Solución algorítmica para el problema de ruteo de vehículos

Descripción del software

La descripción del software que se presenta a continuación sirve de guía para entender su funcionamiento.

Primero que todo es importante entender que el enrutamiento de vehículos es un factor importante para el optimo desarrollo de las organizaciones. Por esta razón, ha sido desarrollado el software Green-Routes (GR) 1.0, el cual permite realizar una gestión del enrutamiento de flotas de vehículos del sector de la automoción, pero que puede ser replicado en cualquier sector.

La idea de realizar el software, está basada en el problema al que se enfrentan las organizaciones en el día a día, ya que deben tener un sistema adecuado para la gestión del transporte de materia prima. Esta herramienta busca la correcta integración de la empresa y sus proveedores, ya que la materia prima es parte de la columna vertebral en una organización. Adicionalmente, la herramienta permite optimizar la distancia recorrida y calcular las emisiones de gases de efecto invernadero que han sido generadas en las rutas.

El componente principal se estructura de tal forma que existen un numero de nodos en el sistema, estos nodos se componen de proveedores y una planta de producción, que es el lugar, donde toda la materia prima recolectada de estos proveedores llega a la planta donde serán procesados para poder obtener un producto final. Para la óptima utilización del software se debe contar con la matriz de distancias entre todos los puntos involucrados en el sistema, la demanda que deberá ser recogida, y el número de vehículos con los que se cuenta.

Diseño

La herramienta GR, ha sido diseñada para el problema de enrutamiento de vehículos con capacidad homogénea, en un sistema de recolección muchos a uno, que esta basado en recolectar de diferentes puntos, para finalmente llegar a un solo punto. Es importante desatacar que existen una serie de restricciones que se aplican dentro del problema, como el numero de nodos que pueden ser visitados por ruta, la capacidad del vehículo y la distancia máxima en la que se puede desviar un vehículo de la ruta planificada. El presente documento busca ser una guía didáctica, que pueda ser consultada para el optimo entendimiento y posterior manejo del software.

Explicación paso a paso de la interfaz

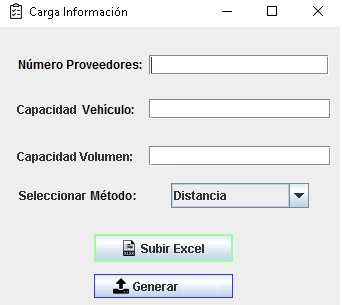
Este software se utiliza dos interfaces sencillas como se observa en la IMG 1, la primera con dos opciones que son



IMG 1

Ingreso Datos se dirige a la parte de configuración del Excel para el resultado esperado, Salir finaliza la aplicación sin calcular ningún resultado.

Seleccionamos Ingreso de datos para pasar a la interfaz de cálculo de datos.



IMG 2

Se despliega esta interfaz que contiene varios campos y un archivo para subir datos para la ejecución del cálculo.

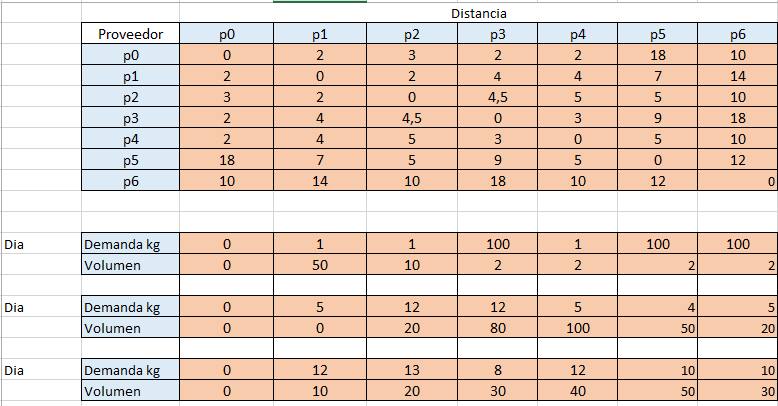
Número Proveedores: son los puntos en los cuales el vehículo va recoger la carga indicada.

Capacidad Vehículo: es la capacidad que el vehículo tiene para el trayecto que va realizar.

Capacidad Volumen: es la capacidad que el vehículo tiene para el trayecto que va realizar.

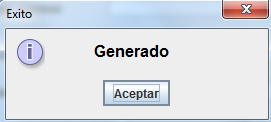
Seleccionar método: el cálculo se va realizar de acuerdo al ahorro que exista entre los puntos para poder unir las rutas a realizar el vehículo.

Subir Excel: esta parte es para subir la matriz cuadrada donde están los proveedores y las distancias entre cada uno, además la demanda y el volumen de cada proveedor adicional dentro del Excel estarán la demanda y volumen para desarrollar un cálculo por cada día, Si se sube un archivo diferente a una extensión xlsx el programa le va indicar que el archivo no es el correspondiente.



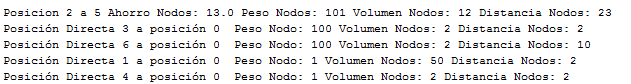
IMG 3

Generar: Despliega la salida por día que debería realizar cada vehículo, si la salida fue exitosa el programa indicara como se observa en IMG 4

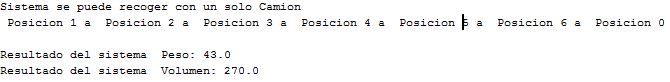


IMG 4

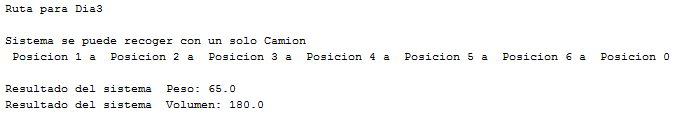
Día 1:



Día 2:

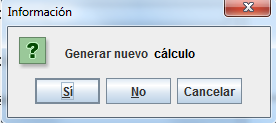


Día 3:



Se observa que cada día se comporta diferente según el Excel que se suba para los cálculos para cada vehículo.

Luego de finalizar el cálculo el programa le indicara que hacer como se observa IMG 5



IMG 5

Si se selecciona la opción NO o Cancelar el programa finaliza automáticamente pero si se selecciona SI el usuario podrá ejecutar un nuevo procedimiento diferente.

Estructura del software

Para la realización del software se ha utilizado la plataforma NETBEANS 8.2, la cual posee una licencia de libre uso.

A continuación, se presenta el código fuente utilizado para un mayor entendimiento.

Código de cálculos para rutas:

/\*\*

\*

\* @author cesard.chacond

\*/

public class MetodosCalculoDistanciaImp extends MetodosCalculoDistancia {

public MetodosCalculoDistanciaImp(double Volumen, double Capacidad) {

datos.getCapVolVehiculo().put(Capacidad\_Vehiculo, Capacidad);

datos.getCapVolVehiculo().put(Volumen\_Vehiculo, Volumen);

PesosNormal = new ArrayList<>();

PesosNosuperados = new ArrayList<>();

Aux = new ArrayList<>();

}

@Override

public boolean InicioProceso() {

SacarPesoPorDia();

String dia = null;

try {

for (int i = 0; i < datos.getPesoVolProvedores().size(); i++) {

dia = "Dia" + String.valueOf(i + 1);

System.out.println("Ruta para " + dia);

System.out.println("");

if (datos.getPesoTotalPorDia().get(dia).get(Constantes.Constantes.Demanda\_kg) < datos.getCapVolVehiculo().get(Constantes.Constantes.Capacidad\_Vehiculo)) {

System.out.println("Sistema se puede recoger con un solo Camion");

System.out.println(CrearNodosDirectos(dia));

ImprimirResultadoSistema(false, dia);

} else {

CrearPuntosArrayList(datos.getDistancias(), dia, datos.getPesoVolProvedores().size()); // unir parejas con ahorro

CompararNodosConCapacidadVehiculo(dia, datos.getCapVolVehiculo().get(Capacidad\_Vehiculo), datos.getCapVolVehiculo().get(Volumen\_Vehiculo),

datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Demanda\_kg), datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Volumen));

VerificarNodos(dia, datos.getMatrizPuntos().get(dia), datos.getNumeroProvedores(),

datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Demanda\_kg), datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Volumen));

ImprimirResultado(dia, datos.getMatrizPuntos().get(dia), datos.getDistancias(),

datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Demanda\_kg), datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Volumen));

ImprimirResultadoSistema(true, dia);

System.out.println("");

}

}

return true;

} catch (Exception e) {

return false;

}

}

@Override

public void CrearPuntosArrayList(double[][] Puntos, String dia, int numeroDias) {

ArrayList<ArrayList<Double>> CA = new ArrayList<>(); // con ahorro

ArrayList<ArrayList<Double>> SinA = new ArrayList<>(); // sin ahorro

boolean NSV = true;

boolean entreNVS = true;

try {

for (int x = 1; x < Puntos.length; x++) {

for (int y = x + 1; y < Puntos[x].length; y++) {

if (NSV) {

datos.getNodosSuperanVehiculo().put(dia, new ArrayList<>());

NSV = false;

}

if (datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Demanda\_kg).get(x) > datos.getCapVolVehiculo().get(Constantes.Constantes.Capacidad\_Vehiculo)) {

for (int z = 0; z < datos.getNodosSuperanVehiculo().get(dia).size(); z++) {

if (datos.getNodosSuperanVehiculo().get(dia).get(z) == x) {

entreNVS = false;

break;

}

}

if (entreNVS) {

datos.getNodosSuperanVehiculo().get(dia).add(x);

}

entreNVS = true;

break;

} else if (datos.getPesoVolProvedores().get(dia).get(Constantes.Constantes.Demanda\_kg).get(y) > datos.getCapVolVehiculo().get(Constantes.Constantes.Capacidad\_Vehiculo)) {

for (int z = 0; z < datos.getNodosSuperanVehiculo().get(dia).size(); z++) {

if (datos.getNodosSuperanVehiculo().get(dia).get(z) == y) {

entreNVS = false;

break;

}

}

if (entreNVS) {

datos.getNodosSuperanVehiculo().get(dia).add(y);

}

entreNVS = true;

} else {

double result = CalcularAhorros(Puntos, x, y);

if (result > 0) {

if (CAB) {

CrearListas(datos.getMatrizPuntos(), Nodos\_Con\_Ahorro, numeroDias, dia);

CAB = false;

}

CA = datos.getMatrizPuntos().get(dia).get(Nodos\_Con\_Ahorro);

CA.get(0).add((double) x);

CA.get(1).add((double) y);

CA.get(2).add(result);

datos.getMatrizPuntos().get(dia).put(Nodos\_Con\_Ahorro, CA);

} else {

if (SinAB) {

CrearListas(datos.getMatrizPuntos(), Nodos\_Sin\_Ahorro, numeroDias, dia);

SinAB = false;

}

SinA = datos.getMatrizPuntos().get(dia).get(Nodos\_Sin\_Ahorro);

SinA.get(0).add((double) x);

SinA.get(1).add((double) y);

SinA.get(2).add(result);

datos.getMatrizPuntos().get(dia).put(Nodos\_Sin\_Ahorro, SinA);

}

}

}

}

CAB = true;

SinAB = true;

System.out.println("Lista de ahorro");

System.out.println(datos.getMatrizPuntos().get(dia).get(Nodos\_Con\_Ahorro)); // cuando queda null no hay ahorro

System.out.println(datos.getMatrizPuntos().get(dia).get(Nodos\_Sin\_Ahorro));

System.out.println("");

} catch (Exception e) {

AtributosSistema.getInstance().setError("1");

}

}

@Override

public void CompararNodosConCapacidadVehiculo(String dia, Double CapacidadVehiculo, Double Volumen, ArrayList<Double> PesoProvedor, ArrayList<Double> VolumenProvedor) {

ArrayList<ArrayList<Double>> SinA = new ArrayList<>(); // sin ahorro

SinA = CalcularNodosCapacidadVehiculo(datos.getMatrizPuntos().get(dia).get(Nodos\_Con\_Ahorro),

CapacidadVehiculo, Volumen, PesoProvedor, VolumenProvedor,

datos.getMatrizPuntos().get(dia).get(Nodos\_Sin\_Ahorro), dia);

if (SinA != null) {

datos.getMatrizPuntos().get(dia).put(Nodos\_Superan\_Capacidad\_Vehiculo, SinA);

}

System.out.println("Milk Run");

System.out.println(datos.getMatrizPuntos().get(dia).get(Nodos\_Con\_Ahorro));

System.out.println("Superan la capacidad del vehiculo");

System.out.println(datos.getMatrizPuntos().get(dia).get(Nodos\_Superan\_Capacidad\_Vehiculo));

System.out.println("");

}

@Override

public void VerificarNodos(String dia, HashMap<String, ArrayList<ArrayList<Double>>> matrizPuntos, int numeroProvedores, ArrayList<Double> Peso, ArrayList<Double> Volumen) {

VerificarNodosConRuta(matrizPuntos, numeroProvedores, Peso, Volumen, dia);

System.out.println("Rutas directas");

System.out.println(datos.getMatrizPuntos().get(dia).get(Nodos\_Directos));

System.out.println("");

}

@Override

public void ImprimirResultado(String dia, HashMap<String, ArrayList<ArrayList<Double>>> matrizPuntos, double[][] distancias, ArrayList<Double> Peso, ArrayList<Double> Volumen) {

ArrayList<ArrayList<Double>> CA = new ArrayList<>(); // con ahorro

ArrayList<ArrayList<Double>> SinA = new ArrayList<>(); // puntos directos

if (matrizPuntos.get(Nodos\_Con\_Ahorro) != null) {

CA = OrdenarAhorro(matrizPuntos.get(Nodos\_Con\_Ahorro));

}

if (matrizPuntos.get(Nodos\_Directos) != null) {

SinA = matrizPuntos.get(Nodos\_Directos);

} else {

SinA = NodosDirectosSinPuntos(SinA, dia);

matrizPuntos.put(Nodos\_Directos, SinA);

SinA = matrizPuntos.get(Nodos\_Directos);

}

ImprimirPuntos(CA, SinA, distancias, Peso, Volumen, dia);

System.out.println("Funcionalidades.EvaluarPuntos.ImprimirResultado()");

}

@Override

public ArrayList<ArrayList<Double>> NodosDirectosSinPuntos(ArrayList<ArrayList<Double>> Sin, String dia) {

for (int i = 1; i <= datos.getNumeroProvedores(); i++) {

Sin.add(new ArrayList<>());

Sin.get(i - 1).add((double) i);

Sin.get(i - 1).add(datos.getPesoVolProvedores().get(dia).get("Demanda kg").get(i)); //Obtener el peso del punto

Sin.get(i - 1).add(datos.getPesoVolProvedores().get(dia).get("Volumen").get(i)); // OBtener el Volumen del punto

}

return Sin;

}

@Override

public void ImprimirResultadoSistema(boolean par, String dia) {

if (par) {

System.out.println(" ");

System.out.println("Resultado del sistema " + " Peso: " + datos.getCargaTotalSistema().get(Constantes.Constantes.PesoTotalSistema));

System.out.println("Resultado del sistema " + " Distancia: " + datos.getCargaTotalSistema().get(Constantes.Constantes.DistanciaTOtalSistema));

System.out.println("Resultado del sistema " + " Volumen: " + datos.getCargaTotalSistema().get(Constantes.Constantes.VolumenTotalSistema));

if (datos.getNodosSuperanVehiculo().size() > 0) {

for (int i = 0; i < datos.getNodosSuperanVehiculo().get(dia).size(); i++) {

System.out.println("Nodos Con cargar Mayor al vehiculo Provedor " + datos.getNodosSuperanVehiculo().get(dia).get(i));

}

}

System.out.println(" ");

} else {

System.out.println(" ");

System.out.println("Resultado del sistema " + " Peso: " + datos.getPesoTotalPorDia().get(dia).get(Constantes.Constantes.Demanda\_kg));

//System.out.println("Resultado del sistema " + " Distancia: " + atributosCarga.getPesoTotalPorDia().get(dia).get(Constantes.Constantes.DistanciaTOtalSistema));

System.out.println("Resultado del sistema " + " Volumen: " + datos.getPesoTotalPorDia().get(dia).get(Constantes.Constantes.Volumen));

// if (datos.getNodosSuperanVehiculo().size() > 0) {

// for (int i = 0; i < datos.getNodosSuperanVehiculo().get(dia).size(); i++) {

// System.out.println("Nodos Con cargar Mayor al vehiculo Provedor " + datos.getNodosSuperanVehiculo().get(dia).get(i));

// }

// }

System.out.println(" ");

}

}

@Override

public void SacarPesoPorDia() {

double resultadoPeso = 0.0;

double resultadoVolumen = 0.0;

for (int i = 1; i <= datos.getPesoVolProvedores().size(); i++) {

datos.getPesoTotalPorDia().put("Dia" + i, new HashMap<>());

for (int y = 0; y < datos.getPesoVolProvedores().get("Dia" + i).get(Constantes.Constantes.Demanda\_kg).size(); y++) {

resultadoPeso += datos.getPesoVolProvedores().get("Dia" + i).get(Constantes.Constantes.Demanda\_kg).get(y);

resultadoVolumen += datos.getPesoVolProvedores().get("Dia" + i).get(Constantes.Constantes.Volumen).get(y);

}

datos.getPesoTotalPorDia().get("Dia" + i).put(Constantes.Constantes.Demanda\_kg, resultadoPeso);

datos.getPesoTotalPorDia().get("Dia" + i).put(Constantes.Constantes.Volumen, resultadoVolumen);

resultadoPeso = 0.0;

resultadoVolumen = 0.0;

}

}

@Override

public String CrearNodosDirectos(String dia) {

String imprimir = "";

for (int i = 1; i <= datos.getNumeroProvedores(); i++) {

imprimir += " Posicion " + i + " a ";

}

return imprimir += " Posicion 0 ";

}

@Override

public ArrayList<ArrayList<Double>> OrdenarAhorro(ArrayList<ArrayList<Double>> Puntos) {

for (int i = 0; i < Puntos.get(2).size() - 1; i++) {

for (int j = 0; j < Puntos.get(2).size() - 1; j++) {

if (Puntos.get(2).get(j) < Puntos.get(2).get(j + 1)) {

double tmp0 = Puntos.get(0).get(j + 1);

Puntos.get(0).set(j + 1, Puntos.get(0).get(j));

Puntos.get(0).set(j, tmp0);

double tmp1 = Puntos.get(1).get(j + 1);

Puntos.get(1).set(j + 1, Puntos.get(1).get(j));

Puntos.get(1).set(j, tmp1);

double tmp = Puntos.get(2).get(j + 1);

Puntos.get(2).set(j + 1, Puntos.get(2).get(j));

Puntos.get(2).set(j, tmp);

}

}

}

return Puntos;

}

@Override

public double CalcularAhorros(double[][] distancia, int PuntoX, int PuntoY) {

int I = (int) PuntoX;

int J = (int) PuntoY;

return /\*distancia[0][I] +\*/ distancia[J][0] - distancia[I][J];

}

@Override

public void CrearListas(HashMap<String, HashMap<String, ArrayList<ArrayList<Double>>>> matrizPuntos, String numero, int dias, String nombreDia) {

ArrayList<ArrayList<Double>> a = new ArrayList<>();

for (int i = 0; i < 3; i++) {

a.add(new ArrayList<>());

}

if (matrizPuntos.isEmpty() || !(datos.getMatrizPuntos().containsKey(nombreDia))) {

datos.getMatrizPuntos().put(nombreDia, new HashMap<>());

datos.getImprimirNodos().put(nombreDia, new HashMap<>());

}

datos.getMatrizPuntos().get(nombreDia).put(numero, a);

// matrizPuntos.put("1", a); // Matriz con ahorro

// matrizPuntos.put("2", a); // sin ahorro

}

@Override

public boolean CalcularPeso(double PuntoX, double PuntoY, double CapVehiculo) {

double resul = 0;

boolean resulFinal = false;

resul = PuntoX + PuntoY;

if (resul <= CapVehiculo) {

resulFinal = true;

}

return resulFinal;

}

@Override

public boolean CalcularVolumen(double PuntoX, double PuntoY, double CapVolumen) {

double resul = 0;

boolean resulFinal = false;

resul = PuntoX + PuntoY;

if (resul <= CapVolumen) {

resulFinal = true;

}

return resulFinal;

}

@Override

public ArrayList<ArrayList<Double>> CalcularNodosCapacidadVehiculo(ArrayList<ArrayList<Double>> CA, Double CapacidadVehiculo, Double Volumen, ArrayList<Double> PesoProvedor,

ArrayList<Double> VolumenProvedor, ArrayList<ArrayList<Double>> SinA, String dia) {

// falta crear inicio metodos arraylist

if (CA != null) {

CrearListas();

for (int x = 0; x < CA.get(0).size(); x++) {

boolean a = CalcularPeso(PesoProvedor.get(CA.get(0).get(x).intValue()),

PesoProvedor.get(CA.get(1).get(x).intValue()), CapacidadVehiculo);

boolean b = CalcularVolumen(VolumenProvedor.get(CA.get(0).get(x).intValue()),

VolumenProvedor.get(CA.get(1).get(x).intValue()), Volumen);

if (a == false || b == false) {

PesosNosuperados.get(0).add((double) CA.get(0).get(x).intValue());

PesosNosuperados.get(1).add((double) CA.get(1).get(x).intValue());

} else {

PesosNormal.get(0).add((double) CA.get(0).get(x).intValue());

PesosNormal.get(1).add((double) CA.get(1).get(x).intValue());

PesosNormal.get(2).add((double) CA.get(2).get(x).intValue());

}

}

datos.getMatrizPuntos().get(dia).put(Nodos\_Con\_Ahorro, PesosNormal);

return PesosNosuperados;

}

return null;

}

@Override

public void CrearListas() {

PesosNormal.clear();

PesosNosuperados.clear();

for (int i = 0; i < 3; i++) {

PesosNormal.add(new ArrayList<>());

PesosNosuperados.add(new ArrayList<>());

}

}

@Override

public void VerificarNodosConRuta(HashMap<String, ArrayList<ArrayList<Double>>> matrizPuntos, int numeroProvedores, ArrayList<Double> Peso, ArrayList<Double> Volumen, String dia) {

PesosNormal = matrizPuntos.get(Nodos\_Con\_Ahorro);

PesosNosuperados = new ArrayList<>();

boolean entre = true;

int count = -1;

if (PesosNormal != null) {

if (PesosNormal.get(0).size() > 0) {

for (int i = 1; i <= numeroProvedores; i++) {

for (int y = 0; y < PesosNormal.get(0).size(); y++) {

if (PesosNormal.get(0).get(y).intValue() == i || PesosNormal.get(1).get(y).intValue() == i) {

entre = false;

break;

} else {

entre = true;

}

}

if (entre) {

for (int z = 0; z < datos.getNodosSuperanVehiculo().get(dia).size(); z++) {

if (datos.getNodosSuperanVehiculo().get(dia).get(z) == i) {

entre = false;

break;

}

}

if (entre) {

PesosNosuperados.add(new ArrayList<>());

count += 1;

PesosNosuperados.get(count).add((double) i);

PesosNosuperados.get(count).add(Peso.get(i)); //Obtener el peso del punto

PesosNosuperados.get(count).add(Volumen.get(i)); // OBtener el Volumen del punto

//entre = false;

}

}

}

/\*

Cuando un nodo tiene mas un punto

\*/

Aux = VerificarCantidadNodos(PesosNormal, numeroProvedores, Peso, Volumen); // solo queda un nodo

entre = true;

for (int z = 0; z < Aux.size(); z++) {

for (int i = 0; i < PesosNormal.get(0).size(); i++) {

System.out.println(Aux.get(z).get(0).intValue() + " == " + PesosNormal.get(0).get(i).intValue() + " 0 " + Aux.get(z).get(0).intValue() + " == " + PesosNormal.get(1).get(i).intValue());

if (Aux.get(z).get(0).intValue() == PesosNormal.get(0).get(i).intValue() || Aux.get(z).get(0).intValue() == PesosNormal.get(1).get(i).intValue()) {

entre = false;

break;

} else {

entre = true;

}

}

if (entre) {

for (int p = 0; p < datos.getNodosSuperanVehiculo().get(dia).size(); p++) {

if (datos.getNodosSuperanVehiculo().get(dia).get(p) == Aux.get(z).get(0).intValue()) {

entre = false;

break;

}

}

if (entre) {

PesosNosuperados.add(Aux.get(z));

}

}

}

datos.getMatrizPuntos().get(dia).put(Nodos\_Directos, PesosNosuperados);

datos.getMatrizPuntos().get(dia).put(Nodos\_Con\_Ahorro, PesosNormal);

} else {

count = -1;

for (int i = 1; i <= numeroProvedores; i++) {

PesosNosuperados.add(new ArrayList<>());

count += 1;

PesosNosuperados.get(count).add((double) i);

PesosNosuperados.get(count).add(Peso.get(i)); //Obtener el peso del punto

PesosNosuperados.get(count).add(Volumen.get(i)); // OBtener el Volumen del punto

}

datos.getMatrizPuntos().get(dia).put(Nodos\_Directos, PesosNosuperados);

datos.getMatrizPuntos().get(dia).put(Nodos\_Con\_Ahorro, PesosNormal);

}

}

}

@Override

public void ImprimirPuntos(ArrayList<ArrayList<Double>> CA, ArrayList<ArrayList<Double>> SinA, double[][] distancias,

ArrayList<Double> Peso, ArrayList<Double> Volumen, String dia) {

double resultadoSistemaPeso = 0.0;

double resultadoSistemaDistancia = 0.0;

double resultadosSistemaVolumen = 0.0;

int count = 0;

PesosNormal = new ArrayList<>();

PesosNosuperados = new ArrayList<>();

if (CA.size() > 0) {

for (int i = 0; i < CA.get(0).size(); i++) {

// PesosNormal.get(i).addAll();

PesosNormal.add(new ArrayList<>());

PesosNormal.get(i).add(CA.get(0).get(i)); // pos

PesosNormal.get(i).add(CA.get(1).get(i)); // pos

PesosNormal.get(i).add(CA.get(2).get(i)); // ahorro

PesosNormal.get(i).add(CalcularPesoImprimir(Peso.get(CA.get(0).get(i).intValue()), Peso.get(CA.get(1).get(i).intValue()), 0));

PesosNormal.get(i).add(CalcularVolumenImprimir(Volumen.get(CA.get(0).get(i).intValue()), Volumen.get(CA.get(1).get(i).intValue()), 0));

PesosNormal.get(i).add(CalcularDistancia(distancias, CA.get(0).get(i).intValue(), CA.get(1).get(i).intValue()) + retornarDistanciaCero(CA.get(1).get(i).intValue()));

}

}

if (SinA.size() > 0) {

boolean entre;

for (int i = 0; i < SinA.size(); i++) {

entre = true;

for (int z = 0; z < datos.getNodosSuperanVehiculo().get(dia).size(); z++) {

if (datos.getNodosSuperanVehiculo().get(dia).get(z) == SinA.get(i).get(0).intValue()) {

entre = false;

break;

}

}

if (entre) {

PesosNosuperados.add(new ArrayList<>());

PesosNosuperados.get(count).add(SinA.get(i).get(0)); // pos

PesosNosuperados.get(count).add(SinA.get(i).get(1)); // peso

PesosNosuperados.get(count).add(SinA.get(i).get(2)); // volumen

PesosNosuperados.get(count).add(CalcularDistancia(distancias, PesosNosuperados.get(count).get(0).intValue(), Integer.parseInt("0"))); // distancia

count += 1;

}

}

}

datos.getImprimirNodos().get(dia).put(Nodos\_Con\_Ahorro, PesosNormal);

datos.getImprimirNodos().get(dia).put(Nodos\_Directos, PesosNosuperados);

for (int i = 0; i < datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).size(); i++) {

System.out.println(

" Posicion " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(0).intValue()

+ " a " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(1).intValue()

+ " Ahorro Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(2)

+ " Peso Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(3).intValue()

+ " Volumen Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(4).intValue()

+ " Distancia Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(5).intValue()

);

resultadoSistemaPeso += datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(3);

resultadoSistemaDistancia += datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(5);

resultadosSistemaVolumen += datos.getImprimirNodos().get(dia).get(Nodos\_Con\_Ahorro).get(i).get(4);

}

for (int i = 0; i < datos.getImprimirNodos().get(dia).get(Nodos\_Directos).size(); i++) {

System.out.println(

" Posición Directa " + datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(0).intValue()

+ " a posición 0 "

+ " Peso Nodo: " + datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(1).intValue()

+ " Volumen Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(2).intValue()

+ " Distancia Nodos: " + datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(3).intValue()

);

resultadoSistemaPeso += datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(1);

resultadoSistemaDistancia += datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(3);

resultadosSistemaVolumen += datos.getImprimirNodos().get(dia).get(Nodos\_Directos).get(i).get(2);

}

datos.getCargaTotalSistema().put(Constantes.Constantes.PesoTotalSistema, resultadoSistemaPeso);

datos.getCargaTotalSistema().put(Constantes.Constantes.DistanciaTOtalSistema, resultadoSistemaDistancia);

datos.getCargaTotalSistema().put(Constantes.Constantes.VolumenTotalSistema, resultadosSistemaVolumen);

}

@Override

public double CalcularPesoImprimir(double PuntoX, double PuntoY, double CapVehiculo) {

double resul = 0;

resul = PuntoX + PuntoY;

return resul;

}

@Override

public double CalcularVolumenImprimir(double PuntoX, double PuntoY, double CapVolumen) {

double resul = 0;

resul = PuntoX + PuntoY;

return resul;

}

@Override

public double CalcularDistancia(double[][] distancia, int PuntoX, int PuntoY) {

int I = (int) PuntoX;

int J = (int) PuntoY;

return distancia[I][J];

}

private ArrayList<ArrayList<Double>> VerificarCantidadNodos(ArrayList<ArrayList<Double>> PesosNormall, int numeroProvedores, ArrayList<Double> Peso, ArrayList<Double> Volumen) {

ArrayList<Integer> countX = new ArrayList<>();

ArrayList<Integer> countY = new ArrayList<>();

ArrayList<ArrayList<Double>> directo = new ArrayList<>();

ArrayList<ArrayList<Double>> auxV = new ArrayList<>();

for (int i = 0; i < 3; i++) {

auxV.add(new ArrayList<>());

}

int contar = -1;

for (int i = 1; i <= numeroProvedores; i++) {

for (int y = 0; y < PesosNormall.get(0).size(); y++) {

if (PesosNormall.get(0).get(y).intValue() == i) {

countX.add(y);

}

if (PesosNormall.get(1).get(y).intValue() == i) {

countY.add(y);

}

}

if (countX.size() > 1) {

double result = 0;

int pos = 0;

for (int b = 0; b < countX.size(); b++) {

if (b == 0) {

result = PesosNormall.get(2).get(countX.get(b));

pos = b;

}

if (PesosNormall.get(2).get(countX.get(b)) > result) {

result = PesosNormall.get(2).get(countX.get(b));

pos = countX.get(b);

}

}

for (int b = 0; b < countX.size(); b++) {

if (b != pos) {

directo.add(new ArrayList<>());

contar += 1;

directo.get(contar).add(PesosNormall.get(1).get(countX.get(b)));

directo.get(contar).add(Peso.get(PesosNormall.get(1).get(countX.get(b)).intValue()));

directo.get(contar).add(Volumen.get(PesosNormall.get(1).get(countX.get(b)).intValue()));

}

}

for (int b = 0; b < countX.size(); b++) {

if (b != pos) {

for (int z = 0; z < 3; z++) {

PesosNormall.get(z).set(countX.get(b), 0.0);

// auxV.get(z).add(PesosNormall.get(z).get(pos));

}

}

}

}

if (countY.size() > 1) {

double result = 0;

int pos = 0;

for (int b = 0; b < countY.size(); b++) {

if (b == 0) {

result = PesosNormall.get(2).get(countY.get(b));

pos = countY.get(b);

}

if (PesosNormall.get(2).get(countY.get(b)) > result) {

result = PesosNormall.get(2).get(countY.get(b));

pos = countY.get(b);

}

}

for (int b = 0; b < countY.size(); b++) {

if (countY.get(b).intValue() != pos) {

directo.add(new ArrayList<>());

contar += 1;

directo.get(contar).add(PesosNormall.get(0).get(countY.get(b)));

directo.get(contar).add(Peso.get(PesosNormall.get(0).get(countY.get(b)).intValue()));

directo.get(contar).add(Volumen.get(PesosNormall.get(0).get(countY.get(b)).intValue()));

}

}

for (int b = 0; b < countY.size(); b++) {

if (countY.get(b).intValue() != pos) {

for (int z = 0; z < 3; z++) {

PesosNormall.get(z).set(countY.get(b), 0.0);

// auxV.get(z).add(PesosNormall.get(z).get(pos));

}

}

}

countX = new ArrayList<>();

countY = new ArrayList<>();

} else {

countX = new ArrayList<>();

countY = new ArrayList<>();

}

}

for (int y = 0; y < PesosNormall.get(0).size(); y++) {

if (PesosNormall.get(0).get(y) == 0.0 || PesosNormall.get(1).get(y) == 0.0 || PesosNormall.get(2).get(y) == 0.0) {

PesosNormall.get(0).remove(y);

PesosNormall.get(1).remove(y);

PesosNormall.get(2).remove(y);

y = 0;

}

}

for (int y = 0; y < PesosNormall.get(0).size(); y++) {

if (PesosNormall.get(0).get(y) == 0.0 || PesosNormall.get(1).get(y) == 0.0 || PesosNormall.get(2).get(y) == 0.0) {

PesosNormall.get(0).remove(y);

PesosNormall.get(1).remove(y);

PesosNormall.get(2).remove(y);

y = 0;

}

}

for (int y = 0; y < directo.size(); y++) {

if (directo.get(y).get(0) == 0.0) {

directo.remove(y);

y = 0;

}

}

PesosNormal = PesosNormall;

return directo;

}

public double retornarDistanciaCero(int pos) {

return datos.getDistancias()[0][pos];

}

}