SISTEMA DE SIMULACION DE TRÁFICO

DOCUMENTACION TECNICA

Presentación

La siguiente documentación ha sido desarrollada con la finalidad de dar a conocer la información necesaria para realizar el mantenimiento, instalación y exploración del software del **Sistema de Simulación de Trafico**, el cual consta de diferentes funcionalidades. El manual ofrece la información necesaria de ¿Cómo está desarrollado el software? Para que la persona (Desarrollador en Java) que quiera editar el software lo haga de una manera apropiada, dando a conocer la estructura del desarrollo del aplicativo.

Resumen

El manual detalla los aspectos técnicos e informativos del software del **Sistema de Simulación de Trafico** con la finalidad de explicar la estructura del aplicativo al personal que quiera administrarlo, editarlo o configurarlo. La siguiente guía se encuentra dividida en las herramientas que se usaron para el desarrollo del software con una breve explicación paso a paso. El aplicativo maneja diferentes funcionalidades los cuales se explicara que funcionamiento realiza cada uno de ellos, dando sugerencias para el debido uso del sistema informático.

Introducción

El manual se realiza con el fin de detallar el software para el **Sistema de Simulación de Trafico** en términos técnicos para aquella persona que vaya a administrar, editar o configurar el aplicativo lo haga de una manera apropiada. El documento se encuentra dividido en las siguientes secciones:

- ENTORNO DE DESARROLLO: Se darán a conocer las herramientas de desarrollo que se utilizaron para el software, así como se darán breves explicaciones de instalaciones y configuraciones necesarias de las herramientas anteriormente expuestas
- COMPLEJIDAD ALGORITMICA DE CADA ALGORITMO IMPLEMENTADO EN EL PROYECTO: Se describirá la complejidad algorítmica de los métodos que fueron implementados dentro del código fuente de la aplicación y que tienen una gran relevancia en Estructura De Datos
- DESCRIPCION DE LAS ESTRUCTURAS DE DATOS: Se explicaran las estructuras de datos que se utilizaron para el desarrollo de la aplicación

ENTORNO DE DESARROLLO

El aplicativo del **Sistema de Simulación de Trafico** tiene la finalidad de gestionar de manera eficiente la circulación de vehículos en una ciudad, priorizando vehículos especiales. Se recomienda que el siguiente manual sea manipulado únicamente por la persona que quiera administrar, editar o configurar el software para el **Sistema de Simulación de Trafico** para velar por la seguridad de los datos que se almacenan. A continuación se presenta la instalación, configuración y descripción de las herramientas utilizadas para el desarrollo de la aplicación:

Graphviz

- Descripción

Graphviz es un software de visualización de gráficos de código abierto. La visualización de gráficos es una forma de representar información estructural como diagramas de redes y gráficos abstractos. Tiene importantes aplicaciones en redes, bioinformática, ingeniería de software, diseño de base de datos y sitios web, aprendizaje automático y en interfaces visuales para otros dominios técnicos.

Los programas de diseño de Graphviz toman descripciones de gráficos en un lenguaje de texto simple y crean diagramas en formato útiles, como imágenes y SVG para páginas web; PDF Postscript para incluir en otros documentos; o visualizar en un navegador concreto, como opción

de colores, fuentes, diseños de nodos tabulares, estilos de líneas, hipervínculos y formas personalizadas.

- Instalación

Windows: Para la instalación de Graphviz desde su página oficial dirígete al siguiente enlace: https://graphviz.org/download/ en donde podrás descargar el ejecutable para Windows y así hacer su instalación.

Apache NetBeans

- Descripción

NetBeans es un entorno de desarrollo integrado (IDE) para Java. NetBeans permite desarrollar aplicaciones a partir de un conjunto de componentes de software modulares llamados módulos. NetBeans se ejecuta en Windows, macOS, Linux y Solaris. Además del desarrollo en Java, cuenta con extensiones para otros lenguajes como PHP, C, C++, HTML5 y JavaScript. Las aplicaciones basadas en NetBeans, incluido NetBeans IDE, pueden ser ampliadas por desarrolladores externos.

NetBeans IDE es un entorno de desarrollo integrado de código abierto. NetBeans IDE admite el desarrollo de todos los tipos de aplicaciones Java (Java SE (incluido JavaFX), Java ME, web, EJB y aplicaciones móviles) listas para usar. Entre otras características se encuentran un sistema de proyectos basado en Ant, soporte Maven, refactorizaciones, control de versiones (compatible con CVS, Subversion, Git, Mercurial y Clearcase). La plataforma Apache NetBeans es un marco genérico para aplicaciones Swing. Proporciona la "plomería" que, antes, cada desarrollador tenía que escribir por sí mismo: guardar el estado, conectar acciones a elementos del menú, elementos de la barra de herramientas y atajos de teclado; gestión de ventanas, etc.

- Instalación

  Windows: Para la instalación de NetBeans desde su página oficial dirígete al siguiente enlace: https://netbeans.apache.org/front/main/download/ en donde podrás descargar el ejecutable de instalación para Windows.

Java

- Descripción

La plataforma Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo.

Oracle Java es el principal lenguaje de programación y plataforma de desarrollo. Reduce costos, disminuye los tiempos de desarrollo, fomenta la innovación y mejora los servicios de las aplicaciones. Java sigue siendo la plataforma de desarrollo que eligen las empresas y los desarrolladores.

Java Development Kit es un software que provee herramientas de desarrollo para la creación de programas en Java. Puede instalarse en una computadora local o en una unidad de red. En la unidad de red se pueden tener las herramientas distribuidas en varias computadoras y trabajar como una sola aplicación.

- Instalación

  Windows: Para la instalación de Java JDK desde su página oficial dirígete al siguiente enlace: https://www.oracle.com/java/technologies/downloads/ en donde podrás descargar el ejecutable de instalación para Windows.

COMPLEJIDAD ALGORITMICA DE CADA ALGORITMO IMPLEMENTADO EN EL PROYECTO

A continuación se presenta la estructuración del proyecto junto con el análisis algorítmico de cada algoritmo implementado en el proyecto:

- Controllers
  - IntersectionController

```
private void buildTree() {
    NodeMatrix<Intersection> aux = this.city.getMatrix().getRoot();
    for (int i = 0; i < this.city.getMatrix().getDimensionY(); i++) {
        for (int j = 0; j < this.city.getMatrix().getDimensionX(); j++) {
            TreeNode<Intersection> newNode = new TreeNode<>(aux.getData(), aux.getData().getComplexity(), aux.getData().getId());
            this.avlTree.insert(newNode);
            aux = aux.getNext();
        }
        aux = this.city.getMatrix().getNode(i, 0);
        aux = aux.getBottom();
    }
}
```

O(1)
O(y)
O(x)
O(1)
O(log n)
O(1)

O(x + 0)
O(1)
TOTAL: O(x * y)

```
public void updateTree(LinkedList<Intersection> newIntersections) {
    for (int i = 0; i < newIntersections.getSize(); i++) {
        Intersection intersection = newIntersections.getElementAt(i).getData();
        TreeNode<Intersection> newNode = new TreeNode<>(intersection, intersection.getComplexity(), intersection.getId());
        this.avlTree.insert(newNode);
    }
}
```

O(n)
O(1)
O(1)
O(log n)
TOTAL: O(n)

  - ReportsController

```
public void vehicleRanking(LinkedList<Vehicle> vehicles) {
    System.out.println("---------- RRANKING DE VEHICULOS POR PRIORIDAD Y TIEMPO DE ESPERA ----------");
    this.sortingAlgorithms.insertionSort(vehicles);
    NodeList<Vehicle> node = vehicles.getHead();
    while (node != null) {
        System.out.println(node.getData());
        node = node.getNext();
    }
    System.out.println("");
}
```

O(1)
$O(n^2)$
O(1)
O(n)
O(1)
O(1)

O(1)
TOTAL; $O(n^2)$

```java
public void numberOfVehiclesCrossed(OrthogonalMatrix<Intersection> matrix) {
    System.out.println("--------------- CANTIDAD DE VEHICULOS QUE HAN CRUZADO POR INTERSECCION ---------------");
    NodeMatrix<Intersection> aux = matrix.getRoot();
    for (int i = 0; i < matrix.getDimensionY(); i++) {
        for (int j = 0; j < matrix.getDimensionX(); j++) {
            System.out.print("Interseccion: " + aux.getData().getId());
            System.out.println(", cruzaron/han estado: " + aux.getData().getNumberOfVehiclesCrossed() + " vehiculos");
            aux = aux.getNext();
        }
        aux = matrix.getNode(i, 0);
        aux = aux.getBottom();
    }
    System.out.println("");
}
```

O(1)
O(1)
O(y)
O(x)
O(1)
O(1)
O(1)

O(x + 0)
O(1)

O(1)
TOTAL: O(x * y)

```java
public void averageVehicleWaitingTime(LinkedList<Vehicle> vehicles) {
    System.out.println("---------- TIEMPO PROMEDIO DE ESPERA POR TIPO DE VEHICULO ----------");
    int[] time = new int[VehicleType.values().length];
    int[] numberOfVehicles = new int[VehicleType.values().length];
    for (int i = 0; i < vehicles.getSize(); i++) {
        Vehicle vehicle = vehicles.getElementAt(i).getData();
        time[vehicle.getVehicleType().ordinal()] += vehicle.getWaitingTime();
        numberOfVehicles[vehicle.getVehicleType().ordinal()]++;
    }
    for (int i = 0; i < VehicleType.values().length; i++) {
        if (numberOfVehicles[i] == 0) {
            System.out.println(VehicleType.values()[i].toString() + ": 0");
        }
        System.out.println(VehicleType.values()[i].toString() + ": " + time[i]/numberOfVehicles[i]);
    }
    System.out.println("");
}
```

O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)

O(1)
O(1)
O(1)

O(1)

O(1)
TOTAL: O(n)

```java
public void intersectionGraph(AVLTree<Intersection> avlTree) {
    avlTree.print(5);
    String dot = "digraph{\nrankdir = TB;\nlabel = \"Arbol de Intersecciones\";\nlabelloc = t;
    dot += this.traversePreOrden(avlTree.getRoot(), dot);
    dot += "\n}";
    Path outputDir = Paths.get("../Graficas");
    try {
        Files.createDirectories(outputDir);
        try (FileWriter file = new FileWriter("../Graficas/ReporteArbolAVL.txt")) {
            file.write(dot);
        }
        Process process = Runtime.getRuntime().exec(new String[] {
            "dot",
            "-Tpng",
            "../Graficas/ReporteArbolAVL.txt",
            "-o",
            "../Graficas/ReporteArbolAVL.png"
        });
        process.waitFor();
        System.out.println("\n>> Reporte Generado!!!");
    } catch (IOException | InterruptedException ex) {
        System.out.println(ex.getMessage());
    }
}
```

O(n)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(n)

```java
private String traversePreOrden(TreeNode<Intersection> node, String dot) {
    String dotAux = "";
    if (node != null) {
        dotAux += "\"" + node.getId()
        + "\"[label=<Interseccion" + node.getId()
        + "<br/>Complejidad: " + node.getSize()
        + "> fillcolor=\"#8080F0\"];\n";
        if (node.getLeft() != null) {
            dotAux += traversePreOrden(node.getLeft(), dotAux) + "\""
            + node.getId() + "\" -> \"" + node.getLeft().getId()
            + "\";\n";
        }
        if (node.getRight() != null) {
            dotAux += traversePreOrden(node.getRight(), dotAux) + "\""
            + node.getId() + "\" -> \"" + node.getRight().getId()
            + "\";\n";
        }
    }
    return dotAux;
}
```

O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(n)
O(1)
TOTAL: O(n)

```java
public void duplicatePlatesConflict(HashTable<Vehicle> hashTable) {
    System.out.println("--------------- PLACAS DUPLICADA/CONFLICTO ---------------");
    LinkedList<Vehicle> list = hashTable.getCollisions();
    if (list.isEmpty()) {
        System.out.println("NO HAY PLACAS DUPLICADAS/CONFLICTO EN EL SISTEMA");
        return;
    }
    for (int i = 0; i < list.getSize(); i++) {
        System.out.println(list.getElementAt(i).getData().toString());
    }
    System.out.println("");
}
```

O(1)
O(n)
O(1)
O(1)
O(1)

O(n)
O(1)

O(1)
TOTAL: O(n)

```java
public void latestRelevantEvents(Stack<Event> recordedEvents) {
    System.out.println("--------------- ULTIMOS 20 EVENTOS DEL SISTEMA ---------------");
    Stack<Event> auxStack = new Stack<>();
    for (int i = 0; i < this.NUMBER_OF_RELEVANT_EVENTS; i++) {
        Event event = recordedEvents.pop();
        if (event == null) {
            System.out.println("NO HAY MAS EVENTOS REGISTRADOS POR EL MOMENTO");
            break;
        }
        System.out.println(event.toString());
        auxStack.push(event);
    }
    while (true) {
        recordedEvents.push(auxStack.pop());
        if (auxStack.isEmpty()) break;
    }
    System.out.println("");
}
```

O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)

O(1)
O(1)

O(n)
O(1)
O(1)

O(1)
TOTAL: O(n)

- SystemController

```
private void enterVechicle() {
    VehicleType vehicleType = this.posters.incomeType();                                                                                                    O(1)
    String plate = this.posters.entrancePlate();                                                                                                            O(1)
    String origin = this.posters.incomeSource();                                                                                                            O(1)
    String destination = this.posters.destinationIncome();                                                                                                  O(1)
    int priority = this.posters.priorityEntry();                                                                                                            O(1)
    int waitingTime = this.posters.incomeWaitingTime();                                                                                                     O(1)
    Vehicle newVechicle = new Vehicle(vehicleType, plate, origin, destination, waitingTime, priority);                                                     O(1)
    this.hashTable.insert(plate, newVechicle);                                                                                                              O(1)
    int[] originCoordinates = this.utilities.convertCoordinate(origin);                                                                                     O(1)
    int[] destinationCoordinates = this.utilities.convertCoordinate(destination);                                                                           O(1)
    int dimensionX = originCoordinates[0] > destinationCoordinates[0] ? originCoordinates[0] : destinationCoordinates[0];                                   O(1)
    int dimensionY = originCoordinates[1] > destinationCoordinates[1] ? originCoordinates[1] : destinationCoordinates[1];                                   O(1)
    if (this.city.checkDimensions(dimensionX, dimensionY, this.newsIntersections)) {                                                                        O()
        this.intersectionsController.updateTree(this.newsIntersections);                                                                                    O(n)
        this.newsIntersections.clearList();                                                                                                             O(n)
    }
    NodeMatrix<Intersection> node = this.city.putVehicle(newVechicle, originCoordinates);                                                               O()
    this.avlTree.updateNode(node.getData().getComplexity() - 1, node.getData().getComplexity(), node.getData().getId());                                   O()
    if (this.city.getIncrasedCongestion() < node.getData().getComplexity()) this.city.setIncrasedCongestion(node.getData().getComplexity());               O(1)
    this.vehiclesInTheCity++;                                                                                                                               O(1)
    this.vehicles.addElementAt(newVechicle);                                                                                                                O(n)
    System.out.println("Vehiculo Registrado en el Sistema\n");                                                                                              O(1)
}                                                                                                                                                           TOTAL: O(n)
```

```
private void enterFile() {
    this.utilities.readTrafficFile(this.FILE_NAME, this.vehicles, this.coordinates);                                                                        O()
    this.vehiclesInTheCity += this.vehicles.getSize();                                                                                                      O(1)
    int dimensionX = 0, dimensionY = 0;                                                                                                                     O(1)
    for (int i = 0; i < this.coordinates.getSize(); i++) {                                                                                                  O(n)
        int[] coordinate = this.coordinates.getElementAt(i).getData();                                                                                      O(1)
        int currentX = coordinate[0];                                                                                                                       O(1)
        if (currentX > dimensionX) dimensionX = currentX;                                                                                                   O(1)

        int currentY = coordinate[1];                                                                                                                       O(1)
        if (currentY > dimensionY) dimensionY = currentY;                                                                                                   O(1)
    }
    if (this.city.checkDimensions(dimensionX, dimensionY, this.newsIntersections)) {                                                                        O()
        this.intersectionsController.updateTree(this.newsIntersections);                                                                                    O(n)
        this.newsIntersections.clearList();                                                                                                             O(n)
    }
    for (int i = 0; i < this.coordinates.getSize(); i+=2) {                                                                                                 O(n)
        Vehicle vehicle = this.vehicles.getElementAt(i/2).getData();                                                                                        O(1)
        NodeMatrix<Intersection> node = this.city.putVehicle(vehicle, this.coordinates.getElementAt(i).getData());                                         O()
        this.avlTree.updateNode(node.getData().getComplexity() - 1, node.getData().getComplexity(), node.getData().getId());                               O()
        if (this.city.getIncrasedCongestion() < node.getData().getComplexity()) this.city.setIncrasedCongestion(node.getData().getComplexity());           O(1)
        this.hashTable.insert(vehicle.getPlate(), vehicle);                                                                                                 O(1)
    }
}                                                                                                                                                           TOTAL: O(n)
```

```java
private void processIntersection(Intersection intersection) {
    //SE ACTUALIZA LA INTERSECCION ANTERIOR DEL VEHICULO
    Vehicle vehicle = intersection.reduceComplexity();
    //SE ACTUALIZA EL AVL
    this.avlTree.updateNode(intersection.getComplexity() + 1, intersection.getComplexity(), intersection.getId());
    //EL VEHICULO TIENE QUE MOVERSE
    vehicle.updateCurrentCoordinate();
    if (vehicle.getIsAtDestination()) {
        this.registerEvent(vehicle, intersection, null);
        this.vehiclesInTheCity--;
    } else {
        //SE ACTUALIZA LA INTERSECCION NUEVA DEL VEHICULO
        NodeMatrix<Intersection> newNode = this.city.putVehicle(vehicle, vehicle.getCurrentCoordinate());
        //SE ACTUALIZA EL AVL
        this.avlTree.updateNode(newNode.getData().getComplexity() - 1, newNode.getData().getComplexity(), newNode.getData().getId());
        this.registerEvent(vehicle, intersection, newNode.getData());
    }
    if (intersection.isCheckpoint()) {
        System.out.println("LA INTERSECCION " + intersection.getId() + " ES UN PUNTO DE CONTROL");
        int option = this.posters.menuViwReports();
        if (option == 1) this.generateReports();
    }
}
```

O(1)
O()
O(1)
O()
O()
O(1)
O(1)
O()
O()
O()
O(1)
O(1)
O(1)
O()
TOTAL: O(1)

```java
public void execute() {
    int option = this.posters.mainMenu();
    switch (option) {
        case 1 -> this.start();
        case 2 -> System.out.println("---------- FIN DE LA APLICACION ----------");
    }
}
```

O(1)
O(1)
O()
O(1)
TOTAL: O(1)

```java
private void start() {
    this.posters.initialConfiguration();
    this.enterFile();
    this.city.printCity();
    System.out.println("---------- INICIO DE LA SIMULACION DE TRAFICO ----------");
    while (this.vehiclesInTheCity > 0) {
        int option2 = this.posters.simulationType();
        switch (option2) {
            case 1 -> this.manualSimulation();
            case 2 -> this.automaticSimulation();
        }
        int option3 = this.posters.enterVehicleMenu();
        if (option3 == 1) this.enterVechicle();
    }
    System.out.println("---------- SIMULACION TERMINADA ----------");
    int option4 = this.posters.menuViwReports();
    switch (option4) {
        case 1 -> this.generateReports();
        case 2 -> System.out.println("---------- CERRANDO SISTEMA ----------");
    }
}
```

O(1)
O(n)
O(x * y)
O(1)
O(n)
O(1)
O(1)
O()
O()

O(1)
O(n)

O(1)
O(1)
O(1)
O()
O(1)
TOTAL: O(x * y)

```java
private void manualSimulation() {
    System.out.println("---------- SIMULACION DE TRAFICO MANUAL ----------");
    System.out.println("-------- INTERSECCIONES MAS CONGESTIONADAS --------");
    boolean thereAreIntersections = false;
    int congestion = this.city.getIncrasedCongestion();
    while (!thereAreIntersections && congestion > 0) {
        LinkedList<TreeNode> mostCongestedIntersections = this.avlTree.search(congestion);
        for (int i = 0; i < mostCongestedIntersections.getSize(); i++) {
            Intersection intersection = (Intersection) mostCongestedIntersections.getElementAt(i).getData().getData();
            if (!intersection.getVehiclesWaiting().isEmpty()) {
                intersection.printIntersection();
                thereAreIntersections = true;
            }
        }
        congestion--;
    }
    System.out.println("----- QUE INTERSECCION QUIERE LIBERAR? -----");
    System.out.print("Ingrese su opcion aqui: ");
    String coordinate = this.scanner.nextLine().toUpperCase();
    //BUSCAR LA INTERSECCION DENTRO DE LA MATRIZ
    int[] coordinateNumber = this.utilities.convertCoordinate(coordinate);
    NodeMatrix<Intersection> node = this.city.getMatrix().getNode(coordinateNumber[0] - 1, coordinateNumber[1] - 1);
    this.processIntersection(node.getData());
}
```

O(1)
O(1)
O(1)
O(1)
O(n)
O(log n)
O(n)
O(1)
O(1)
O(1)
O(1)

O(1)

O(1)
O(1)
O(1)

O()
O(x + y)
O(1)
TOTAL: O(x + y)

```java
private void automaticSimulation() {
    System.out.println("--------------- SIMULACION DE TRAFICO AUTOMATICA ---------------");
    System.out.println("--- CUANTO TIEMPO/PASOS QUIERE AVANZAR EN LA SIMULACION? ---");
    System.out.print("Ingrese su opcion aqui: ");
    int cycles = Integer.parseInt(scanner.nextLine());
    for (int i = 0; i < cycles; i++) {
        if (this.vehiclesInTheCity <= 0) break;
        boolean thereAreIntersections = false;
        int congestion = this.city.getIncrasedCongestion();
        while (!thereAreIntersections && congestion > 0) {
            LinkedList<TreeNode> mostCongestedIntersections = this.avlTree.search(congestion);
            for (int j = 0; j < mostCongestedIntersections.getSize(); j++) {
                Intersection intersection = (Intersection) mostCongestedIntersections.getElementAt(j).getData().getData();
                if (!intersection.getVehiclesWaiting().isEmpty()) {
                    this.processIntersection(intersection);
                    thereAreIntersections = true;
                    break;
                }
            }
            congestion--;
        }
    }
}
```

```java
public void registerEvent(Vehicle vehicle, Intersection oldIntersection, Intersection newIntersection) {
    String description;
    if (newIntersection == null) {
        description = "EL VEHICULO " + vehicle.getPlate() + " LLEGO A SU DESTINO";
    } else {
        description = "EL VEHICULO " + vehicle.getPlate() + " CRUZO DEL PUNTO " + oldIntersection.getId() + " AL PUNTO " + newIntersection.getId();
    }
    System.out.println(description);
    System.out.println("");
    int priorityAttended = oldIntersection.getComplexity() + 1;
    for (int i : vehicle.getTotalPriority()) {
        priorityAttended += i;
    }
    Event newEvent = new Event(vehicle, priorityAttended, vehicle.getWaitingTime(), description);
    this.recordedEvents.push(newEvent);
}
```

```java
public void generateReports() {
    int option = this.posters.reportsMenu();
    switch (option) {
        case 1 -> this.reportsController.vehicleRanking(this.vehicles);
        case 2 -> this.reportsController.numberOfVehiclesCrossed(this.city.getMatrix());
        case 3 -> this.reportsController.averageVehicleWaitingTime(this.vehicles);
        case 4 -> this.reportsController.intersectionGraph(this.avlTree);
        case 5 -> this.reportsController.duplicatePlatesConflict(this.hashTable);
        case 6 -> this.reportsController.latestRelevantEvents(this.recordedEvents);
    }
}
```

- Models
  - City

```
public boolean checkDimensions(int rowDimensions, int columnDimensions, LinkedList<Intersection> newIntersections) {
    int incrementX = rowDimensions - this.matrix.getDimensionX();
    int incrementY = columnDimensions - this.matrix.getDimensionY();
    if (incrementX > 0 || incrementY > 0) {
        this.matrix.increaseMatrix(incrementX, incrementY, newIntersections);
        if (this.incrasedCongestion < this.matrix.getIncrasedCongestion()) this.incrasedCongestion = this.matrix.getIncrasedCongestion();
        this.updateCityTemplate();
        return true;
    }
    return false;
}
```

O(1)
O(1)
O(1)
O()
O(1)
O()
O(1)

O(1)
TOTAL: O(1)

```
public NodeMatrix<Intersection> putVehicle(Vehicle newVehicle, int[] origin) {
    NodeMatrix<Intersection> node = this.matrix.getNode(origin[0] - 1, origin[1] - 1);
    node.getData().increaseComplexity(newVehicle);
    return node;
}
```

O(x + y)
O(1)
O(1)
TOTAL: O(x + y)

```
private void updateCityTemplate() {
    NodeMatrix<Intersection> aux = this.matrix.getRoot();
    for (int i = 0; i < this.matrix.getDimensionY(); i++) {
        for (int j = 0; j < this.matrix.getDimensionX(); j++) {
            this.setStreetType(aux);
            aux = aux.getNext();
        }
        aux = this.matrix.getNode(i, 0);
        aux = aux.getBottom();
    }
}
```

O(1)
O(y)
O(x)
O()
O(1)

O(x + 0)
O(1)

TOTAL: O(x*y)

```java
public void printCity() {
    int height = StreetType.INTERSECCION.getLines().length;
    int maxWidth = 0;
    for (int i = 0; i < this.matrix.getDimensionX(); i++) {
        for (int j = 0; j < this.matrix.getDimensionY(); j++) {
            StreetType t = this.matrix.getNode(i, j).getStreetType();
            for (String line : t.getLines()) {
                maxWidth = Math.max(maxWidth, line.length());
            }
        }
    }
    int labelWidth = 2;
    System.out.print(" ".repeat(labelWidth));
    for (int i = 0; i < this.matrix.getDimensionY(); i++) {
        System.out.print(String.format("%-" + maxWidth + "s", i + 1));
    }
    System.out.println();
    for (int i = 0; i < this.matrix.getDimensionX(); i++) {
        char rowChar = (char)('A' + i);
        for (int line = 0; line < height; line++) {
            if (line == 1) {
                System.out.print(String.format("%-" + labelWidth + "s", rowChar));
            } else {
                System.out.print(" ".repeat(labelWidth));
            }
            for (int j = 0; j < this.matrix.getDimensionY(); j++) {
                StreetType type = this.matrix.getNode(i, j).getStreetType();
                String[] lines = type.getLines();
                String fragment = (line < lines.length ? lines[line] : "");
                System.out.print(String.format("%-" + maxWidth + "s", fragment));
            }
            System.out.println();
        }
    }
    System.out.println("");
}
```

- Event
- Intersection

```
public void incraseComplexity(Vehicle newVehicle) {
    this.vehiclesWaiting.enqueue(newVehicle, newVehicle.getTotalPriority());
    this.complexity++;
    this.numberOfVehiclesCrossed++;
}
```

O(1)
O(1)
O(1)
TOTAL: O(1)

```
public Vehicle reduceComplexity() {
    this.complexity--;
    return this.vehiclesWaiting.unqueue();
}
```

O(1)
O(1)
TOTAL: O(1)

```
public void printIntersection() {
    System.out.println("Interseccion: " + this.id);
    System.out.println("\tComplejidad: " + this.complexity);
    System.out.println("\tTipo de Interseccion: " + this.intersectionType.toString());
    System.out.println("\tVehiculos en Espera:");
    NodeQueue<Vehicle> vehicles = this.vehiclesWaiting.getHead();
    while (vehicles != null) {
        System.out.println("\t\t" + vehicles.getData().toString());
        vehicles = vehicles.getNext();
    }
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
TOTAL: O(n)

```
private void initializeCheckpoint() {
    int random = (int) (Math.random() * 10);
    this.checkpoint = random < 2;
}
```

O(1)
O(1)
TOTAL: O(1)

- Vehicle

```
private void setDirectionMovement() {
    int[] originCoordinates = this.utilities.convertCoordinate(this.intersectionOrigin);
    int[] destinationCoordinates = this.utilities.convertCoordinate(this.destinationIntersection);
    this.direction[0] = destinationCoordinates[0] - originCoordinates[0];
    this.direction[1] = destinationCoordinates[1] - originCoordinates[1];
}
```

```
private void getFullPriority() {
    this.totalPriority[0] = this.vehicleType.getPriority();
    this.totalPriority[1] = this.urgencyLevel;
    this.totalPriority[2] = this.waitingTime;
    this.totalPriority[3] = this.direction[0] + this.direction[1];
}
```

```
public void updateCurrentCoordinate() {
    if (!isAtDestination) {
        if (this.direction[0] != 0) {
            if (this.direction[0] < 0) {
                this.direction[0]++;
                this.currentCoordinate[0]--;
            } else if (this.direction[0] > 0) {
                this.direction[0]--;
                this.currentCoordinate[0]++;
            }
        } else if (this.direction[1] != 0) {
            if (this.direction[1] < 0) {
                this.direction[1]++;
                this.currentCoordinate[1]--;
            } else if (this.direction[1] > 0) {
                this.direction[1]--;
                this.currentCoordinate[1]++;
            }
        }
    }
}
```

- VehicleType
- Structs
  - Hash
    - HashTable

```
public void insert(String key, T data) {
    int hash = this.hashFunction(key);
    this.map[hash].addElementAt(data);
}
```

O(n)
O(1)
TOTAL: O(n)

```
private int hashFunction(String key) {
    int hash = 0;
    for (char c : key.toCharArray()) {
        if (Character.isLetter(c)) {
            hash += c - 'A' + 1;
        } else if (Character.isDigit(c)) {
            hash += c - '0' + 1;
        } else {
            hash += c - '-' + 1;
        }
    }
    return hash % this.map.length;
}
```

O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)

O(1)
TOTAL: O(n)

```
public T get(String key) {
    int hash = this.hashFunction(key);
    return this.map[hash].getElementAt(key).getData();
}
```

O(n)
O(1)
TOTAL: O(n)

```
private boolean isOnTheList(LinkedList<Vehicle> list, String plate) {
    for (int i = 0; i < list.getSize(); i++) {
        Vehicle vehicle = list.getElementAt(i).getData();
        if (vehicle.getPlate().equals(plate)) return true;
    }
    return false;
}
```

O(n)
O(1)
O(1)

O(1)
TOTAL: O(n)

```java
public LinkedList<Vehicle> getCollisions() {
    LinkedList<Vehicle> list = new LinkedList<>();
    LinkedList<String> singlePlates = new LinkedList<>();
    for (LinkedList<T> linkedList : this.map) {
        for (int i = 0; i < linkedList.getSize(); i++) {
            Vehicle cuurent = (Vehicle) linkedList.getElementAt(i).getData();
            String currentPlate = cuurent.getPlate();
            if (isOnTheList(list, currentPlate)) continue;

            for (int j = i + 1; j < linkedList.getSize(); j++) {
                Vehicle compare = (Vehicle) linkedList.getElementAt(j).getData();
                if (currentPlate.equals(compare.getPlate())) {
                    if (!isOnTheList(list, currentPlate)) {
                        list.addElementAt(cuurent);
                        singlePlates.addElementAt(currentPlate);
                    }
                    list.addElementAt(compare);
                }
            }
        }
    }
    return list;
}
```

- List
  - LinkedList

```java
public NodeList<T> getElementAt(int index) {
    if (index < 0 || index >= this.size) return null;

    if (index == 0) return this.head;

    int counter = 0;
    NodeList<T> current = this.head;
    while (counter < index) {
        current = current.getNext();
        counter++;
    }
    return current;
}
```

```java
public NodeList<T> getElementAt(String key) {
    NodeList<T> current = this.head;
    while (current != null) {
        if (current.getKey().equals(key)) {
            return current;
        }
        current = current.getNext();
    }
    return null;
}
```

O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(n)

```java
public void addElementAt(T element) {
    NodeList<T> newNode = new NodeList<>(element);
    if (this.isEmpty()) {
        this.head = newNode;
        ++this.size;
        return;
    }
    NodeList<T> current = this.head;
    while (current.getNext() != null) current = current.getNext();

    current.setNext(newNode);
    ++this.size;
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
TOTAL: O(n)

```java
public T removeElementAt(int index) {
    NodeList<T> current = this.head;
    NodeList<T> prev = null;
    for (int i = 0; i < index; i++) {
        prev = current;
        current = current.getNext();
    }
    if (prev != null) {
        prev.setNext(current.getNext());
    } else {
        this.head = current.getNext();
    }
    current.setNext(null);
    --this.size;
    return current.getData();
}
```

O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(n)

```java
public T removeElements(T element) {
    NodeList<T> current = this.head;
    NodeList<T> prev = null;
    while (current != null && current.getData() != element) {
        prev = current;
        current = current.getNext();
    }
    if (current == null) return null;

    if (prev != null) {
        prev.setNext(current.getNext());
    } else {
        this.head = current.getNext();
    }
    current.setNext(null);
    --this.size;
    return current.getData();
}
```

O(1)
O(1)
O(n)
O(1)
O(1)

O(1)

O(1)
O(1)
O(1)
O(1)

O(1)
O(1)
O(1)
TOTAL: O(n)

```java
public boolean isEmpty() {
    return this.size == 0 && this.head == null;
}
```

O(1)
TOTAL: O(1)

```java
public void clearList() {
    if (!this.isEmpty()) {
        NodeList<T> current = this.head;
        while (current != null) {
            NodeList<T> next = current.getNext();
            current.setNext(null);
            current = next;
        }
        this.head = null;
        this.size = 0;
    }
}
```

O(1)
O(1)
O(n)
O(1)
O(1)
O(1)

O(1)
O(1)
TOTAL: O(n)

- NodeList
- Matrix
  - IntersectionComplexityType
  - NodeMatrix
  - OrthogonalMatrix

```
private void createOrthogonalMatrix() {
    NodeMatrix<T> aux = this.root;
    for (int i = 0; i < this.dimensionX - 1; i++) {
        this.connectNodes(aux, i);
        aux = this.getNode(i, 0);
        String coordinateLetter = this.utilities.convertCoordinate(i + 1);
        String coordinateNumber = "1";
        Intersection newIntersection = new Intersection(coordinateLetter + coordinateNumber);
        if (this.incrasedCongestion < newIntersection.getComplexity()) this.incrasedCongestion = newIntersection.getComplexity();
        NodeMatrix newNode = new NodeMatrix<>(newIntersection, i + 1, 0);
        aux.setBottom(newNode);
        newNode.setTop(aux);
        aux = newNode;
        this.connectNodes(aux, i + 1);
    }
}
```

```
private void connectNodes(NodeMatrix<T> aux, int i) {
    for (int j = 0; j < this.dimensionY - 1; j++) {
        String coordinateLetter = this.utilities.convertCoordinate(i);
        String coordinateNumber = String.valueOf(j + 2);
        Intersection newIntersection = new Intersection(coordinateLetter + coordinateNumber);
        if (this.incrasedCongestion < newIntersection.getComplexity()) this.incrasedCongestion = newIntersection.getComplexity();
        NodeMatrix newNode = new NodeMatrix<>(newIntersection, i, j + 1);
        aux.setNext(newNode);
        newNode.setPrev(aux);
        if (i > 0) {
            newNode.setTop(aux.getTop().getNext());
            aux.getTop().getNext().setBottom(newNode);
        }
        aux = newNode;
    }
}
```

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

O(1)
TOTAL: O(1)

```java
public NodeMatrix<T> getNode(int x, int y) {
    NodeMatrix<T> aux = this.root;
    for (int i = 0; i < x; ++i) aux = aux.getBottom();

    for (int i = 0; i < y; ++i) aux = aux.getNext();

    return aux;
}
```

O(1)
O(x)

O(y)

O(1)
TOTAL: O(x + y)

```java
public void incraseMatrix(int incrementX, int incrementY, LinkedList<Intersection> newIntersections) {
    NodeMatrix<T> aux = this.getNode(0, this.dimensionY - 1);
    for (int i = 0; i < this.dimensionX - 1; i++) {
        this.incrementColumns(aux, i, incrementY, newIntersections);
        aux = this.getNode(i + 1, this.dimensionY - 1);
        this.incrementColumns(aux, i + 1, incrementY, newIntersections);
    }
    if (incrementX > 0) {
        for (int i = this.dimensionX; i < this.dimensionX + incrementX; i++) {
            aux = this.getNode(i - 1, 0);
            String coordinateLetter = this.utilities.convertCoordinate(i);
            String coordinateNumber = "1";
            Intersection newIntersection = new Intersection(coordinateLetter + coordinateNumber);
            if (this.incrasedCongestion < newIntersection.getComplexity()) this.incrasedCongestion = newIntersection.getComplexity();
            newIntersections.addElementAt(newIntersection);
            NodeMatrix newNode = new NodeMatrix(newIntersection, i, 0);
            newNode.setTop(aux);
            aux.setBottom(newNode);
            aux = newNode;
            this.incrementRows(aux, i, incrementY, newIntersections);
        }
        this.dimensionX += incrementX;
    }
    if (incrementY > 0) this.dimensionY += incrementY;
}
```

```java
private void incrementColumns(NodeMatrix<T> aux, int i, int incrementY, LinkedList<Intersection> newIntersections) {
    for (int j = 0; j < incrementY; j++) {
        String coordinateLetter = this.utilities.convertCoordinate(i);
        String coordinateNumber = String.valueOf(aux.getY() + 2);
        Intersection newIntersection = new Intersection(coordinateLetter + coordinateNumber);
        if (this.incrasedCongestion < newIntersection.getComplexity()) this.incrasedCongestion = newIntersection.getComplexity();
        newIntersections.addElementAt(newIntersection);
        NodeMatrix newNode = new NodeMatrix(newIntersection, i, aux.getY() + 1);
        aux.setNext(newNode);
        newNode.setPrev(aux);
        if (i > 0) {
            newNode.setTop(aux.getTop().getNext());
            aux.getTop().getNext().setBottom(newNode);
        }
        aux = newNode;
    }
}
```

```java
private void incrementRows(NodeMatrix<T> aux, int i, int incrementY, LinkedList<Intersection> newIntersections) {
    for (int j = 0; j < this.dimensionY + incrementY - 1; j++) {
        String coordinateLetter = this.utilities.convertCoordinate(i);
        String coordinateNumber = String.valueOf(j + 2);
        Intersection newIntersection = new Intersection(coordinateLetter + coordinateNumber);
        if (this.incrasedCongestion < newIntersection.getComplexity()) this.incrasedCongestion = newIntersection.getComplexity();
        newIntersections.addElementAt(newIntersection);
        NodeMatrix newNode = new NodeMatrix(newIntersection, i, j + 1);
        aux.setNext(newNode);
        newNode.setPrev(aux);
        newNode.setTop(aux.getTop().getNext());
        aux.getTop().getNext().setBottom(newNode);
        aux = newNode;
    }
}
```

- StreetType

- Queue
  - NodeQueue
  - PriorityQueu

```
public void enqueue(T data, int[] priority) {
    NodeQueue<T> newNode = new NodeQueue<>(data, priority);
    if (this.head == null) {
        this.head = newNode;
        return;
    }
    NodeQueue<T> current = this.head;
    NodeQueue<T> prev = null;
    while (current != null) {
        if (this.comparePriority(priority, current.getPriority()) > 0) {
            if (prev == null) { // Insertar al inicio
                newNode.setNext(this.head);
                this.head = newNode;
            } else { // Insertar entre anterior y actual
                prev.setNext(newNode);
                newNode.setNext(current);
            }
            return;
        }
        // Avanzar
        prev = current;
        current = current.getNext();
    }
    // Si no se insertó, agregar al final
    if (prev != null) prev.setNext(newNode);
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(n)

```
private int comparePriority(int[] newPriority, int[] currentPriority) {
    for (int i = 0; i < this.FIELDS_EVALUATE_PRIORITY; i++) {
        if (newPriority[i] != currentPriority[i]) {
            return Integer.compare(newPriority[i], currentPriority[i]);
        }
    }
    return 0;
}
```

O(n)
O(1)
O(1)
O(1)
TOTAL: O(n)

```
public T unqueue() {
    if (this.head == null) return null;

    T data = this.head.getData();
    this.head = this.head.getNext();
    return data;
}
```

O(1)

O(1)
O(1)
O(1)
TOTAL: O(1)

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

O(1)
TOTAL: O(1)

- Queue

```
public void enqueue(T data) {
    NodeQueue<T> newNode = new NodeQueue<>(data);
    if (this.end == null) {
        this.start = this.end = newNode;
        return;
    }
    this.end.setNext(newNode);
    this.end = newNode;
}
```

O(1)
O(1)
O(1)
O(1)

O(1)
O(1)
TOTAL: O(1)

```
public T unqueue() {
    if (this.start == null) return null;

    T data = this.start.getData();
    this.start = this.start.getNext();
    if (this.start == null) this.end = null;

    return data;
}
```

O(1)

O(1)
O(1)
O(1)

O(1)
TOTAL: O(1)

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

O(1)
TOTAL: O(1)

- Stack
  - NodeStack
  - Stack

```
public void push(T data) {
    NodeStack<T> newNode = new NodeStack<>(data);
    newNode.setNext(this.root);
    this.root = newNode;
}
```

O(1)
O(1)
O(1)
TOTAL: O(1)

```
public T pop() {
    if (this.root == null) return null;

    T data = this.root.getData();
    this.root = this.root.getNext();
    return data;
}
```

O(1)

O(1)
O(1)
O(1)
TOTAL: O(1)

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

O(1)
TOTAL: O(1)

- Tree
  - AVLTree

```
public void insert(TreeNode newNode) {
    this.root = this.insert(newNode, this.root);
}
```

O(1)
TOTAL: O(1)

```java
private TreeNode insert(TreeNode node, TreeNode root) {
    if (root == null) {
        node.setBalanceFactor(this.obtainBalanceFactor(node));
        //root = node;
        return node;
    }
    if (node.getSize() < root.getSize()) {
        root.setLeft(this.insert(node, root.getLeft()));
    } else {
        root.setRight(this.insert(node, root.getRight()));
    }
    return this.balanceTree(root);
}
```

```java
private int obtainBalanceFactor(TreeNode node) {
    int leftHeight =  this.getMaximumHeight(node.getLeft());
    int rightHeight =  this.getMaximumHeight(node.getRight());
    return rightHeight - leftHeight;
}
```

```java
private int getMaximumHeight(TreeNode node) {
    if (node == null) return 0;

    int leftHeight =  this.getMaximumHeight(node.getLeft());
    int rightHeight =  this.getMaximumHeight(node.getRight());
    return 1 + (leftHeight > rightHeight ? leftHeight: rightHeight);
}
```

```java
private TreeNode balanceTree(TreeNode root) {
    root.setBalanceFactor(this.obtainBalanceFactor(root));
    if (root.getBalanceFactor() < -1) {
        if (root.getLeft().getBalanceFactor() > 0) {
            return this.doubleRightRotation(root);
        }
        return this.rightRotation(root);
    }
    if (root.getBalanceFactor() > 1) {
        if (root.getRight().getBalanceFactor() < 0) {
            return this.doubleLeftRotation(root);
        }
        return this.leftRotation(root);
    }
    return root;
}
```

```java
private TreeNode doubleRightRotation(TreeNode node) {
    node.setLeft(this.leftRotation(node.getLeft()));
    return this.rightRotation(node);
}
```

O(1)
O(1)
TOTAL: O(1)

```java
private TreeNode rightRotation(TreeNode node) {
    TreeNode aux = node.getLeft();
    node.setLeft(aux.getRight());
    aux.setRight(node);
    //node = aux;
    node.setBalanceFactor(this.obtainBalanceFactor(node));
    aux.setBalanceFactor(this.obtainBalanceFactor(aux));
    return aux;
    //node.getRight().setBalanceFactor(this.obtainBalanceFactor(node.getRight()));
    //if (node.getLeft() == null) return null;

    //node.getLeft().setBalanceFactor(this.obtainBalanceFactor(node.getLeft()));
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(1)

```java
private TreeNode doubleLeftRotation(TreeNode node) {
    node.setRight(this.rightRotation(node.getRight()));
    return this.leftRotation(node);
}
```

O(1)
O(1)
TOTAL: O(1)

```java
private TreeNode leftRotation(TreeNode node) {
    TreeNode aux = node.getRight();
    node.setRight(aux.getLeft());
    aux.setLeft(node);
    //node = aux;
    node.setBalanceFactor(this.obtainBalanceFactor(node));
    aux.setBalanceFactor(this.obtainBalanceFactor(aux));
    return aux;
    //node.getLeft().setBalanceFactor(this.obtainBalanceFactor(node.getLeft()));
    //if (node.getRight() == null) return null;

    //node.getRight().setBalanceFactor(this.obtainBalanceFactor(node.getRight()));
}
```

O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: O(1)

```java
public void delete(int size, String id) {
    this.root = this.delete(size, this.root, id);
}
```

O(log n)
TOTAL: O(log n)

```java
private TreeNode delete(int size, TreeNode root, String id) {
    if (root == null) {
        //System.out.println("Nodo con valor: " + size + " e ID: " + id + " no se encontro para eliminar");
        return null;
    }
    if (size < root.getSize()) {
        root.setLeft(this.delete(size, root.getLeft(), id));
    } else if (size > root.getSize()) {
        root.setRight(this.delete(size, root.getRight(), id));
    } else {
        if (id.equals(root.getId())) {
            if (this.isLeaf(root)) return null;
            if (root.getLeft() == null) return root.getRight();
            if (root.getRight() == null) return root.getLeft();

            TreeNode rightNode = this.getMajorNode(root.getLeft());
            root.setData(rightNode.getData());
            root.setSize(rightNode.getSize());
            root.setId(rightNode.getId());
            root.setLeft(this.delete(rightNode.getSize(), root.getLeft(), rightNode.getId()));
        } else {
            //System.out.println("Nodo con valor: " + size + "coincide, pero ID: " + id + " no. Buscando/Eliminando derecha...");
            root.setRight(this.delete(size, root.getRight(), id));
            //System.out.println("Nodo con valor: " + size + "coincide, pero ID: " + id + " no. Buscando/Eliminando izquierda...");
            root.setLeft(this.delete(size, root.getLeft(), id));
        }
        //size = root.getSize();
    }
    return this.balanceTree(root);
}
```

```java
public boolean isLeaf(TreeNode node) {
    return node.getLeft() == null && node.getRight() == null;
}
```

O(1)
TOTAL: O(1)

```java
public void updateNode(int size, int newSize, String id) {
    LinkedList<TreeNode> nodesFound = this.search(size);
    TreeNode node = null;
    TreeNode aux;
    for (int i = 0; i < nodesFound.getSize(); i++) {
        aux = nodesFound.getElementAt(i).getData();
        if (aux.getId().equals(id)) {
            node = aux;
            break;
        }
    }
    if (node != null) {
        TreeNode updatedNode = new TreeNode(node.getData());
        updatedNode.setSize(newSize);
        updatedNode.setId(id);
        this.delete(size, id);
        this.insert(updatedNode);
    }
}
```

```java
public TreeNode getMajorNode(TreeNode node) {
    if (node.getRight() == null) {
        return node;
    }
    return getMajorNode(node.getRight());
}
```

```java
public LinkedList<TreeNode> search(int size) {
    LinkedList<TreeNode> nodesFound = new LinkedList<>();
    this.search(size, root, nodesFound);
    return nodesFound;
}
```

O(1)
O(log n)
O(1)

TOTAL: O(log n)

```java
private void search(int size, TreeNode node, LinkedList<TreeNode> list) {
    if (node != null) {
        if (size < node.getSize()) {
            this.search(size, node.getLeft(), list);
        } else if (size > node.getSize()) {
            this.search(size, node.getRight(), list);
        } else {
            list.addElementAt(node);
            this.search(size, node.getLeft(), list);
            this.search(size, node.getRight(), list);
        }
    }
}
```

O(1)
O(1)
O(log n)
O(1)
O(log n)
O(1)
O(1)
O(1)
O(1)

TOTAL: O(log n)

```java
public void print(int indentIncrement) {
    System.out.println("\n-------------------- Arbol AVL Visual --------------------");
    if (this.root == null) {
        System.out.println("(Árbol vacío)");
    } else {
        print(this.root, 0, indentIncrement);
    }
    System.out.println("\n");
}
```

O(1)
O(1)
O(1)
O(1)
O(n)

O(1)
TOTAL: O(n)

```java
private void print(TreeNode node, int space, int level) {
    if (node == null) return;

    int newSpace = space + level;
    print(node.getRight(), newSpace, level);
    System.out.println();
    for (int i = 0; i < space; i++) System.out.print(" ");

    System.out.print("" + node.getData() + node.getBalanceFactor());
    print(node.getLeft(), newSpace, level);
}
```

O(1)

O(1)
O(n)
O(1)
O(1)

O(1)
O(n)
TOTAL: O(n)

- TreeNode

- Utils
  - Posters
  - SortingAlgorithms

```java
public void insertionSort(LinkedList<Vehicle> list) {
    NodeList<Vehicle> sortedHead = null;
    NodeList<Vehicle> current = list.getHead();
    while (current != null) {
        NodeList<Vehicle> nextNode = current.getNext();
        if (sortedHead == null
                || compare(current.getData(), sortedHead.getData()) <= 0) {
            //Insertar al frente
            current.setNext(sortedHead);
            sortedHead = current;
        } else {
            //Encontrar posicion
            NodeList<Vehicle> aux = sortedHead;
            while (aux.getNext() != null
                    && compare(aux.getNext().getData(), current.getData()) < 0) {
                aux = aux.getNext();
            }
            //Insertar entre aux y aux.next
            current.setNext(aux.getNext());
            aux.setNext(current);
        }
        current = nextNode;
    }
    //Acutalizar cabeza de la lista original
    list.setHead(sortedHead);
}
```

O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
O(n)
O(1)
O(1)
O(1)
O(1)
O(1)
TOTAL: $O(n^2)$

```java
private int compare(Vehicle currentVehicle, Vehicle nextVehicle) {
    int result = Integer.compare(nextVehicle.getUrgencyLevel(), currentVehicle.getUrgencyLevel());
    if (result != 0) return result;

    return Integer.compare(nextVehicle.getWaitingTime(), currentVehicle.getWaitingTime());
}
```

O(1)
O(1)
O(1)
TOTAL: O(1)

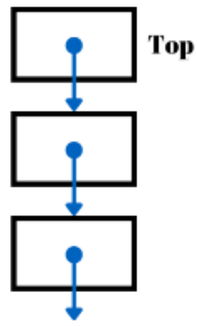
  - Utilities

DESCRIPCION DE LAS ESTRUCTURAS DE DATOS

En el software se usaron las Estructuras de Datos tales como: Listas Enlazadas (Linked List), Pilas (Stack), Cola (Queue) y Cola de Prioridad tipo Lista Ordenada (Priority Queue), Árbol AVL (AVL Tree) y Matriz Ortogonal (Orthogonal Matrix). A continuación se incluye una visualización de dichas estructuras, así como una breve explicación de su funcionamiento para entender los algoritmos de implementación.

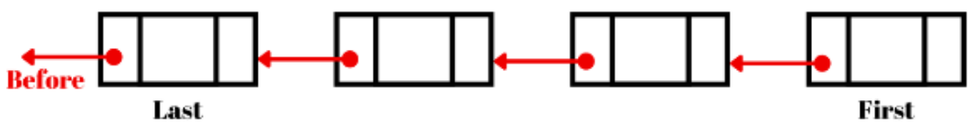
Pila (Stack)

En una Pila se ingresan elementos hasta el tope y solo pueden sacarse de ahí, sigue la regla **"El último que entra es el primero en salir – Last Int First Out - LIFO"**



Cola (Queue)

En una Cola se ingresan elementos hasta el top y solo puede sacarse de ahí el primer elemento, sigue la regla de **"El Primero que entra es el primero en salir – First Int First Out - FIFO"**

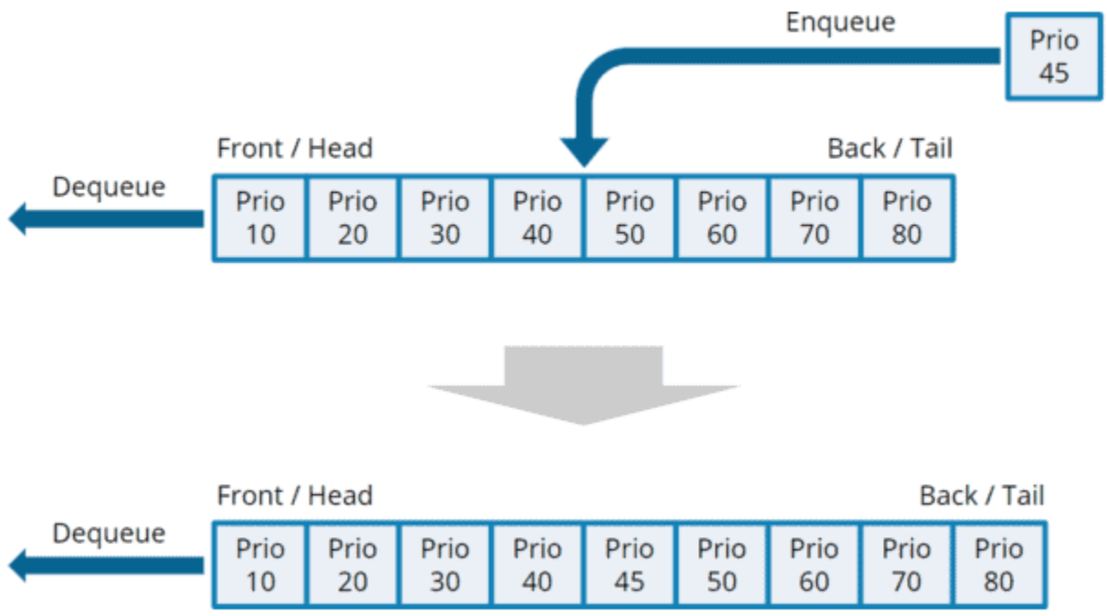


Cola de Prioridad (Priority Queue)

- Descripción

Una cola de prioridad es una estructura de datos que, como una cola normal, guarda elementos, pero con la característica adicional de que cada elemento tiene un aprioridad asignada. Los elementos se procesan de la cola según su prioridad, los de mayor prioridad se procesa antes que los de menor prioridad.

- Conceptos clave
  - Prioridad: Cada elemento tiene una prioridad asociada que determina su orden de procesamiento
  - Orden de procesamiento: Los elementos con mayor prioridad se procesan antes que los de menor prioridad
  - Implementación común: Los montículos (especialmente los montículos binarios) son una implementación común y eficiente para las colas de prioridad
- Operaciones básicas
  - Insertar: Agregar un nuevo elemento con su prioridad asociada
  - Eliminar: Eliminar el elemento con la mayor prioridad (el que se encuentra en la parte superior de la cola)
  - Ver elementos de mayor prioridad: Obtener el elemento con mayor prioridad sin eliminarlo



Lista Enlazada Simple (Linked List)

Esta Lista tiene un solo enlace hacia adelante, tal cual se muestra en la imagen:

Es una estructura de datos lineal formada por una secuencia de nodos. Cada nodo contiene al menos dos partes fundamentales: un **dato** y un **enlace o referencia** al siguiente nodo de la lista.

Árbol AVL

- Descripción

Un árbol AVL es un tipo especial de árbol binario, fue el primer árbol de búsqueda auto-balanceado que se ideo. Los arboles AVL están siempre equilibrados de tal modo que para todos los nodos, la altura de la rama izquierda no difiere en más de una unidad de la altura de la rama derecha o viceversa. Gracias a esta forma de equilibrio, la complejidad de una búsqueda en uno de estos árboles se mantiene siempre en orden de complejidad O(log n). El factor de equilibrio puede ser almacenado directamente en cada nodo o ser computado a partir de las alturas de los subárboles.

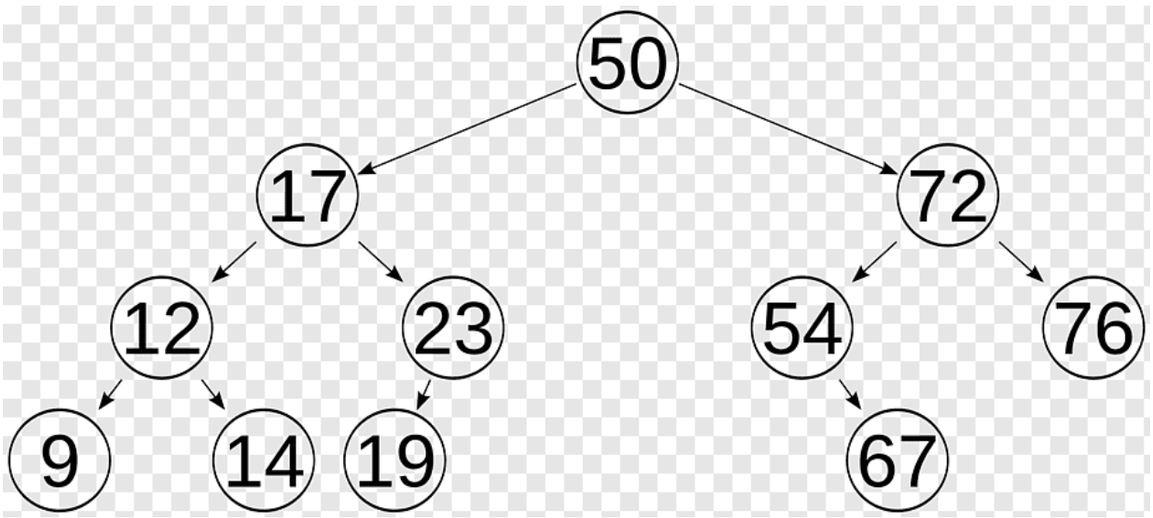
Para conseguir esta propiedad de equilibrio, la inserción y eliminación de los nodos se ha de realizar de una forma especial. Si al realizar una operación de inserción o eliminación se rompe la condición de equilibrio, hay que realizar una serie de rotaciones de los nodos.

- Operaciones Básicas
  - Insertar: La inserción en un árbol AVL puede ser realizada insertando el valor dado en el árbol como si fuera un árbol de búsqueda binario desequilibrado y después retrocediendo hacia la raíz, rotando sobre cualquier nodo que pueda haberse desequilibrado durante la inserción. Para la inserción se busca hasta encontrar la posición de la inserción o modificación, luego se inserta el nuevo nodo con factor de equilibrio, verificando el equilibrio de los nodos y re-equilibrando si es necesario. Debido a que las rotaciones son una operación que tienen complejidad constante y a que la altura está limitada a O(log n), el tiempo de ejecución para la inserción es del orden O(log n).
  - Eliminar: El procedimiento de eliminación es el mismo que en un árbol binario de búsqueda. La diferencia se encuentra en el proceso de reequilibrio posterior. Una extracción trae consigo una disminución de la altura de la rama donde se extrajo y

tendrá como efecto un cambio en el factor de equilibrio del nodo padre de la rama en cuestión, pudiendo necesitarse una rotación simple o rotación doble.

- Rotaciones: El reequilibrio se produce de abajo hacia arriba sobre los nodos en los que se produce el desequilibrio. Pueden darse dos casos: rotación simple o rotación doble; a su vez ambos casos pueden ser hacia la derecha o hacia la izquierda.
    - Rotación Simple a la Derecha: Esta rotación se usara cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su Factor de Equilibrio sea de -2. Y además, la raíz del subárbol izquierdo tenga un Factor de Equilibrio de -1 o 0, es decir, que este cargado a la izquierda o equilibrado.
    - Rotación Simple a la Izquierda: Se trata de caso simétrico de anterior. Esta rotación se usara cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su Factor de equilibrio sea de 2. Y además, la raíz del subárbol derecho tenga un Factor de Equilibrio de 1 o 0, es decir, que este cargado a la derecha o este equilibrado.
    - Rotación Doble a la Derecha: Esta rotación se usara cuando el subárbol izquierdo de un nodo sea 2 unidades más alto que el derecho, es decir, cuando su Factor de Equilibrio sea de -2. Y además, la raíz del subárbol izquierdo tenga Factor de Equilibrio de 1, es decir, que este cargado a la derecha.
    - Rotación Doble a la Izquierda: Esa rotación se usara cuando el subárbol derecho de un nodo sea 2 unidades más alto que el izquierdo, es decir, cuando su Factor de Equilibrio sea de 2: Y además, la raíz del subárbol derecho tenga un Factor de Equilibrio de -1, es decir, que este cargado a la izquierda: Se trata del caso simétrico del anterior.

Búsqueda: El procedimiento es casi similar a un Árbol de Búsqueda Binaria. Sin embargo por ser un árbol ordenado y equilibrado no se tomara mucho tiempo para hallar el elemento deseado.



Matriz Ortogonal (Orthogonal Matrix)

Es una estructura de datos que representa una matriz dispersa (con muchos ceros o valores nulos), utilizando listas enlazadas para almacenar solo los elementos no nulos. Se usa para ahorrar espacio en matrices muy grandes con pocos valores significativos.

- Cada fila y columna se representa como una lista enlazada
- Los nodos contiene como mínimo:
    - Valor
    - Referencia a la fila siguiente y anterior

- Referencia a la columna siguiente y anterior