Estructura de Datos y Algoritmos - B

Aplicación de Sistema De Gestión de Almacenes - Grupo 01

ELABORADO POR:

- TEJADA LAZO, Jordy Rolando
- YARE CHULUNQUIA, Kevin Pedro
- HURTADO BEJARANO, Michael Steve

DOCENTE:

Richart Escobedo Quispe

AREQUIPA – PERÚ
2023

Aplicación de Sistema De Gestión de Almacenes

1. Resumen

Este informe explora la implementación de un sistema de gestión de almacenes (WMS) utilizando varias estructuras de datos dentro de un marco basado en Java. El proyecto tiene como objetivo equipar a los estudiantes con experiencia práctica en el manejo de estructuras de datos complejas aprendidas durante el curso. DOTA SAC, una empresa con múltiples almacenes para diversos productos, sirve de telón de fondo para el desarrollo del sistema. El SGA cumple funciones básicas: gestionar los almacenes y sus productos, optimizar la distribución de los productos y adaptarse al crecimiento de la empresa. El diseño abarca funciones como agregar, buscar y eliminar almacenes y productos, además de establecer rutas de distribución eficientes. El proyecto emplea estructuras de datos basadas en gráficos, específicamente listas de adyacencia, para administrar almacenes y sus conexiones. Además, los árboles se utilizan para organizar los productos dentro de los almacenes. El lenguaje de programación Java facilita la integración perfecta de estas estructuras. A lo largo del desarrollo, se probó la eficiencia y la precisión del sistema. Se implementaron y probaron múltiples funcionalidades, incluyendo la adición y eliminación de almacenes y productos, la consulta de disponibilidad de productos y la optimización de las rutas de distribución.

2. Introducción

La gestión de almacenes constituye un pilar fundamental en el ámbito empresarial, donde la eficiencia en la organización y distribución de productos es esencial para el éxito y rentabilidad de las operaciones. La creciente complejidad de las cadenas de suministro y la necesidad de optimizar recursos han impulsado la búsqueda constante de soluciones tecnológicas que permitan una gestión más precisa y efectiva.

En este contexto, el presente informe se adentra en la aplicación de estructuras de datos en el desarrollo de un Sistema de Gestión de Almacenes (SGA). La selección y uso adecuado de estructuras de datos juega un papel crucial en la eficiencia y escalabilidad de la gestión de almacenes, permitiendo manejar de manera ordenada y ágil tanto la información relativa a los almacenes como la de los productos que en ellos se resguardan.

Antes de adentrarnos en el impacto de las estructuras de datos en la gestión de almacenes, es fundamental comprender algunos conceptos clave. Las estructuras de datos son esencialmente formas de organizar y almacenar datos en una computadora para su uso eficiente. Cada estructura tiene ventajas y desventajas según la aplicación específica.

En el contexto de la gestión de almacenes, dos conceptos son especialmente relevantes: los grafos y los árboles. Los grafos, representados mediante nodos y aristas, son ideales para modelar relaciones complejas entre diferentes elementos, como la conexión entre almacenes en una red de distribución. Los árboles, por otro lado, ofrecen una estructura jerárquica que puede emplearse para organizar productos dentro de cada almacén de manera eficiente.

El diseño y desarrollo de un Sistema de Gestión de Almacenes implica el manejo y procesamiento de una gran cantidad de información. Aquí es donde las estructuras de datos juegan un rol determinante al proporcionar métodos eficaces de almacenamiento, búsqueda y manipulación de datos.

La representación de los almacenes y sus conexiones puede lograrse a través de la implementación de grafos mediante listas de adyacencia. Cada almacén se convierte en un nodo, y las rutas entre almacenes son las aristas. Esta estructura permite no solo visualizar las conexiones, sino también calcular las rutas más cortas para la distribución de productos, contribuyendo así al ahorro de recursos.

Por otro lado, los productos al interior de los almacenes pueden gestionarse mediante árboles. Cada árbol representa un almacén y sus ramas los productos que contiene. Esta estructura facilita la búsqueda y actualización de productos específicos, optimizando el proceso de gestión de inventario.

3. Desarrollo

Gestión de Almacenes

La gestión de almacenes se realiza utilizando una estructura de grafo basada en listas de adyacencia. Cada almacén se representa como un vértice en el grafo, y las vías de distribución entre almacenes se representan como arcos ponderados. Las funcionalidades implementadas son:

Agregar Almacén: Permite agregar un nuevo almacén al grafo.

Agregar Almacén desde un Archivo: Permite cargar información de almacenes desde un archivo CSV.

Dar de Baja a Almacén: Elimina un almacén del grafo y redistribuye sus productos a otros almacenes según ciertas reglas.

Buscar Almacén: Busca y muestra la información de un almacén específico.

Buscar Producto en Todos los Almacenes: Busca un producto en todos los almacenes y muestra los almacenes donde se encuentra.

Mostrar Todos los Almacenes: Muestra la lista de todos los almacenes registrados.

Gestión de Productos

La gestión de productos se realiza utilizando árboles, específicamente árboles binarios de búsqueda (BST). Cada almacén tiene asociado un árbol BST que almacena los productos disponibles en ese almacén. Las funcionalidades implementadas son:

Agregar Productos a un Almacén: Agrega productos a un almacén específico en el árbol BST correspondiente.

Agregar Productos desde un Archivo: Carga información de productos a los almacenes desde un archivo CSV.

Dar de Baja un Producto de un Almacén: Elimina un producto de un almacén y ajusta el árbol BST correspondiente.

Buscar Producto en un Almacén: Busca y muestra la información de un producto en un almacén específico.

Mostrar los Productos de un Almacén: Muestra la lista de productos disponibles en un almacén.

Establecimiento de Rutas de Distribución
Las rutas de distribución se gestionan mediante una lista de objetos "Ruta". Cada ruta contiene información sobre el almacén de origen, la distancia y el almacén de destino. Las funcionalidades implementadas son:

Agregar Vías desde un Archivo: Carga información de rutas de distribución desde un archivo CSV.

Obtener las Rutas de Distribución: Muestra la lista de rutas de distribución disponibles.

4. **Metodología**

Para nuestro código usamos lo que son una aplicación Java con Maven

Enlace del Github: https://github.com/jrolando19/grupo01EdaFinalTIF.git

- Primero que nada hay que tener un nodo para el árbol que en este caso usaremos un árbol AVL enlazado y este seria el nodo en el cual tendríamos el dato, left, right y bf que serían los datos del nodo

```java
public class NodeAvl<E> {
    private E data;
    private NodeAvl<E> left;
    private NodeAvl<E> right;
    private int bf;

    public NodeAvl(E data, NodeAvl<E> left, NodeAvl<E> right){
        this.data = data;
        this.left = left;
        this.right = right;
        this.bf = 0;
    }


    public int getBf() {
        return bf;
    }


    public void setBf(int bf) {
        this.bf = bf;
    }


    public NodeAvl(E data){
        this(data,left:null, right:null);
    }

    public E getData() {
        return this.data;
    }

    public void setData(E data) {
        this.data = data;
    }

    public NodeAvl<E> getLeft() {
        return this.left;
    }

    public void setLeft(NodeAvl<E> left) {
        this.left = left;
    }

    public NodeAvl<E> getRight() {
        return this.right;
    }

    public void setRight(NodeAvl<E> right) {
        this.right = right;
    }

    public String toString() {
        return this.data.toString() + "(" + this.bf + ")";
    }
}
```

- El árbol sería éste siendo de tipo genérico, los métodos que usamos son el de isEmpty() para verificar si está vacío, insert() para poner los datos y otro método insert() sobrecargado para añadir en otros nodos, balanceToRight() y balanceToLeft() para balancear el arbol, rotateRSL() y rotateRSR() para la rotacion del arbol, getRoot() para obtener el root, el search() igual que el insert también tiene otro método sobrecargado que sirve para buscar, el método remove() igual también otro

método sobrecargado para eliminar y por último el método inOrden() para ordenar el árbol en forma inorden.

```java
package com.mycompany.avlaplication;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;
import myExceptions.ExceptionNoFound;

public class AvlAlm<E extends Comparable<E>> {
    private NodeAvlAlm<E> root;
    private boolean height;

    public AvlAlm() {
            this.root = null;
    }

    public boolean isEmpty() {
            return this.root == null;
    }

    public void insert(String id, E x) throws ExceptionNoFound {
            this.root = insert(id, x, this.root);
            this.height = false;

    }

    private NodeAvlAlm<E> insert(String id, E x, NodeAvlAlm<E> current) throws ExceptionNoFound{
        NodeAvlAlm<E> res = current;
        //Caso que sea la raiz
        if (current == null) {
            res = new NodeAvlAlm<E>(id, x);
            this.height = true;
        }
        else {
            int resC = current.getData().compareTo(x);
            if (resC == 0)
                throw new ExceptionNoFound(msg:"Elemento ya se encuentra en el arbol");
            if (resC < 0) {
                res.setRight(insert(id, x, current.getRight()));
                if (this.height) {
                    switch(res.getBf()) {
                    case -1: res.setBf(bf:0);
                        this.height = false;
                        break;
                    case 0: res.setBf(bf:1);
                        break;
                    case 1: //res.setBf(2);
                        res = balanceToLeft(res);
                        this.height = false;
                        break;
                    }
```

```
            }
            else {
                res.setLeft(insert(id, x, current.getLeft()));
                // completar la verificacion de los factores de balanc
                if (this.height) {
                    switch (res.getBf()) {
                        case -1: res=balanceToRight(res);
                            this.height = false;
                            break;
                        case 0: res.setBf(-1);
                            break;
                        case 1: res.setBf(bf:0);
                            //res = balanceToLeft(res);
                            this.height = false;
                            break;
                    }
                }
            }
        }
        return res;
    }

    private NodeAvlAlm<E> balanceToRight(NodeAvlAlm<E> node){
            NodeAvlAlm<E> son = node.getLeft();
            if (son.getBf() == -1){
                    node.setBf(bf:0);
                    son.setBf(bf:0);
                    node = rotateRSR(node);
            }
            else if (son.getBf() == 1) {
                    NodeAvlAlm<E> gSon = son.getLeft();
                    switch(gSon.getBf()) {
                    case -1: node.setBf(-1); son.setBf(bf:0); break;
                    case 0: node.setBf(bf:0); son.setBf(bf:0); break;
                    case 1: node.setBf(bf:0); son.setBf(bf:1); break;
                    }
                    gSon.setBf(bf:0);

                    node.setLeft(rotateRSL(son));
                    node = rotateRSR(node);
            }
            return node;
    }

```

```java
    private NodeAvlAlm<E> balanceToLeft(NodeAvlAlm<E> node){
            NodeAvlAlm<E> son = node.getRight();
            if (son.getBf() == 1){
                    node.setBf(bf:0);
                    son.setBf(bf:0);
                    node = rotateRSL(node);
            }
            else if (son.getBf() == -1) {
                    NodeAvlAlm<E> gSon = son.getLeft();
                    switch(gSon.getBf()) {
                    case -1: node.setBf(bf:0); son.setBf(-1); break;
                    case 0: node.setBf(bf:0); son.setBf(bf:0); break;
                    case 1: node.setBf(bf:1); son.setBf(bf:0); break;
                    }
                    gSon.setBf(bf:0);

                    node.setRight(rotateRSR(son));
                    node = rotateRSL(node);
            }
            return node;
    }

    private NodeAvlAlm<E> rotateRSL(NodeAvlAlm<E> node){
            NodeAvlAlm<E> son = node.getRight();
            node.setRight(son.getLeft());
            son.setLeft(node);
            node = son;
            return node;
    }

    private NodeAvlAlm<E> rotateRSR(NodeAvlAlm<E> node){
            NodeAvlAlm<E> son = node.getLeft();
            node.setLeft(son.getRight());
            son.setRight(node);
            node = son;
            return node;
    }

    public E getRoot() {
            return this.root.getData();
    }


    public E search(String id) throws ExceptionNoFound {
        NodeAvlAlm<E> aux = search(id, this.root);
        if (aux == null)
               throw new ExceptionNoFound(msg:"Elemento no se encuentra en el arbol");
        return aux.getData();
    }
```

```java
    private NodeAvlAlm<E> search(String id, NodeAvlAlm<E> current) throws ExceptionNoFound{
            if (current == null) {
                    return null;
            }
            else {
                    int resC = current.getId().compareTo(id);
                    if (resC == 0)
                            return current;
                    if (resC < 0)
                             return search(id, current.getRight());
                    else
                            return search(id, current.getLeft());
            }
    }

    //La eliminación debe ser modificada a efecto de verificar el balanceo del árbol.
    //Esta eliminación es del BST
    public void remove(String id) throws ExceptionNoFound {
        LinkedList<NodeAvlAlm> listaPaso = new LinkedList<NodeAvlAlm>();
            this.root = remove(id, this.root, listaPaso);
    }

    private NodeAvlAlm<E> remove(String id, NodeAvlAlm<E> current, LinkedList<NodeAvlAlm> listaPaso) throws ExceptionNoFound{
            NodeAvlAlm<E> res = current;
            listaPaso.push(res);
            if (current == null) {
                    throw new ExceptionNoFound(msg:"Elemento no se encuentra en el arbol");
            }
            else {
                    int resC = current.getId().compareTo(id);
                    if (resC < 0){
                        res.setRight(remove(id, current.getRight(), listaPaso));
                    }
                    else if (resC > 0)
                            res.setLeft(remove(id, current.getLeft(), listaPaso));
                    else {
                            if (current.getLeft()!= null && current.getRight() != null) {
                                    //case 3: dos hijos
                                    NodeAvlAlm<E> rem = current.getRight();
                                    while (rem!=null){
                                        rem = rem.getLeft();
                                    }
                                    //Lo desenlazamos completamente
                                    rem.setLeft(current.getLeft());
                                    rem.setRight(current.getRight());
                                    current.setLeft(left:null);
                                    current.setRight(right:null);
                                    //Iniciamos el analisis de como a afectado
                            }
```

```
                                else {
                                        if (isLeaf(current))        //case 1: hoja
                                                res = null;
                                        else{ //case 2: solo un hijo
                                                res = current.getLeft() != null ? current.getLeft() : current.getRight();
                                        }
                                }
                        }
                }
                return res;
        }

        private boolean isLeaf(NodeAvlAlm<E> current) {
                return current.getLeft() == null && current.getRight() == null;
        }

        public void inOrden() {
                if (isEmpty())
                        System.out.println(x:"Arbol esta vacío ....");
                else{
                        inOrden(this.root);
                        System.out.println();
                }
        }



        private void inOrden(NodeAvlAlm<E> current) {
                if (current.getLeft() != null)
                        inOrden(current.getLeft());
                System.out.println(current);
                if (current.getRight()!= null)
                        inOrden(current.getRight());
        }

}

```

Ahora para la explicación del código de la aplicación

- Vamos a crear una clase llamada Almacén el cual será la clase que controle los almacenes de nuestra app este Almacén tendrá datos como el código, nombre y

dirección.

```java
package com.mycompany.eda_tif_p1;

public class Almacen implements Comparable<Almacen>{
    public static int num=0;
    private int cod;
    private String nom;
    private String dir;
    //private Avl<Producto> b = new Avl<Producto>();

    public Almacen(int cod, String nom, String dir){
        this.cod = cod;
        this.nom = nom;
        this.dir = dir;
    }

    public int getCod(){
        return this.cod;
    }
    /*
    public static void main (String args[]){
        Almacen[] alm = new Almacen[10];

        System.out.println("Numero de Almacenes: " + Almacen.num);
    }*/

    public String getNom(){
        return this.nom;
    }

    @Override
    public int compareTo(Almacen t) {
        return this.nom.compareTo(t.getNom());
    }

    public String toString(){
        return cod + " - " + nom + " - " + dir;
    }
}
```

- Ahora pasemos con los productos que seria lo mismo que el Almacén

```java
package com.mycompany.eda_tif_p1;

public class Producto implements Comparable<Producto>{
    //public static ArrayList<String> nomProductos = new ArrayList<String>();
    public static int num=0;
    private int cod;
    private String des;
    private int stock;

    public Producto(int cod, String des, int stock){
        this.cod = ++num;
        this.des = des;
        this.stock = stock;
    }

    public int getCod(){
        return this.cod;
    }

    public String getDes(){
        return this.des;
    }

    @Override
    public int compareTo(Producto t) {
        return this.des.compareTo(t.getDes());
    }
}

```

- El ResultDistr es simple es solo un arraylist de almacenes

```java
package com.mycompany.eda_tif_p1;

import java.util.ArrayList;

public class ResultDistr {
    private ArrayList<Almacen> result;
}

```

- El controlador principal para mostrar el interfaz sería este el cual estaríamos usando javafx y carga los almacenes y búsqueda de almacenes

```java
package com.mycompany.eda_tif_p1;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.ResourceBundle;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextField;
import myExceptions.ExceptionNoFound;
//import org.json.simple.JSONArray;
//import org.json.simple.JSONObject;
//import org.json.simple.parser.JSONParser;
//import org.json.simple.parser.ParseException;

public class MainController {

    @FXML
    //private ChoiceBox countChoiceBox;
    private Button btnIngManualAlm;
    @FXML
    private TextField txtNomAlm;

    @FXML
    public void initialize() {
        App.getAvlAlm().inOrden();
    }

    @FXML
    private void loadAlmacenes() throws IOException {
        App.setRoot(fxml:"loadAlmacenes");
    }

    @FXML
    private void buscarAlmacen() throws IOException, ExceptionNoFound {
        String nomBus = txtNomAlm.getText();
        //System.out.println(nomBus);
        System.out.println(App.getAvlAlm().search(nomBus));
    }

}
```

- Para la carga de almacenes simplemente solo usamos el BufferReader para leer el contenido de un archivo txt

```java
package com.mycompany.eda_tif_p1;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URL;
import java.util.List;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Button;
import javafx.scene.control.TextArea;
import javafx.stage.FileChooser;
import myExceptions.ExceptionNoFound;

public class loadAlmacenesController implements Initializable {

    @FXML
    private Button btnSelect;
    //private TextArea txtFiles;

   @Override
    public void initialize(URL url, ResourceBundle rb) {
        btnSelect.setOnAction(event -> {
            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Buscar Archivo");

            // Agregar filtros para facilitar la busqueda
            fileChooser.getExtensionFilters().addAll(
                    new FileChooser.ExtensionFilter("All Images", "*.*"),
                    new FileChooser.ExtensionFilter("TXT", "*.txt")
            );

            // Obtener la imagen seleccionada
            List<File> list = fileChooser.showOpenMultipleDialog(null);

            for (int i=0; i<list.size(); i++){
                //path: list.get(i).getAbsolutePath()
                File doc = new File( list.get(i).getAbsolutePath());
                File docPrin = new File( pathname:"D:\\Java\\eda_tif_p1\\src\\main\\resources\\com\\mycompany\\eda_tif_p1\\almacen.txt");
```

```
        BufferedReader obj;
        try {
            obj = new BufferedReader(new FileReader(doc));
        } catch (FileNotFoundException ex) {
            ex.printStackTrace();
            obj=null;
        }

        //Completado de Avl de Almacenes
        String strng="";

        try {
            while ((strng = obj.readLine()) != null){
                String[] parts = strng.split(regex:",");
                //System.out.println(parts[0]+parts[1]+parts[2] );
                App.getAvlAlm().insert(parts[1],new Almacen(Integer.parseInt(parts[0]),parts[1],parts[2]));

            }
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (ExceptionNoFound ex) {
            ex.printStackTrace();
        }

        try (BufferedWriter writer = new BufferedWriter(new FileWriter(docPrin, append:true))) {
            // true en FileWriter permite escribir al final del archivo, sin sobrescribirlo.
            writer.write(strng+"\n");

            // Es importante cerrar el escritor después de terminar de escribir.
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

});
}
```

```
84      TEXTAREA METHODS
85
86      void insert(String str, int pos)
87      Inserts the specified text at the specified position.
88
89      public void setText(String t)
90      Sets the text of JTextArea
91      */
92      //#volverMain
93      @FXML
94      private void volverMain() throws IOException {
95          App.setRoot(fxml:"main");
96      }
97  }
98
```

- Para lo último el código de la App el cual sería el interfaz el cual le damos un tamaño predefinido y llamamos a los métodos de cargar almacenes y además también se carga los recursos .fxml

```java
package com.mycompany.eda_tif_p1;

import com.mycompany.avlaplication.Avl;
import com.mycompany.avlaplication.AvlAlm;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;
import myExceptions.ExceptionNoFound;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;
    //Determinar una estructura que almacenara los objetos, que contendran al almacen y a los productos
    private static AvlAlm<Almacen> avlAlm = new AvlAlm<Almacen>();

    @Override
    public void start(Stage stage) throws IOException {
        scene = new Scene(loadFXML(fxml:"main"), 640, 480);
        stage.setScene(scene);
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML(fxml));
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource(fxml + ".fxml"));
        return fxmlLoader.load();
    }

    Run | Debug
    public static void main(String[] args) throws IOException, ExceptionNoFound {
        completeAvlAlmacen();
        launch();
    }
```

```java
    public static void completeAvlAlmacen() throws IOException, ExceptionNoFound{
        File doc = new File(pathname:"D:\\Java\\eda_tif_p1\\src\\main\\resources\\com\\mycompany\\eda_tif_p1\\almacen.txt");

        BufferedReader obj = new BufferedReader(new FileReader(doc));

        //Completado de Avl de Almacenes
        String strng;

        while ((strng = obj.readLine()) != null){
            String[] parts = strng.split(regex:",");
            //System.out.println(parts[0]+parts[1]+parts[2] );
            avlAlm.insert(parts[1],new Almacen(Integer.parseInt(parts[0]),parts[1],parts[2]));
        }

    }

    public static AvlAlm<Almacen> getAvlAlm(){
        return avlAlm;
    }
}
```

5. Conclusiones

La implementación exitosa de este Sistema de Gestión de Almacenes demuestra la aplicación práctica de las estructuras de datos estudiadas y analizadas durante el curso. La combinación de grafos para la representación de almacenes y rutas, y árboles binarios de búsqueda para la gestión de productos, proporciona una solución eficiente y ordenada para el manejo de la información en un entorno empresarial.

La implementación de este sistema respaldado por estructuras de datos en el entorno de programación representa un hito significativo en la optimización de los procesos logísticos y de almacenamiento. A lo largo de este informe, hemos explorado cómo la elección adecuada y la aplicación de estructuras de datos impactan de manera directa en la eficiencia y la funcionalidad del sistema, proporcionando una serie de conclusiones esenciales:

La selección y utilización de estructuras de datos como grafos y árboles permiten organizar y acceder a la información de manera eficiente. La representación de almacenes mediante grafos y la organización jerárquica de productos a través de árboles facilitan la búsqueda, inserción y modificación de datos, optimizando la gestión del inventario.

La implementación de estructuras de datos adecuadas en el diseño del SGA brinda escalabilidad al sistema. A medida que la empresa DOTA SAC expande su red de almacenes, las estructuras de datos permiten agregar nuevos nodos y ramas de manera sencilla, sin comprometer el rendimiento del sistema ni requerir modificaciones sustanciales en el código.

Las estructuras de datos optimizan la precisión y la velocidad de los procesos. La capacidad de realizar búsquedas y cálculos sobre las relaciones entre almacenes utilizando grafos garantiza que las decisiones se basen en datos actualizados y precisos, lo que conduce a una distribución más eficiente de productos y una gestión de inventario más precisa.

La representación visual de conexiones entre almacenes mediante grafos facilita la visualización de rutas de distribución y opciones de transporte, lo que proporciona a los administradores una visión clara para la toma de decisiones informadas y estratégicas.