UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIVISIÓN DE CIENCIAS DE LA INGENIERÍA
ÁREA PROFESIONAL

ESTRUCTURA DE DATOS

MANUAL TÉCNICO: Travel Gt

Carné: 201430801 Nombre: Luis Basilio Baquiax Sic

Tabla de contenido

Requerimientos del programa:

1. Tener instalado JDK de java:
    a. Versión de java: java 17.0.7 2023-04-18 LTS Java(TM) SE Runtime Environment (build 17.0.7+8-LTS-224) Java HotSpot(TM) 64-Bit Server VM (build 17.0.7+8-LTS-224, mixed mode, sharing)
2. Instalación de Graphiz:
    a. Version: graphviz-10.0.1
    b. Link de descarga: https://graphviz.org/download/
3. Requerimientos de la computadora:
    a. Memoria RAM: al menos 2GB

Ejecución del programa

Para ejecutar correctamente el programa se debe tomar en cuenta que el archivo.jar debe estar al mismo nivel que la carpeta "img" para no tener ningún inconveniente. Así como se muestra en la siguiente imagen:

Ubicación del archivo jar y la carpeta "img" 1

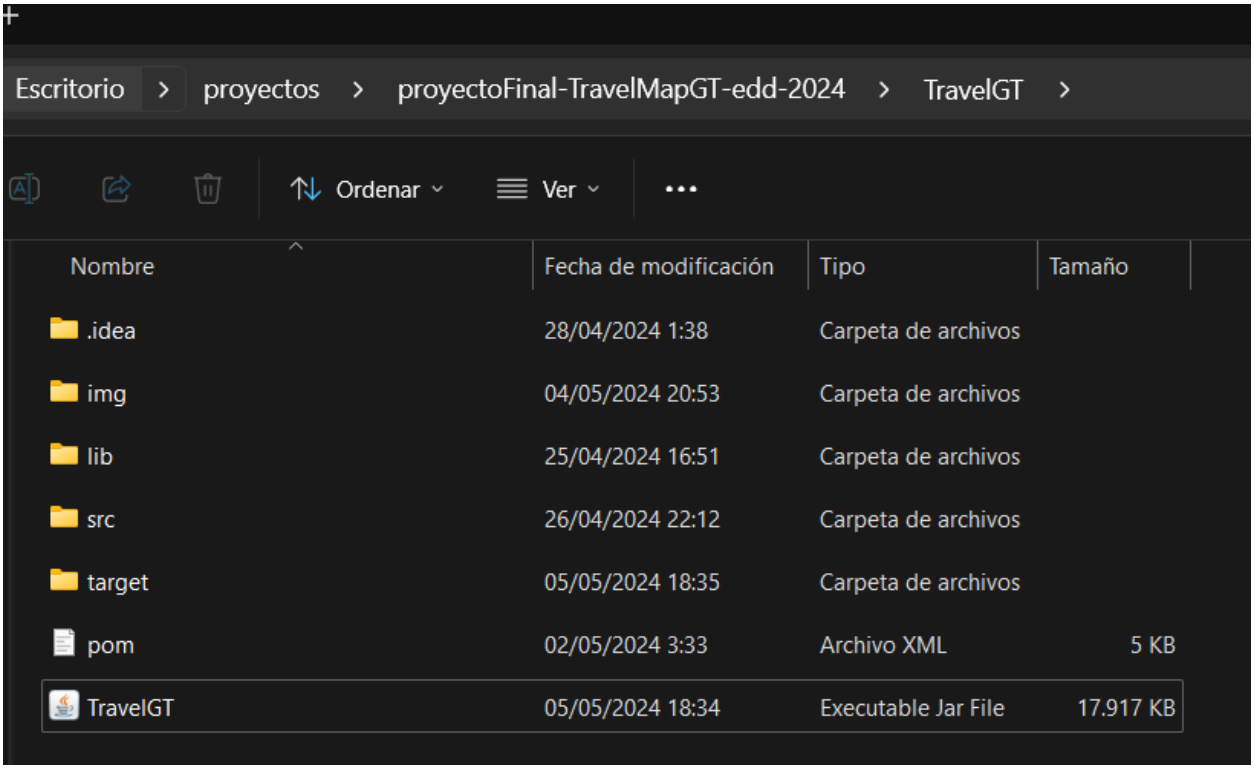
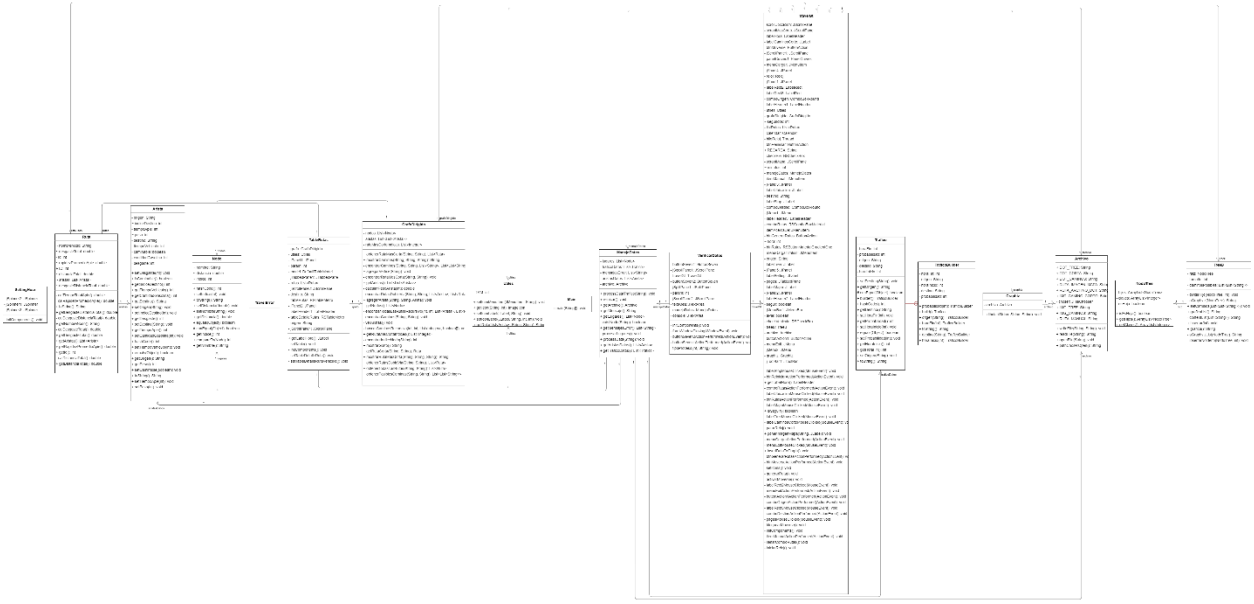


Diagrama de clases:

En la siguiente imagen se presenta el diagrama de clases, sin embargo, en la carpeta "documentación" se encontrará la imagen del diagrama de clases con mejor resolución.



Estructuras

Árbol B:

Atributos:

Variable	Nombre	Descripción
private NodoTree	raiz	Representa la raíz del árbol
private int tamaño	tamaño	Representa las aristas o caminos entre los nodos o lugares
private List<List<String>>	caminosPosibles	Una lista de índices de las aristas que pertenecen a la ruta más corta

Métodos públicos:

Valor de Retorno	Nombre	Parámetros	Descripción

void	insertar	int clave	Simplemente se ingresará números enteros al árbol.
void	setCaminos	List<List<String>> caminosPosibles	Establece la lista de posibles caminos entre dos nodos.
void	insertarNoLleno	NodoTree nodo, int clave	Permitirá ingresar una clave o valor a un nodo que no esté lleno.
void	dividirHijo	NodoTree nodoPadre, int indiceHijo	Divide un nodo hijo que no es hoja
String	graficaList		Retorna un texto con el contenido en formato graphiz
String	aGraphvizList	NodoTree nodo	Servirá para conectar recursivamente los nodos del árbol y complementar la función anterior

Métodos privados:

Grafo dirigido:

Atributos:

Variable	Nombre	Descripción
private List<Nodo>	Nodos	Representa los nodos en el grafo
private List<List<Arista>>	aristas	Representa las aristas o caminos entre los nodos o lugares

private List<Integer>	rutaMasCortaIndices	Una lista de índices de las aristas que pertenecen a la ruta más corta

Métodos públicos:

Valor de Retorno	Nombre	Parámetros	Descripción
void	agregarVertice	String vertice	Representa los nododos en el grafo
void	agregarArista	String origen, String destino, Arista arista	Ingresa una arista en la lista de lista de aristas
String	mostarGrafo		Genera texto en formato graphiz.
String	mostrarCaminos	String origen, String destino, String color	Muestra la ubicación del usuario en el mapa y su destino
List<List<String>>	obtenerPosiblesCaminos	String origen, String destino	Retorna la lista el nombre de los posibles rutas o caminos entre un nodo origen y destino
void	encontrarCaminos	String origen, String destino, List<String> caminoActual, List<List<String>> posiblesCaminos	Método recursive para encontrar los caminos posibles entre un origen y un destino
void	buscarCaminosRecursivo	int origen, int destino, List<Integer> ruta, boolean[] visitados	Servirá para contrar los caminos recursivamente

String	mostrarRutaMasCorta	String origen, String destino, String color	Retorna texto para graficar la ruta más corta entre dos lugares (nodos)
void	encontrarRutaMasCorta	String origen, String destino	Servirá para encontrar la ruta más corta
int	encontrarIndiceNodo	String vertice	Retorna el índice de un dodo en la lista de nodos

Funciones importantes:

Métodos del árbol B:

Insertar(clave)

```
    Si raíz es nulo
        Crear nuevo nodo
        Establecer que es hoja
        Agregar a lista de claves de este nodo la clave
    Sino
        Verificar si no esta lleno la raíz
            Si está lleno
                Crear nuevo nodo
                Asignar que no es nodo el nuevoNodo
                Agregar la raíz como hijo al nuevo nodo
                Asignar como raíz el nuevo nodo
                Asignar el nuevo nodo como la raíz
            Ingresar la raíz actual a un nodo que no está lleno
FIN insertar
```

insertarNoLLeno (nodo, clave)

```
tamañoAuxi = nodo.claves.tamaño -1
    si el nodo es hoja
        mientras que tamañoAuxi>= 0 y clave < claves[tamañoAuxi]
            tamañoAuxi - -
        fin mientras
        agregar clave a la lista de claves del nodo en la posición tamaño +1
    sino
        mientras que tamañoAuxi>= 0 y clave < claves[tamañoAuxi]
```

```
                tamañoAuxi + +
            fin mientras
            agregar clave a la lista de claves del nodo en la posición tamaño +1
fin insertarNoLLeno
```

dividirHijo(nodoPadre, indiceHijo)

```
    nodoHijo = nodoPadre.hijos en el indiceHijo
    nodoNuevoHijo = crear nodo(hijo.esHoja)
    mientras j > 0 y j < tamnioArbol
        nodoQuitar = hijo.claves.remover(tamanioArbol)
        nodoNuevohijo.agregarClave(nodoQuitar)
    fin mientras
    si hijo no es hoja
        a nuevoHijo agregamos un hijo de nodoHijo en la posición tamanio
fin dividirHijo
```