoritmo llena los espacios del nuevo nodo, luego de ser insertado en una nueva pagina o en la pagina anterior, si inserto en la pagina anterior, entonces arregla los datos de donde insertara asi mismo con los punteros de sus hijos.

```
for (int i = 0; i < nuevaEstructura.espaciosMaximosPermitidos ;i++) {
    if(i<2){
        nuevaEstructura.hijoI[i] = new estructura();
        nuevaEstructura.hijoI[i].dato = datos[i];
        nuevaEstructura.hijoI[i].lugar = lugares;
        nuevaEstructura.hijoI[i].padre = nuevaEstructura;
        nuevaEstructura.hijoI[i].raiz = obtenerRaiz(indices[2]);
        nuevaEstructura.hijoI[i].anterior = nuevaEstructura.anterior;
    }else if(i!=2 && i>2){
        nuevaEstructura.hijoD[nuevoContador] = new estructura();
        nuevaEstructura.hijoD[nuevoContador].dato = datos[i];
        nuevaEstructura.hijoD[nuevoContador].lugar = lugares;
        nuevaEstructura.hijoD[nuevoContador].padre = nuevaEstructura;
        nuevaEstructura.hijoD[nuevoContador].raiz = obtenerRaiz(indices[2]);
        nuevaEstructura.hijoD[nuevoContador].anterior = nuevaEstructura.anterior;
        nuevoContador++;
    }
```

Metodos principales de grafo

La primer funcion obtiene todos los grafos que llegan hacia el, los grafos adyacentes a el.

```
public static List <String> origenesDestino(String destino,origen origeneEntrada[],List <String> listadoAnterior){
    List <String> retorno = new ArrayList<>();
    for (int i = 0; i < origeneEntrada.length; i++) {
        for (int j = 0; j < origeneEntrada[i].getDestino().length; j++) {
            if(origeneEntrada[i].getDestino()[j].getDestino().equals(destino)){
                if(listadoAnterior!= null && listadoAnterior.contains(origeneEntrada[i].getOrigen()))break;
                retorno.add(origeneEntrada[i].getOrigen());
            }
        }
    }

    return retorno;
}
```

Este metodo obtiene el arbol de las rutas que inserto, tiene 3 opciones de insertar. Este incia antes de las opciones por obtener las rutas adyacentes al desino y si estas no van directo a la posicion de donde se encuentra, entonces pasa por la primer opcion, la cual es de llamar a un metodo que verifica si este tiene alguna conexión con el lugar de origen, de serlo retorna la ruta completa con su costo. La segunda opcion es que entre los destinos de los

nodos adyacentes se encuentre el adyacente al destino. Por ultimo esta que este valla dierecto al destino que se desea

```
public static arbolB.estructura [] getRutas(List <String> datosOrigen, String posicionActual,origen origenes[],String destino,origen
    arbolB.estructura retorno [] = new estructura[5];
    List <String> listadoOrigenDestino = origenesDestino(destino, origenes,null);
    for (int i = 0; i < listadoOrigenDestino.size(); i++) {
        List <String[] > rutasObtenida = rutas(listadoOrigenDestino.get(i), origenes, origenEntrada,destino);
        if(!origenEntrada.getNombresDestinos().contains(listadoOrigenDestino.get(i)) && !origenEntrada.getOrigen().equals(listadoOri
            rutasObtenida = rutas(listadoOrigenDestino.get(i), origenes, origenEntrada,destino);
            if(rutasObtenida.size()>0){
                for (int j = 0; j < rutasObtenida.size(); j++) {
                    int posicionUltima = rutasObtenida.get(j).length-1;
                    rutasObtenida.get(j)[posicionUltima] = destino;
                }
                retorno = insertarRuta(rutasObtenida, origenes, tipoConsumoBusqueda, retorno);
                rutasObtenida = null;
            }
```

```
    }else if (origenEntrada.getNombresDestinos().contains(listadoOrigenDestino.get(i))){
        rutasObtenida = new ArrayList<>();
        String aux [] = new String[3];
        aux[0] = origenEntrada.getOrigen();
        aux[1] = listadoOrigenDestino.get(i);
        aux[2] = destino;
        rutasObtenida.add(aux);
        retorno = insertarRuta(rutasObtenida, origenes, tipoConsumoBusqueda, retorno);
        rutasObtenida = null;
    }else if (origenEntrada.getOrigen().equals(listadoOrigenDestino.get(i))){
        rutasObtenida = new ArrayList<>();
        String aux [] = new String[2];
        aux[0] = origenEntrada.getOrigen();
        aux[1] = destino;
        rutasObtenida.add(aux);
        retorno = insertarRuta(rutasObtenida, origenes, tipoConsumoBusqueda, retorno);
        rutasObtenida = null;
    }
```

Luego de analizar las rutas, este inserta la ruta al arbol si el origen y el destino estan conectados. Siendo la funcion la siguiente

```java
private static arbolB.estructura [] insertarRuta (List <String []> ruta,origen origenes[],
    arbolB.estructura retorno [] = estructuraEntrada;
    int sumador = 0;
    for (int i = 0; i < ruta.size(); i++) {
        for (int j = 0; j < ruta.get(i).length; j++) {
            if((j+1)<ruta.get(i).length){
                origen origenAux = buscarOrigen(ruta.get(i)[j], origenes);
                sumador+= obtenerDistancia(origenAux, ruta.get(i)[j+1], tipoConsumo);
            }
        }
        retorno = arbolB.insertar.insetarDato(sumador, retorno, false, ruta.get(i));
        sumador = 0;
    }
    return retorno;
}
private static double obtenerDistancia(origen origen, String destino,int tipoConsumo){
    for (int i = 0; i < origen.getDestino().length; i++) {
        if(origen.getDestino()[i].getDestino().equals(destino)){
            switch(tipoConsumo){//utiliza una case para saber que tipo de consumo quiere s
                case 0:
                    return origen.getDestino()[i].getTiempo_Vehiculo();
                case 1:
                    return origen.getDestino()[i].getTiempo_Pie();
                case 2:
                    return origen.getDestino()[i].getConsumo_Gas();
                case 3:
                    return origen.getDestino()[i].getConsumo_Persona();
            }
        }
    }
    return -1;
```