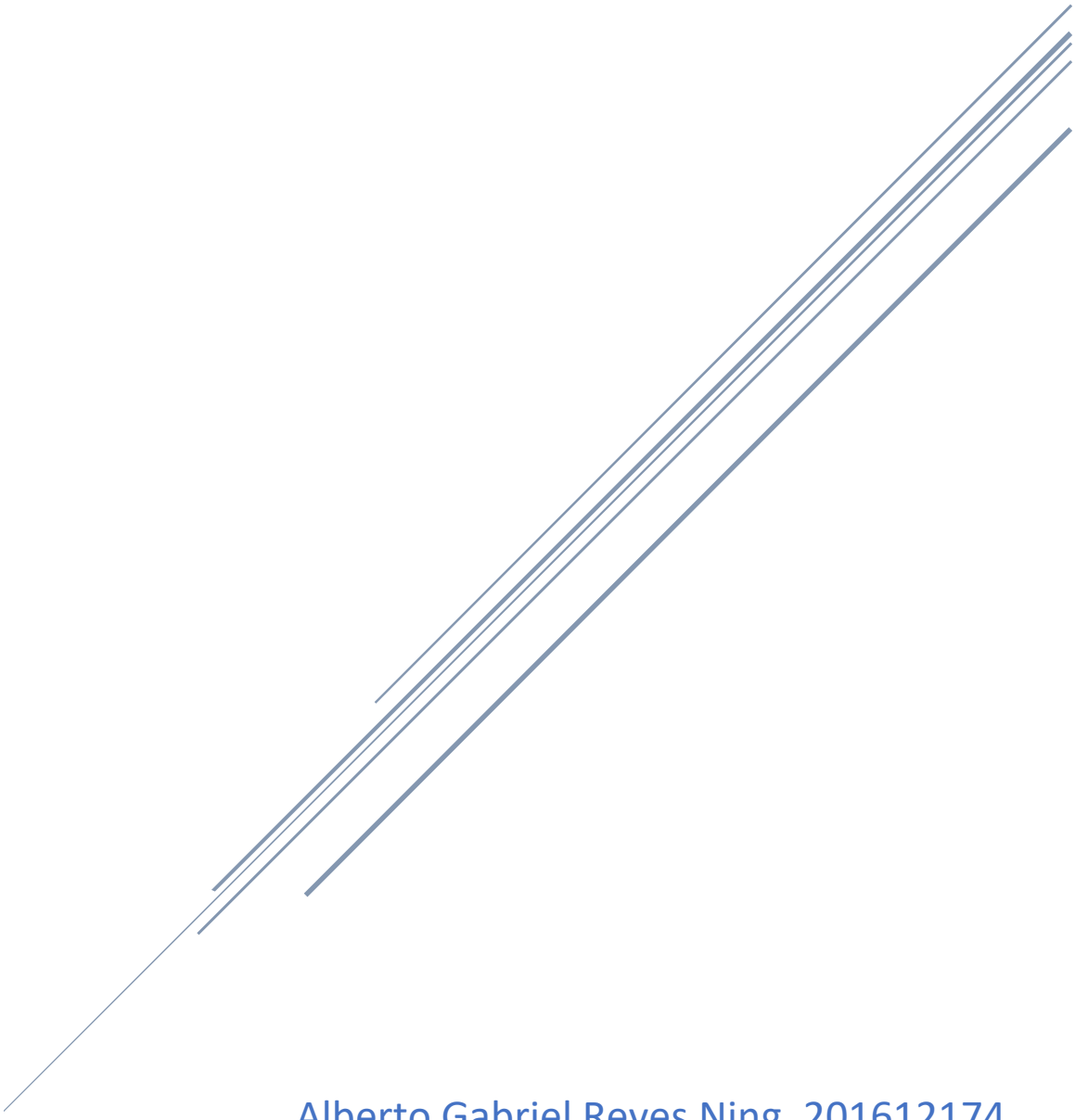
MANUAL TECNICO

EDD PROYECTO FASE 3



Alberto Gabriel Reyes Ning, 201612174
Universidad de San Carlos de Guatemala

Contents

Objetivos del Sistema

Desarrollar una aplicación utilizando las estructuras de datos y sus algoritmos explicados en clase, de tal forma que pueda simular los diferentes procesos que se dan en la empresa. Dicha aplicación deberá ser capaz de representar las estructuras de forma visual, mediante la utilización de bibliotecas soportadas. Desea realizar una aplicación de generar imágenes utilizando diferentes estructuras y sus diferentes recorridos.

Especificación Técnica

Requisitos de Hardware

- Memoria RAM: 4 GBs o Superior
- Procesador: Intel Celeron o Superior

Requisitos de Software

Sistema Operativo

- Windows 8 o Superior

Lenguajes de Programación e IDE

- Java Versión 8 Update 271
- NetBeans 8.2

Librerías Usados

- JsonSimple 1.1
- GraphViz

Lógica del Programa

Clases:

Estructuras Utilizados:

- Tabla Hash
- Lista Simple
- Lista Adyacencia

Nodos:

- Node_Clientes
- Node
- Hash

Interfaz:

- Login
- Administrador
- Usuario
- Modificar

Main:

- Main
- Carga_Masiva
- GraphViz

Clase Main

Contiene una llamada al interfaz de la aplicación, en este caso, el login. También inicializa la estructura donde guardan los usuarios.

```
public class Main {

    public static Lista lugares = new Lista();
    public static Lista_Adyacencia rutas = new Lista_Adyacencia();
    public static Tabla_Hash hash_ = new Tabla_Hash(0);
    public static Lista clientes = new Lista();

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        Login log = new Login(clientes, lugares, rutas, hash_, "");
        log.setVisible(true);

    }
}
```

Clase Carga Masiva

Esta clase contiene cuatro funciones para cargar diferentes tipos de archivos. También se guardan los archivos creados en carpetas creadas por estas funciones.

Carga Masiva: Mensajeros

```
public static void loadFromFile(File f, Tabla_Hash hash) throws FileNotFoundException, IOException, ParseException { //Carga Masiv

    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();
    while (iterator.hasNext()) {
        JSONObject m = iterator.next();
        String dpi = (String) m.get("dpi");
        String nombre = (String) m.get("nombres");
        String apellidos = (String) m.get("apellidos");
        String tipo_Licencia = (String) m.get("tipo_licencia");
        String genero = (String) m.get("genero");
        String direccion = (String) m.get("direccion");
        String telefono = (String) m.get("telefono");
        hash.insertar(dpi, nombre, apellidos, tipo_Licencia, genero, direccion, telefono);
        if (hash.carga > 0.75) {
            Tabla_Hash hash_0 = new Tabla_Hash(hash.size + 1);
            for (int i = 0; i < hash.primos[hash.size]; i++) {
                if (hash.hash.dpi[i] == null) {
                    continue;
                } else {
                    hash_0.insertar(hash.hash.dpi[i], hash.hash.nombres[i], hash.hash.apellidos[i], hash.hash.tipo_Licencia[i], ha
                }
            }
            hash = hash_0;
        }
    }
    hash.graficar();
}
```

Carga Masiva: Rutas

```
public static void loadFromFile(File f, Lista_Adyacencia rutas, Lista lugares) throws FileNotFoundException, IOException, ParseEx
    JSONParser jsonParser = new JSONParser();
    Object obj = jsonParser.parse(new FileReader(f));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray jsonArray = (JSONArray) jsonObject.get("Grafo");

    for (Object o : jsonArray) {
        JSONObject m = (JSONObject) o;
        Long inicio = (Long) m.get("inicio");
        Long fin = (Long) m.get("final");
        Long peso = (Long) m.get("peso");
        rutas.insertar(Math.toIntExact(inicio), Math.toIntExact(fin), Math.toIntExact(peso));
    }
    rutas.graficar();
    rutas.graficar(lugares);
    //rutas.ruta(8, 16);
}
```

Carga Masiva: Lugares

```
public static void loadFromFile(File f, Lista lugares) throws FileNotFoundException, IOException, ParseException { //Carga Masiva
    JSONParser jsonParser = new JSONParser();
    Object obj = jsonParser.parse(new FileReader(f));
    JSONObject jsonObject = (JSONObject) obj;
    JSONArray jsonArray = (JSONArray) jsonObject.get("Lugares");

    for (Object o : jsonArray) {
        JSONObject m = (JSONObject) o;
        Long id = (Long) m.get("id");
        String dept = (String) m.get("departamento");
        String nombre = (String) m.get("nombre");
        String sn_suc = (String) m.get("sn_sucursal");
        lugares.insert(Math.toIntExact(id), dept, nombre, sn_suc.equals("si") ? true : false);
    }
    lugares.graficar(0);
}
```

Carga Masiva: Clientes

```
public static void loadFromFile(File f, Lista usuarios, int x) throws FileNotFoundException, IOException, ParseException {//Carga
    JSONParser jsonParser = new JSONParser();
    JSONArray jsonArray = (JSONArray) jsonParser.parse(new FileReader(f));
    Iterator<JSONObject> iterator = jsonArray.iterator();

    while (iterator.hasNext()) {
        JSONObject m = iterator.next();
        String dpi = (String) m.get("dpi");
        String nombre_Completo = (String) m.get("nombre_completo");
        String nombre_Usuario = (String) m.get("nombre_usuario");
        String correo = (String) m.get("correo");
        String pass = (String) m.get("contrasenia");
        String tel = (String) m.get("telefono");
        String direccion = (String) m.get("direccion");
        Long id = (Long) m.get("id_municipio");
        usuarios.insert(dpi, nombre_Completo, nombre_Usuario, correo, pass, tel, direccion, Math.toIntExact(id));
    }
    usuarios.graficar(1);
}
```

Estructuras:

Tabla Hash

Esta estructura contiene un constructor para crear un nuevo ABB, una función “insert”, dos funciones de hash, una para el primer hash y el otro de doble dispersión, una función para buscar un mensajero y unas funciones para crear la imagen de la tabla hash.

Constructor

```
public Tabla_Hash(int size) {
    this.size = size;
    int amount = this.primos[this.size];
    hash = new Hash(amount);
    this.agregados = 0;
}
```

Insertar

```
public void insertar(String dpi, String nombre, String apellidos, String tipo_Licencia, String genero, String direccion, String te
    long llave = Long.parseLong(dpi);
    int llave_ = getHash(llave);

    if (hash.dpi[llave_] == null) {
        hash.dpi[llave_] = dpi;
        hash.nombres[llave_] = nombre;
        hash.apellidos[llave_] = apellidos;
        hash.tipo_Licencia[llave_] = tipo_Licencia;
        hash.genero[llave_] = genero;
        hash.direccion[llave_] = direccion;
        hash.telefono[llave_] = telefono;
    } else {
        for (int i = 1; i < 100; i++) {
            llave_ = getHash(llave, i);
            if (llave_ > this.primos[this.size]) {
                llave_ = llave_ % this.primos[this.size];
            }
            if (hash.dpi[llave_] == null) {
                hash.dpi[llave_] = dpi;
                hash.nombres[llave_] = nombre;
                hash.apellidos[llave_] = apellidos;
                hash.tipo_Licencia[llave_] = tipo_Licencia;
                hash.genero[llave_] = genero;
                hash.direccion[llave_] = direccion;
                hash.telefono[llave_] = telefono;
                break;
            }
        }
    }
    agregados++;
    carga = (double) agregados / (this.primos[this.size]);
}
```

Funciones de Hash

```
private int getHash(long llave) {
    int key = (int) (llave % this.primos[this.size]);
    return key;
}

private int getHash(long llave, int i) {
    int key = (int) (((llave % 7) + 1) * i);
    return key;
}
```

Buscar Mensajero

```
public void buscar(String dpi) {
    long llave = Long.parseLong(dpi);
    int llave_ = getHash(llave);
    if (hash.dpi[llave_].equals(dpi)) {
        System.out.println("Encontrado: " + hash.dpi[llave_] + " en posicion: " + llave_);
    } else {
        for (int i = 1; i < 100; i++) {
            llave_ = getHash(llave, i);
            if (llave_ > this.primos[this.size]) {
                llave_ = llave_ % this.primos[this.size];
            }
            if (hash.dpi[llave_].equals(dpi)) {
                System.out.println("Encontrado: " + hash.dpi[llave_] + " en posicion: " + llave_);
                break;
            }
        }
    }
}
```

Graficar Tabla Hash

```
public void graficar() {
    try {
        FileWriter myWriter = new FileWriter("src\\salidas\\Tabla_Hash.txt");//
        myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\n");
        myWriter.write(graficadora(hash));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir("Tabla_Hash");//"Tabla_Hash"
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}

public String graficadora(Hash hash) {
    String cadena = "";
    cadena += "n1 [label = < \n<table>\n<tr><td colspan = \"2\" >Mensajeros</td></tr>";
    for (int i = 0; i < hash.dpi.length; i++) {
        cadena += "\n<tr><td> Nombre:" + hash.nombres[i] + " " + hash.apellidos[i] + "</td><td>" + hash.dpi[i] + "</td></tr>";
    }
    cadena += "\n</table>\n> ]";
    return cadena;
}
```

Lista Adyacencia

La estructura de Lista Adyacencia está compuesta de una lista de listas. La clase contiene dos funciones de insertar, una función de aumentar el tamaño de la lista, una función para encontrar la mejor ruta entre nodos y varias funciones para graficar diferentes imágenes.

Constructor:

```
public Lista_Adyacencia() {
    inicio = null;
    fin = null;
    routes = new int[10];
    size = 0;
}
```

Insertar 1:

```
public void insertar(int start, int end, int peso) { //Insert function that adds the dest for each start point
    if (start > end && start > size) {
        increase(start);
    } else if (end > start && end > size) {
        increase(end);
    }

    Node temp = inicio;
    while (temp.id != start) {
        temp = temp.next;
    } //temp should be at the start position now
    temp.dest.insertar_Dest(end);
    temp.peso.insertar_Dest(peso);

    temp = inicio;
    while (temp.id != end) {
        temp = temp.next;
    } //temp should be at the start position now
    temp.dest.insertar_Dest(start);
    temp.peso.insertar_Dest(peso);
}
```

Insertar 2:

```
public void insertar_Dest(int end) { //Insert function just to add to the end of the list
    Node newNode = new Node(end);
    if (this.inicio == null) {
        inicio = newNode;
    } else {
        Node temp = inicio;
        while (temp.next != null) {
            temp = temp.next;
        } //Should be at the end of the dest list
        temp.next = newNode;
    }
}
```

Aumentar Tamano:

```
public void increase(int bigger) {
    if (inicio == null) {
        Node newNode = new Node(1);
        inicio = newNode;
        size++;
    }

    Node temp = inicio;
    while (temp.id != size) {
        temp = temp.next;
    }

    while (temp.id != bigger) {
        Node newNode = new Node(temp.id + 1);
        temp.next = newNode;
        size++;
        temp = temp.next;
    }
}
```

Funcion para encontrar el mejor ruta:

```
public void ruta(int start, int end) {
    best = new Lista();
    Lista route = new Lista();
    Node temp = inicio;
    while (temp.id != start) {
        temp = temp.next;
    }//Should be at start position now

    route.insert(temp.id);
    //Recursion
    try {
        route_Rec(temp.id, end, route, temp.peso.inicio);
        System.out.println("Path Found");
        best.display();
    } catch (Exception ignored) {
        System.out.println("");
    }

    if(routes[0] == 0){
        routes[0] = best.peso_Total;
        this.graficar(routes);
    } else {
        if(routes[9] == 0){
            for(int i = 0; i < 10; i++){
                if(routes[i] == 0){
                    routes[i] = best.peso_Total;
                    this.graficar(routes);
                }
            }
        } else {
            Arrays.sort(routes);
            if(routes[0] < best.peso_Total){
                routes[0] = best.peso_Total;
                Arrays.sort(routes);
                this.graficar(routes);
```

```
                this.graficar(routes);
            }
        }
    }
}

public void route_Rec(int curr, int end, Lista route, Node peso) {
    if (curr == 0) {
        return;
    }

    if (curr == end) {
        if (best.peso_Total == 0) {
            best = new Lista();
            best.copy(route);
            System.out.println("\nFOUND ROUTE1: " + best.peso_Total);
        }
        if (route.peso_Total < best.peso_Total) {
            best = new Lista();
            best.copy(route);
            System.out.println("\nFOUND ROUTE2: " + best.peso_Total);

        }
        route.pop(peso.id);
        return;
    }//Checks if we are at the final destination if not, checks next route

    Node temp = inicio;
    while (temp.id != curr) {
        temp = temp.next;
    }//Should be at current position now

    if (temp.dest.inicio == null) {
        route.pop(peso.id);
        System.out.println("\nFOUND DEAD END: " + route.peso_Total);
        return;
```

```
        return;
    }

    if (route.peso_Total > best.peso_Total && best.peso_Total != 0) {
        route.pop(peso.id);
        System.out.println("\nFOUND INEFFICIENT ROUTE: " + route.peso_Total);
        return;
    }

    Node rand = temp.dest.inicio;
    Node rand_ = temp.peso.inicio;
    while (rand != null && rand_ != null) {
        System.out.println("RAND: " + rand.id);
        //Check if dest is already in the route
        if (route.buscar(rand.id)) {
            rand = rand.next;
            rand_ = rand_.next;
            continue;
        }
        route.insert(rand.id, rand_.id);
        route_Rec(rand.id, end, route, rand_);
        //route_Rec(dest.next, peso.next, end, best, route);

        rand = rand.next;
        rand_ = rand_.next;
        System.out.println("\nFINDING ROUTE: " + route.peso_Total);
    }
    if (route.inicio != null) {
        route.pop(peso.id);
    }
    return;
```

Funciones para Graficar diferentes imagenes

```
public void graficar(Lista lugares) {
    try {
        FileWriter myWriter = new FileWriter("src\\Salidas\\Rutas.txt");
        myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\nedge [arrowhead= non
        myWriter.write(graficadora(inicio, lugares));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir("Rutas");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public String graficadora(Node inicio, Lista lugares) {
    String cadena = "";

    if (this.size == 0) {
        System.out.println("Error: Rutas Vacio");
        return cadena;
    }
    Node temp = this.inicio;
    String nombre;
    while (temp != null) {
        //cadena += "\nnodo" + temp.id;
        nombre = lugares.buscar_Nombre(temp.id);
        cadena += "\nnodo" + temp.id + " [label = \"" + (nombre.equals("") ? temp.id : nombre) + " | ID: " + temp.id + "\"]";
        if (temp.dest.inicio == null) {
            temp = temp.next;
            continue;
        } else {
            Node temp_ = temp.dest.inicio;
            Node rand = temp.peso.inicio;
            while (temp_ != null) {
                if (temp_.id > temp.id) {
                    cadena += "\nnodo" + temp.id + " -> nodo" + temp_.id + "[label = \"" + rand.id + "\"]";
                }
                temp_ = temp_.next;
                rand = rand.next;
            }
        }
        temp = temp.next;
    }
    return cadena;
}
```

```
public void graficar() {
    try {
        FileWriter myWriter = new FileWriter("src\\Salidas\\Lista_Adyacencia.txt");
        myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\nedge [arrowhead= none
        myWriter.write(graficadora(inicio));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir("Lista_Adyacencia");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public String graficadora(Node inicio) {
    String cadena = "";

    if (this.size == 0) {
        System.out.println("Error: Rutas Vacio");
        return cadena;
    }
    Node temp = this.inicio;
    while (temp != null) {
        cadena += "\nnodo" + temp.id + " [label = \"ID: " + temp.id + "\"]";
        if (temp.dest.inicio == null) {
            temp = temp.next;
            continue;
        } else {
            Node temp_ = temp.dest.inicio;
            cadena += "\nnodo" + temp.id + " -> nodo" + temp.id + "_" + temp_.id;
            cadena += "\nnodo" + temp.id + "_" + temp_.id + "[label = \"ID: " + temp_.id + "\"]";
            while (temp_.next != null) {

                cadena += "\nnodo" + temp.id + "_" + temp_.id + "[label = \"ID: " + temp_.id + "\"]";
                cadena += "\nnodo" + temp.id + "_" + temp_.next.id + "[label = \"ID: " + temp_.next.id + "\"]";
                cadena += "\nnodo" + temp.id + "_" + temp_.id + " -> nodo" + temp.id + "_" + temp_.next.id;

                temp_ = temp_.next;
            }
        }
        temp = temp.next;
    }

    return cadena;
}
```

```
public void graficar(int[] routes_) {
    try {
        FileWriter myWriter = new FileWriter("src\\Salidas\\RRR.txt");
        myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\nedge [arrowhead= non
        myWriter.write(graficadora(routes_));
        myWriter.write("}");
        myWriter.close();
        System.out.println("Successfully wrote to the file.");
        GraphViz.imprimir("RRR");
    } catch (IOException e) {
        System.out.println("An error occurred.");
        e.printStackTrace();
    }
}
```

```
public String graficadora(int[] routes_) {
    String cadena = "";

    for(int i = 0; i < routes_.length; i++){
        cadena += "\nnodo" + i + " [label = \"" + routes_[i] + "\"]";
    }

    for(int i = 0; i < routes_.length - 1; i++){
        cadena += "\nnodo" + i + " -> ";
    } cadena += "\nnodo" + 9;

    return cadena;
}
```

Lista:

Las listas se usan para guardar los rutas y clientes. Están compueste de nodos y son simplemente enlazadas. Se cuenta con dos constructores, una función para copiar los nodos de otra lista, cinco funciones para insertar dependiendo de la lista, una función para sacar el ultimo nodo de la lista, tres funciones de búsqueda dependiendo del tipo de lista, y varias funciones para graficar según los requisitos.

Constructors:

```
public Lista() {
    inicio = null;
    fin = null;
    peso_Total = 0;
}

public Lista(Lista route) {
    inicio = null;
    fin = null;
    peso_Total = 0;
    this.copy(route);
}
```

Función para copiar nodos de otra lista.

```
public void copy(Lista route) {
    peso_Total = route.peso_Total;
    Node temp = route.inicio;
    while (temp != null) {
        this.insert(temp.id);
        temp = temp.next;
    }
}
```

Funciones para insertar:

```java
public void insert(int id) {
    Node newNode = new Node(id);
    if (inicio == null) {
        inicio = newNode;
    } else if (fin == null) {
        fin = newNode;
        inicio.next = fin;
    } else {
        fin.next = newNode;
        fin = newNode;
    }
}
```

```java
//Insert function for Client list
public void insert(String dpi, String nombre_Completo, String nombre_Usuario, String correo, String pass, String tel, String dire
    Node_Clientes newNode = new Node_Clientes(dpi, nombre_Completo, nombre_Usuario, correo, pass, tel, direccion, id_Municipio);
    if (start == null) {
        start = newNode;
    } else if (end == null) {
        end = newNode;
        start.next = end;
    } else {
        end.next = newNode;
        end = newNode;
    }
}

public void insert(int id, String departamento, String nombre, Boolean sn_sucursal) {
    Node newNode = new Node(id, departamento, nombre, sn_sucursal);
    if (inicio == null) {
        inicio = newNode;
    } else if (fin == null) {
        fin = newNode;
        inicio.next = fin;
    } else {
        fin.next = newNode;
        fin = newNode;
    }
}
```

```java
public void insert(int id, int peso) {
    Node newNode = new Node(id);
    peso_Total += peso;
    if (inicio == null) {
        inicio = newNode;
    } else if (fin == null) {
        fin = newNode;
        inicio.next = fin;
    } else {
        fin.next = newNode;
        fin = newNode;
    }
}

public void insertar_Dest(int end) { //Insert function just to add to the end of the list
    Node newNode = new Node(end);
    if (this.inicio == null) {
        inicio = newNode;
    } else {
        Node temp = inicio;
        while (temp.next != null) {
            temp = temp.next;
        } //Should be at the end of the dest list
        temp.next = newNode;
    }
}
```

Funciones para sacar el ultimo nodo en la lista:

```
public void pop(int peso) {
    if (this.inicio == null) {
        System.out.println("Error: Ruta Vacio");
        return;
    }

    Node temp = this.inicio;
    while (temp.next != fin) {
        temp = temp.next;
    }
    temp.next = null;
    fin = temp;
    peso_Total -= peso;
}
```

Funciones para buscar:

```java
public boolean buscar(int id) {
    if (inicio == null) {
        System.out.println("Error: Lista Vacio");
        return false;
    }
    Node temp = inicio;
    while (temp != null) {
        if (temp.id == id) {
            return true;
        }
        temp = temp.next;
    }
    return false;
}

public String buscar_Nombre(int id) { //Store Name
    if (inicio == null) {
        System.out.println("Error: Lista Vacio");
        return "";
    }
    Node temp = inicio;
    while (temp != null) {
        if (temp.id == id) {
            return temp.nombre;
        }
        temp = temp.next;
    }
    return "";
}
```

```java
public Node_Clientes buscar_Cliente(String dpi){
    if (start == null) {
        System.out.println("Error: Lista Vacio");
        return null;
    }
    Node_Clientes temp = start;
    while (temp != null) {
        System.out.println("Searching: " + temp.dpi);
        if (temp.dpi.equals(dpi)) {
            return temp;
        }
        temp = temp.next;
    }
    return null;
}
```

Funciones para graficar:

```
public void graficar(int x) { //Graficar lista de lugares
    try {

        if (x == 0) {
            FileWriter myWriter = new FileWriter("src\\Salidas\\Lugares.txt");
            myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\n");
            myWriter.write(graficadora(inicio));
            myWriter.write("}");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
            GraphViz.imprimir("Lugares");
        } else if(x==1){
            FileWriter myWriter = new FileWriter("src\\Salidas\\Clientes.txt");
            myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\n");
            myWriter.write(graficadora(start));
            myWriter.write("}");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
            GraphViz.imprimir("Clientes");
        } else if(x==2){
            FileWriter myWriter = new FileWriter("src\\Salidas\\Best.txt");
            myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\n");
            myWriter.write(graficadora(inicio));
            myWriter.write("}");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
            GraphViz.imprimir("Best");
        } else if(x==3){
            FileWriter myWriter = new FileWriter("src\\Salidas\\RRR.txt");
            myWriter.write("digraph structs\n{\nrankdir=\"TB\"\nlabel=\"Carnet: 201612174\"\nnode [shape=none];\n");
            myWriter.write(graficadora(inicio));
            myWriter.write("}");
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
            GraphViz.imprimir("RRR");
```

```
                myWriter.close();
                System.out.println("Successfully wrote to the file.");
                GraphViz.imprimir("RRR");
            }

        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

```java
public String graficadora(Node inicio) {
    String cadena = "";

    if (this.inicio == null) {
        System.out.println("Error: Rutas Vacio");
        return cadena;
    }
    Node temp = this.inicio;
    while (temp.next != null) {
        //cadena += "\nnodo" + temp.id;
        cadena += "\nnodo" + temp.id + " [label = \"ID: " + temp.id + " | Nombre: " + temp.nombre + "\"]";
        cadena += "\nnodo" + temp.id + " -> " + "nodo" + temp.next.id;
        temp = temp.next;
    }
    cadena += "\nnodo" + temp.id + " [label = \"ID: " + temp.id + " | Nombre: " + temp.nombre + "\"]";
    return cadena;
}
```

```java
public String graficadora(Node_Clientes inicio) {
    String cadena = "";

    if (this.start == null) {
        System.out.println("Error: Rutas Vacio");
        return cadena;
    }
    Node_Clientes temp = this.start;
    while (temp.next != null) {
        //cadena += "\nnodo" + temp.id;
        cadena += "\nnodo" + temp.dpi + " [label = \"" + temp.nombre_Completo + " | " + temp.dpi + "\"]";
        cadena += "\nnodo" + temp.dpi + " -> " + "nodo" + temp.next.dpi;
        temp = temp.next;
    }
    cadena += "\nnodo" + temp.dpi + " [label = \"" + temp.nombre_Completo + " | " + temp.dpi + "\"]";
    return cadena;
}
```