Manual Tecnico

Herramientas de desarrollo

Las herramientas (IDEs, lenguaje de programación, librerías, etc) utilizadas para el desarrollo de la aplicación (TravelMap-GT) se listan a continuación:

- Lenguaje:
    - Java:

        Versión 17.0.8
- Librerías utilizadas:
    - GraphViz

        Versión 0.18.1
- Entorno de desarrollo (IDE);
    - IntelliJ IDEA:

        Versión 2024.1 (Ultimate edition)
- Sistema operativo de desarrollo
    - Windows 11 – home

        Versión 10.0.22621

Conceptos aplicados

Árbol B

Los árboles B, también conocidos como árboles B-search, son una estructura de datos arborescente diseñada para almacenar y recuperar información de manera eficiente, especialmente cuando se trata de grandes conjuntos de datos. Su principal característica es que permiten almacenar m claves en cada nodo, en lugar de solo dos como en los árboles binarios de búsqueda tradicionales. Esta característica, junto con otras propiedades, los convierte en una opción ideal para manejar grandes volúmenes de datos con operaciones de búsqueda, inserción y eliminación eficientes.

Funcionamiento:

Los árboles B se basan en una estructura jerárquica similar a los árboles binarios, donde cada nodo contiene claves y punteros a nodos hijos. Sin embargo, a diferencia de los árboles binarios, los nodos en un árbol B pueden tener un número variable de hijos, entre m/2 y m, donde m es el orden del árbol.

Las claves en cada nodo se mantienen en orden ascendente, y cada nodo hijo almacena un rango de claves correspondiente a los nodos a los que apunta. La raíz del árbol siempre apunta a al menos un nodo hijo, y todas las hojas se encuentran en el mismo nivel.

Propiedades clave:

- Orden (m): Determina el número máximo de claves y nodos hijos que puede contener un nodo.
- Claves en orden: Las claves dentro de cada nodo se encuentran ordenadas de manera ascendente.
- Punteros a nodos hijos: Cada nodo contiene punteros a sus nodos hijos, indicando el rango de claves que estos contienen.
- Mismo nivel de hojas: Todas las hojas se encuentran en el mismo nivel del árbol.

Operaciones básicas:

- Búsqueda: Para buscar una clave en un árbol B, se recorre el árbol desde la raíz hasta la hoja que contiene el rango de claves al que pertenece la clave buscada. En cada nodo, se compara la clave buscada con las claves del nodo para determinar en qué dirección continuar la búsqueda.
- Inserción: Al insertar una nueva clave, se busca el nodo hoja adecuado y se verifica si hay espacio disponible para ella. Si no hay espacio, se divide el nodo en dos, redistribuyendo las claves y creando un nuevo nodo hijo.
- Eliminación: Para eliminar una clave, se busca el nodo hoja que la contiene y se redistribuyen las claves entre los nodos vecinos para mantener el equilibrio del árbol. En algunos casos, puede ser necesario fusionar nodos.

Ventajas:

Eficiencia en grandes conjuntos de datos: Los árboles B son particularmente eficientes para manejar grandes conjuntos de datos, ya que permiten realizar búsquedas, inserciones y eliminaciones con un menor número de comparaciones en comparación con otras estructuras de datos como los árboles binarios de búsqueda.
Menor acceso a disco: Al almacenar más claves por nodo, los árboles B reducen la cantidad de accesos al disco duro necesarios para recuperar información, lo que mejora el rendimiento general del sistema.
Equilibrio y adaptabilidad: Los árboles B mantienen un equilibrio natural entre los nodos, lo que les permite adaptarse a cambios en la cantidad de datos sin perder eficiencia.

Aplicaciones:

Bases de datos: Los árboles B son ampliamente utilizados en bases de datos para almacenar e indexar grandes volúmenes de datos, permitiendo búsquedas rápidas y eficientes.
Sistemas de archivos: Se utilizan en sistemas de archivos para organizar y acceder a los datos en el disco duro, mejorando el rendimiento de lectura y escritura.
Redes de computadoras: En redes de computadoras, los árboles B se emplean para enrutar paquetes de datos de manera eficiente y encontrar nodos específicos dentro de la red.

Grafo

Los grafos, también conocidos como redes o estructuras gráficas, son una herramienta matemática utilizada para representar relaciones entre objetos. Se componen de dos elementos principales:

- Vértices (nodos): Representan las entidades del sistema que se están estudiando. Pueden ser personas, lugares, objetos, eventos, ideas, entre otros.
- Aristas (enlaces): Conectan los vértices y muestran la existencia de una relación entre ellos. Las aristas pueden ser dirigidas o no dirigidas, ponderadas o no ponderadas.

Funcionamiento:

Los grafos se visualizan como diagramas donde los vértices se representan como puntos o círculos y las aristas como líneas que los conectan. El tipo de arista utilizada indica la naturaleza de la relación:

- Aristas dirigidas: Indican una relación unilateral, donde un vértice (origen) apunta a otro (destino).
- Aristas no dirigidas: Muestran una relación mutua entre dos vértices, sin especificar una dirección.
- Aristas ponderadas: Asignan un valor numérico (peso) a la arista, que puede representar la distancia, el costo, la fuerza de la conexión, entre otras características de la relación.

Propiedades:

Los grafos pueden tener diversas propiedades que caracterizan su estructura y conectividad:

- Grado de un vértice: Número de aristas que inciden en un vértice.
- Conectividad: Indica si existe un camino entre todos los pares de vértices del grafo.
- Ciclos: Caminos cerrados que comienzan y terminan en el mismo vértice.
- Componentes conexos: Subgrafos del grafo donde todos los vértices están conectados entre sí.

Aplicaciones:

Los grafos se utilizan en una amplia variedad de campos debido a su capacidad para modelar relaciones complejas y representar sistemas diversos. Algunas aplicaciones comunes incluyen:

- Redes sociales: Los grafos se emplean para representar las conexiones entre personas en redes sociales como Facebook o Twitter, analizando patrones de interacción y comunidades.
- Redes de transporte: Se utilizan para modelar redes de carreteras, ferrocarriles o rutas aéreas, optimizando la planificación de rutas y el flujo de tráfico.
- Redes informáticas: Los grafos representan la estructura de redes informáticas, permitiendo el análisis de rendimiento, la detección de fallas y el diseño de protocolos de comunicación.
- Biología molecular: Se utilizan para modelar la estructura de moléculas como proteínas o ADN, estudiando las interacciones entre sus componentes.
- Ciencia de materiales: Se emplean para representar la estructura de materiales, analizando propiedades como la resistencia, la conductividad y la porosidad.

Nodo

En el contexto de las estructuras de datos, un nodo es una unidad fundamental que almacena información y se conecta con otros nodos para formar estructuras más complejas. Se asemeja a un bloque de construcción que permite crear diversas organizaciones de datos.

Estructura:

Un nodo básico comúnmente contiene dos componentes principales:

- **Dato**: La unidad de información que se almacena en el nodo. Puede ser un número, un carácter, una cadena de texto, un objeto o cualquier otro tipo de dato.
- **Puntero**: Un enlace que apunta a otro nodo, permitiendo conectar nodos y formar estructuras como listas, árboles o grafos. Los punteros pueden ser simples o múltiples, dependiendo de la estructura de datos específica.

Funcionamiento:

Los nodos interactúan entre sí mediante sus punteros, creando relaciones y definiendo la organización de la estructura de datos. Las operaciones básicas sobre nodos incluyen:

- **Creación**: Se asigna memoria para el nodo y se inicializan sus componentes (dato y punteros).
- **Lectura**: Se accede al dato almacenado en el nodo.
- **Escritura**: Se modifica el dato almacenado en el nodo.

- **Conexión**: Se establecen o eliminan punteros entre nodos para crear o modificar la estructura.

Tipos de nodos:

Los nodos pueden clasificarse según su función y la estructura de datos en la que se utilizan:

- **Nodos de lista:** Contienen un dato y un puntero al siguiente nodo en la lista.
- **Nodos de árbol**: Contienen un dato y punteros a sus nodos hijo (si los hay), formando una estructura jerárquica.
- **Nodos de grafo**: Contienen un dato y punteros a nodos vecinos, representando relaciones en un grafo.

Aplicaciones:

Los nodos son elementos esenciales en diversas estructuras de datos y se utilizan en una amplia gama de aplicaciones, incluyendo:

- **Implementación de algoritmos**: Los nodos forman la base de algoritmos comunes como búsqueda, ordenamiento y recorrido de estructuras de datos.
- **Bases de datos**: Los nodos se utilizan para almacenar y organizar datos en bases de datos, permitiendo un acceso eficiente a la información.
- **Sistemas operativos:** Los nodos son utilizados en la gestión de memoria, la representación de procesos y la implementación de estructuras de datos del núcleo del sistema operativo.
- **Compiladores:** Los nodos se emplean en el análisis sintáctico y semántico del lenguaje de programación, construyendo el árbol de sintaxis abstracta del código.

Matriz Adyacente

En el ámbito de los grafos, una matriz adyacente es una estructura de datos bidimensional que representa las relaciones entre vértices en un grafo de manera eficiente. Se compone de una matriz cuadrada donde cada fila y columna corresponden a un vértice del grafo, y los valores de las celdas indican la existencia o ausencia de una arista entre los vértices correspondientes.

Funcionamiento:

La matriz adyacente se construye de la siguiente manera:

- **Dimensiones**: La matriz tiene dimensiones N x N, donde N es el número de vértices del grafo.
- **Valores de celdas:**
    - **Arista no dirigida:** Si existe una arista no dirigida entre los vértices i y j, la celda (i, j) y (j, i) se establecen en 1.
    - **Arista dirigida**: Si existe una arista dirigida de i a j, la celda (i, j) se establece en 1 y la celda (j, i) en 0.
    - **Sin arista:** Si no existe arista entre los vértices i y j, las celdas (i, j) y (j, i) se establecen en 0.

Ejemplo:

Consideremos un grafo con 4 vértices (A, B, C, D) y las siguientes aristas:

A - B (no dirigida)
B - C (dirigida)
C - D (dirigida)
D - A (no dirigida)
La matriz adyacente para este grafo sería:

	A	B	C	D
A	1	1	0	1
B	1	0	1	0
C	0	0	1	1
D	1	0	0	1

Ventajas:

Las matrices adyacentes ofrecen varias ventajas para la representación de grafos:

- **Eficiencia espacial**: Ocupan N^2 unidades de memoria, lo que las hace relativamente eficientes en espacio para grafos densos (con muchas aristas).
- **Acceso rápido:** Permite acceder a la información sobre la existencia de una arista entre dos vértices de manera rápida y directa, con una complejidad de tiempo de O(1).
- **Implementación simple**: Su construcción y manejo son relativamente sencillos.

Desventajas:

Sin embargo, las matrices adyacentes también presentan algunas desventajas:

- **Ineficiencia para grafos dispersos**: Para grafos dispersos (con pocas aristas), pueden ocupar una gran cantidad de espacio innecesario debido a las celdas vacías.
- **No representan pesos de aristas**: No almacenan información sobre los pesos de las aristas, si estas existen.
- **Modificaciones costosas**: Agregar o eliminar aristas puede requerir modificar varias celdas de la matriz.

Aplicaciones:

Las matrices adyacentes se utilizan en diversas aplicaciones relacionadas con grafos, incluyendo:

- **Algoritmos de búsqueda**: Se emplean en algoritmos de búsqueda como la búsqueda en anchura primero (BFS) y la búsqueda en profundidad primero (DFS) para explorar el grafo y encontrar vértices o caminos específicos.

- **Implementación de grafos**: Se utilizan en la implementación de grafos en lenguajes de programación, proporcionando una forma eficiente de almacenar y manipular la información de las relaciones entre vértices.
- **Análisis de redes**: Se utilizan en el análisis de redes sociales, redes de transporte y otros tipos de redes para estudiar las relaciones entre entidades y la estructura de la red.

Búsqueda BFS

La búsqueda en anchura primero (BFS) es un algoritmo de búsqueda en grafos que explora el grafo nivel por nivel, comenzando desde un vértice inicial y visitando todos sus vecinos antes de pasar al siguiente nivel. Se implementa de manera eficiente utilizando una cola de datos para almacenar los vértices por explorar.

Funcionamiento:

1) **Inicialización**: Se marca el vértice inicial como visitado y se agrega a la cola.
2) **Exploración** por nivel: Se repite lo siguiente:
    a) Se extrae el primer vértice de la cola (vértice actual).
    b) Se exploran todos los vecinos no visitados del vértice actual:
    c) Se marca cada vecino como visitado.
    d) Se agrega cada vecino a la cola.
3) **Finalización**: El algoritmo termina cuando la cola está vacía, lo que significa

Lista Generica

En el ámbito de la programación, las listas genéricas, también conocidas como listas parametrizadas o listas tipo seguro, son un tipo de estructura de datos que permite almacenar colecciones de elementos de un mismo tipo de manera flexible y eficiente. A diferencia de las listas tradicionales, que solo pueden almacenar un tipo de dato específico (por ejemplo, números, cadenas de texto), las listas genéricas ofrecen la ventaja de poder almacenar cualquier tipo de dato que se defina al crear la lista.

Funcionamiento:

Las listas genéricas se basan en el concepto de parámetros de tipo, que permiten especificar el tipo de dato que se almacenará en la lista al momento de su creación. Esto proporciona mayor flexibilidad y seguridad en el manejo de datos, ya que el compilador verifica que solo se inserten elementos del tipo correcto en la lista.

Implementación:

La implementación de listas genéricas varía según el lenguaje de programación, pero generalmente se basan en los siguientes principios:

- **Definición del tipo de dato:** Se define el tipo de dato que se almacenará en la lista utilizando un parámetro de tipo.
- **Creación de la lista:** Se crea una instancia de la lista genérica especificando el tipo de dato como argumento.

- **Inserción de elementos:** Se insertan elementos en la lista utilizando métodos específicos, asegurando que los elementos sean del tipo correcto especificado al crear la lista.
- **Acceso a elementos:** Se accede a los elementos de la lista utilizando índices o iteradores, garantizando que los elementos recuperados sean del tipo correcto.

Ventajas:

Las listas genéricas ofrecen diversas ventajas sobre las listas tradicionales:

- **Flexibilidad de tipos:** Permiten almacenar cualquier tipo de dato, lo que las hace más versátiles y adaptables a diferentes necesidades.
- **Seguridad de tipos:** El compilador verifica que solo se inserten elementos del tipo correcto en la lista, evitando errores de tipo de dato en tiempo de ejecución.
- **Mayor eficiencia:** En algunos lenguajes, las listas genéricas pueden ser más eficientes que las listas tradicionales debido a la optimización del compilador y la ausencia de conversiones de tipo.

Desventajas:

Sin embargo, las listas genéricas también presentan algunas desventajas:

- **Mayor complejidad:** Su implementación puede ser más compleja que las listas tradicionales, especialmente en lenguajes con sistemas de tipos estáticos.
- **Posible penalización de rendimiento:** En algunos casos, la verificación de tipos adicional puede generar una pequeña penalización de rendimiento en comparación con las listas tradicionales.

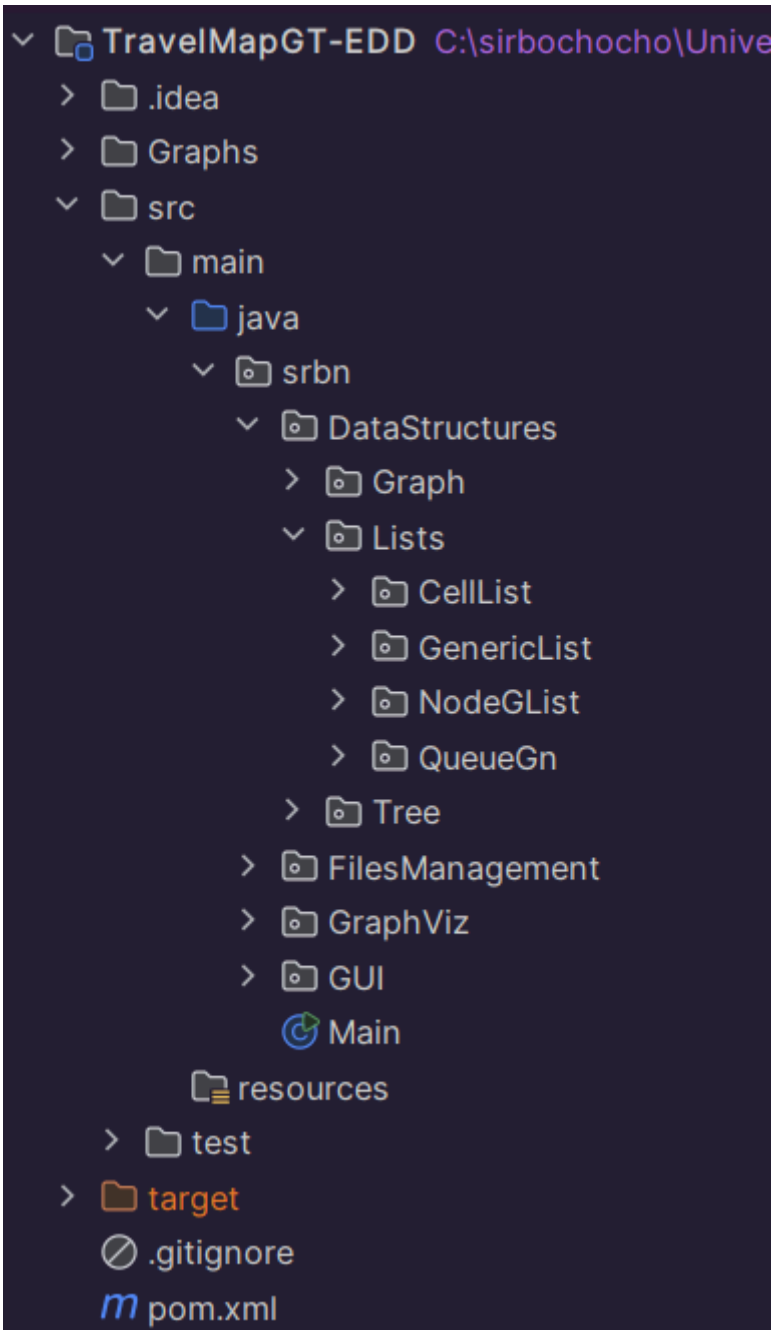
Aplicaciones:

Las listas genéricas se utilizan en una amplia variedad de aplicaciones, incluyendo:

- **Colecciones de datos:** Almacenamiento de colecciones de datos homogéneos de cualquier tipo, como números, cadenas de texto, objetos o incluso otras listas genéricas.
- **Implementación de algoritmos:** Se emplean en la implementación de algoritmos que requieren manipulación de datos de un tipo específico, como algoritmos de búsqueda o ordenamiento.
- **Desarrollo de software:** Se utilizan en el desarrollo de software para crear estructuras de datos flexibles y seguras que se adapten a las necesidades específicas de la aplicación.

Organización del código Fuente

La aplicación TravelMapGT-EDD, presenta la siguiente estructura de paquetes y directorios:



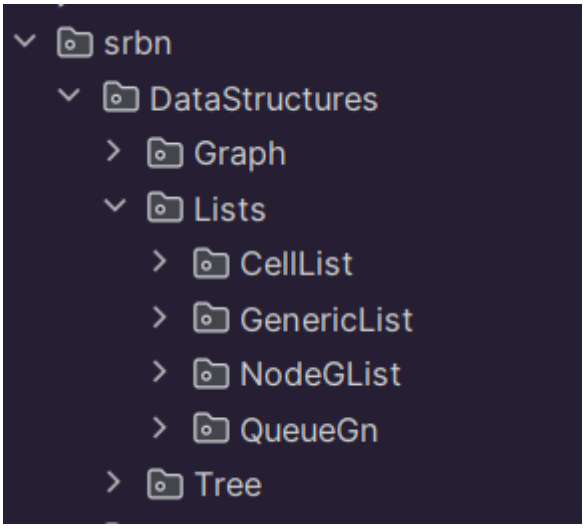
Directorio Java.srbn

El paquete java es el paquete principal de todo el programa, compuesto con la estructura del proyecto distribuida en los siguientes paquetes:

- **DataStructures**
- **FilesManagement**
- **GraphyViz**
- **GUI**

Teniendo la clase *main*.java, como la clase principal con la cual ejecutamos todo el codigo

Paquete **DataStructures**

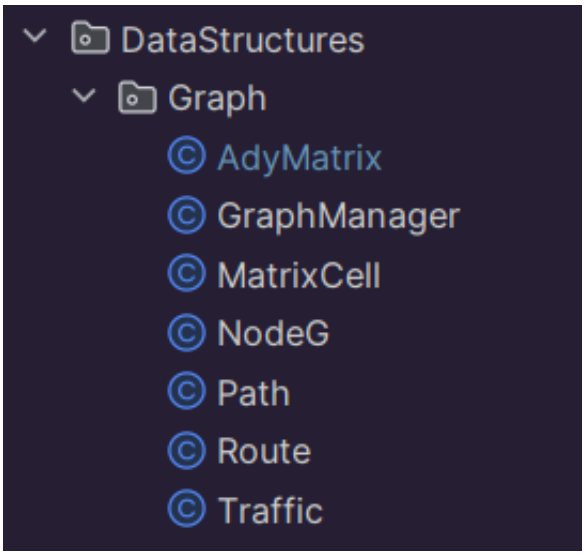


Paquete destinado a estructuras de datos, utilizadas a lo largo del proyecto, distribuido en los siguientes paquetes:

- Graph
- Lists
- Tree

Paquete DataStructures.Graph

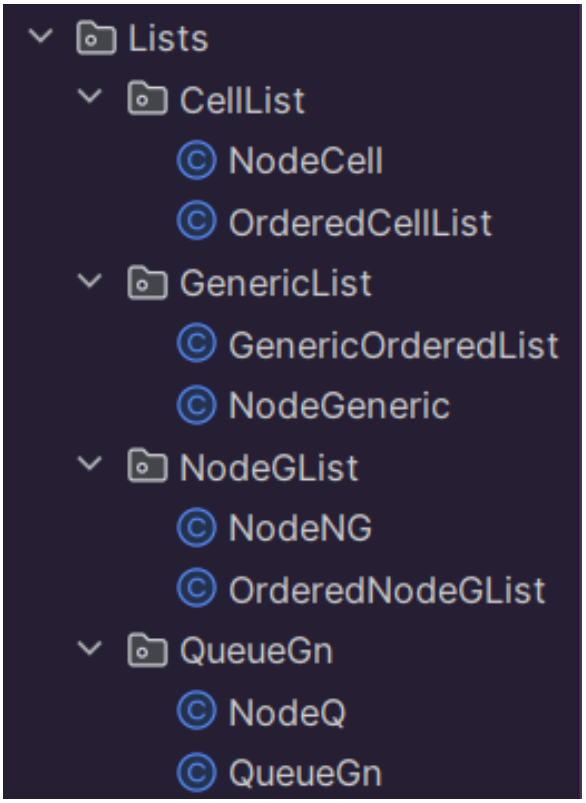
Compuesto por las clases



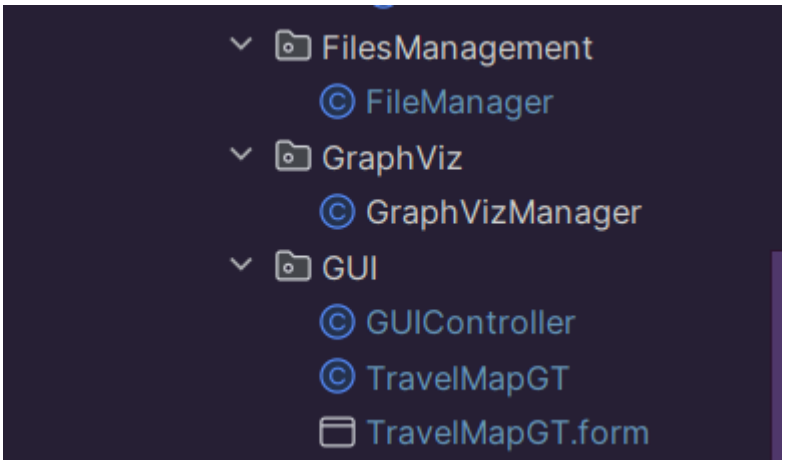
- **AdyMatrix:** Clase dedicada para el manejo e implementación de la matriz adyacente para la conexión de los nodos del grafo.
- **GraphManager:** Clase administradora de los grafos, en esta clase se encuentran funciones para colocar celdas y nodos en la matriz de adyacencia.
- **MatrixCell:** Objeto tipo celda, la cual tiene como atributos un nodo origen, un nodo de destino y los valores que conforman el peso del camino al nodo
- **NodeG:** Objeto que representa a los nodos del grafo, conformado con un id que lo identifica y un alias que lo representa.
- **Route:** Objeto que representa la ruta, teniendo como atributos los valores de distancia, consumo de gasolina, consumo de energía, etc
- **Traffic:** Objeto que representa la posibilidad de tráfico entre rutas, consta de una hora inicial de tráfico, una final y un valor que indica la probabilidad que haya tráfico en dicha ruta.

Paquete DataStructures.Lists

Compuesto por las clases:



- **CellList:** Paquete dedicado a una lista especificamente de celdas, constando:
  - NodeCell: nodo que almacena celdas de la matriz
  - OrderedCellList: lista ordenada de celdas
- **GenericList:** Paquete dedicado a listas genericas, consta de las clases:
  - **GenericOrderedList:** lista ordenada de elementos genericos
  - **NodeGeneric:** nodo que almacena datos genericos
- **NodeGList:** Paquete dedicado a la lista de nodos del grafo, constando de las clases:
  - NodeNG: nodo que almacena un nodo que forma parte del grafo
  - OrderedNodeGList: Lista ordenada de nodos que forman parte del grafo
- **QueueGn:** paquete dedicado a la cola de elementos genericos, constando de:
  - NodeQ: Nodo que forma parte de la cola, almacena datos genericos
  - QueueGn: Cola de elementos genericos



Paquete FilesMnagement

Compuesto por la clase:

- FileManager: administrador de la lectura de archivos de texto

Paquete GraphViz

compuesto por la clase:

- GraphVizManager: administrador de graficas generadas por graphviz

Paquete GUIL:

paquete contenedor de la interfaz grafica, compuesto por

- TravelMapGT: clase UI principal
- GUIController: controlador encargado de llevar la logica utilizada en la interfaz grafica