Universidad de San Carlos de Guatemala
Centro Universitario de Occidente
División de Ciencias de la Ingeniería

Estructuras de Datos
Laboratorio

Proyecto Final
TravelMapGT
Manual Técnico

Jorge Anibal Bravo Rodríguez
202131782

1. HERRAMIENTAS USADAS

Java:
Java es un lenguaje de programación de propósito general que se ha vuelto muy popular debido a su portabilidad y capacidad para funcionar en diferentes plataformas. Se utiliza en una amplia gama de aplicaciones, desde aplicaciones empresariales hasta dispositivos móviles y sistemas integrados. Java es conocido por su sintaxis similar a C++ y su enfoque en la programación orientada a objetos.

NetBeans:
NetBeans es un entorno de desarrollo integrado (IDE) que proporciona herramientas para desarrollar aplicaciones en Java y otras tecnologías. Ofrece características como edición de código, depuración, compilación y gestión de proyectos. NetBeans es conocido por su facilidad de uso y su capacidad para admitir múltiples lenguajes de programación.

Visual Studio Code:
Visual Studio Code es un editor de código fuente desarrollado por Microsoft que es altamente personalizable y admite una amplia gama de lenguajes de programación. Ofrece características como resaltado de sintaxis, finalización de código, depuración y control de versiones integrado. Visual Studio Code es conocido por su rendimiento y su amplia gama de extensiones.

Graphviz:
Graphviz es una herramienta poderosa y versátil para la visualización de estructuras de grafos y redes. Su lenguaje descriptivo fácil de usar, su variedad de formatos de salida y sus herramientas adicionales lo convierten en una opción popular para usuarios de todos los niveles de experiencia técnica.

Swing:

Java Swing es una biblioteca gráfica ligera de Java que forma parte de las Java Foundation Classes (JFC). Esta biblioteca proporciona facilidades para construir interfaces gráficas de usuario (GUI) en aplicaciones de escritorio.
Swing ofrece un conjunto amplio de componentes, como botones, tablas, marcos, etc. Estos componentes utilizan la infraestructura de AWT (Abstract Window Toolkit), incluyendo el modelo de eventos AWT, que define cómo una componente reacciona a eventos como los de teclado y mouse

Fedora Linux:
Fedora es una distribución de Linux patrocinada por Red Hat y conocida por su enfoque en la innovación y la adopción temprana de nuevas tecnologías. Es una distribución de propósito general que se utiliza tanto en entornos de escritorio como en servidores. Fedora es conocida por su enfoque en la comunidad y su compromiso con el software libre y de código abierto.

Git:
Git es un sistema de control de versiones distribuido ampliamente utilizado para el seguimiento de cambios en el código fuente durante el desarrollo de software. Permite a los desarrolladores colaborar en proyectos, realizar un seguimiento de las modificaciones, fusionar ramas y revertir cambios. Git es conocido por su velocidad, escalabilidad y capacidad para manejar proyectos grandes.

GitHub:
GitHub es una plataforma de desarrollo colaborativo que utiliza Git para el control de versiones. Permite a los desarrolladores alojar y revisar el código, gestionar proyectos, realizar seguimiento de problemas y colaborar en equipos distribuidos. GitHub es conocido por su amplia comunidad, su integración con otras herramientas y su soporte para proyectos de código abierto.

1. **Grafo construido**

   La clase Grafo en Java representa un grafo dirigido o no dirigido. Permite la creación, manipulación y visualización de grafos.

   Atributos:

   dirigido: Indica si el grafo es dirigido o no dirigido (true para dirigido, false para no dirigido). Esta es una configuración para la modalidad de funcionamiento del grafo, así, representando posibles rutas por conducción o caminata.

   actual: Nodo actual en el grafo.

   origen: Nombre del nodo de origen para la búsqueda de rutas.

   destino: Nombre del nodo de destino para la búsqueda de rutas.

   grafico: Cadena que representa el grafo en formato DOT.

   rutasCaminando: Lista de rutas a pie encontradas.

   rutasConduciendo: Lista de rutas en vehículo encontradas.

   nodosDirigidos: Lista de nodos dirigidos en el grafo.

   nodosNoDirigidos: Lista de nodos no dirigidos en el grafo.

   Métodos:

   agregarNodo(String nombre**)**: Agrega un nodo con el nombre especificado a la lista de nodos dirigidos y no dirigidos.

   generarPar(String origen, String destino, int tiempo_vehiculo, int tiempo_pie, int consumo_gas, int desgaste_persona, int distancia**)**: Crea dos nodos (origen y destino), los agrega al grafo y los conecta con sus respectivos pesos (tiempo, distancia, consumo de gas, desgaste de persona).

   establecerAdyacencia(String origen, String destino**)**: Establece una adyacencia entre dos nodos (dirigida y no dirigida) en base a sus nombres.

   buscarNodoDirigido(String nombre**)**: Busca un nodo dirigido en la lista de nodos dirigidos por su nombre.

   buscarNodoNoDirigido(String nombre**)**: Busca un nodo no dirigido en la lista de nodos no dirigidos por su nombre.

   buscarRutasConducir(String origen, String destino**)**: Busca rutas en vehículo desde el nodo de origen al nodo de destino.

   buscarRutasCaminar(String origen, String destino**)**: Busca rutas a pie desde el nodo de origen al nodo de destino.

   agregarLista(String lista**)**: Agrega una lista de nodos al árbol de soluciones.

establecerCaminata(): Configura el grafo para recorridos a pie y devuelve la lista de nodos no dirigidos.

establecerConduccion(): Configura el grafo para recorridos en vehículo y devuelve la lista de nodos dirigidos.

establecerDestino(String destino**)**: Establece el nodo de destino para la búsqueda de rutas.

establecerAcual(Nodo nodoActual, VentanaPrincipal ventana**)**: Establece el nodo actual, recalcula las rutas si es necesario y actualiza la interfaz gráfica.

generarGrafico(): Genera un archivo DOT y una imagen que representan el grafo general.

generarGraficoMejor(int mejorCriterio**)**: Genera un archivo DOT y una imagen que representan el grafo con la mejor ruta en base a un criterio (menor distancia, menor desgaste o menor tiempo).

imprimirDirigido(): Imprime la lista de nodos dirigidos y sus adyacentes.

imprimirNoDirigido(): Imprime la lista de nodos no dirigidos y sus adyacentes.

obtenerNodosDirigidos(): Devuelve la lista de nodos dirigidos.

obtenerNodosNoDirigidos(): Devuelve la lista de nodos no dirigidos.

establecerDirigido(boolean dirigido**)**: Establece si el grafo es dirigido o no dirigido.

Uso:

La clase Grafo es utilizada para representar la estructura de carreteras que conectan ciertos puntos del país. Estos puntos son tomados desde un archivo de entrada desde el cual se genera la estructura completa, con origen, destino, tiempo a pie, tiempo conduciendo, consumo combustible, desgaste de la persona y la distancia entre ambos puntos.

2. **Clase Nodo**

   La clase Nodo en Java representa un nodo en un grafo dirigido. Permite la creación, manipulación y búsqueda de nodos en un grafo. Para la representación de un grafo no dirigido se hizo una interconección de los puntos origen y destino, poseyendose ambos en sus respectivas listas de adyacencia.

   Atributos:

   nombre: Nombre del nodo.

   adyacentes: Lista de nodos adyacentes al nodo actual.

   distancia: Diccionario que almacena la distancia a cada nodo adyacente.

   resistencia: Diccionario que almacena la resistencia (combustible o desgaste) a cada nodo adyacente.

   tiempo: Diccionario que almacena el tiempo en llegar a cada nodo adyacente.

   grafo: Referencia al grafo al que pertenece el nodo.

   Métodos:

   obtenerAdyacentes(): Devuelve la lista de nodos adyacentes.

   obtenerNombre(): Devuelve el nombre del nodo.

   buscarNodo(String listaNodos, String destino**)**: Busca el nodo de destino en forma recursiva y agrega la ruta a la lista listaNodos.

   agregarValores(String destino, int tiempo,int distancia, int resistencia**)**: Agrega los valores de tiempo, distancia y resistencia al nodo de destino especificado.

   obtenerDistancia(String destino**)**: Devuelve la distancia al nodo de destino especificado.

   obtenerResistencia(String destino**)**: Devuelve la resistencia al nodo de destino especificado.

   obtenerTiempo(String destino**)**: Devuelve el tiempo en llegar al nodo de destino especificado.

   agregarAdyacente(Nodo nodo**)**: Agrega un nodo adyacente al nodo actual.

   mostrarAdyacentes(): Imprime la lista de nodos adyacentes.

3. **Clase camino**

   La clase Camino en Java representa un camino posible desde un nodo de origen a un nodo de destino en un grafo. Almacena la lista de nodos que conforman el camino, así como los valores de desgaste, tiempo y distancia del recorrido.

   Atributos:

   camino: Lista de nodos que conforman el camino.

   desgaste: Valor total de desgaste acumulado en el recorrido.

   distancia: Distancia total recorrida en el camino.

   tiempo: Tiempo total empleado en recorrer el camino.

   cam: Representación del camino en formato Graphviz.

   Métodos:

   agregarNodo(Nodo nodo**)**: Agrega un nodo a la lista del camino y actualiza los valores de desgaste, distancia y tiempo.

   establecerCam(String cam**)**: Establece la representación del camino en formato Graphviz.

   obtenerCam(): Devuelve la representación del camino en formato Graphviz.

   obtenerDesgaste(): Devuelve el valor total de desgaste acumulado en el recorrido.

   obtenerTiempo(): Devuelve el tiempo total empleado en recorrer el camino.

   obtenerDistancia(): Devuelve la distancia total recorrida en el camino.

   obtenerRuta(): Devuelve una cadena con los nombres de los nodos que conforman el camino.

   obtenerCaminoResaltado(): Devuelve una cadena en formato Graphviz con los nodos del camino resaltados en verde.

4. Manejador de archivos

La clase ManejadorArchivos en Java proporciona métodos para escribir y generar archivos relacionados con grafos.

Métodos:

escribirArchivo(String rutaArchivo, String contenido**)**:

Crea un archivo en la ruta especificada si no existe.

Elimina el contenido existente del archivo.

Escribe el contenido proporcionado en el archivo.

Muestra un mensaje de éxito o error en la consola.

generarGrafo(String rutaGrafo, String nombreGrafo**)**:

Genera una imagen PNG del grafo a partir del archivo DOT especificado en rutaGrafo.

Guarda la imagen con el nombre nombreGrafo.png.

Muestra un mensaje de éxito o error en la consola.

Uso:

La clase ManejadorArchivos se puede utilizar para escribir archivos de texto, como rutas o resultados de búsqueda, y para generar imágenes PNG de grafos a partir de archivos DOT.

5. Archivo Entrada

La clase ArchivoEntrada en Java se encarga de leer un archivo de texto con formato específico y generar las instancias de los nodos y sus conexiones en la clase Grafo.

Pasos del funcionamiento:

Se utiliza un objeto BufferedReader para leer el archivo de texto línea por línea.

Cada línea se divide en partes (tokens) utilizando el separador "|".

Procesamiento de cada línea:

Se verifica que la cantidad de tokens sea la esperada (7).

Se capturan los valores de cada campo (origen, destino, tiempos, consumo, desgaste, distancia) y se convierten a sus tipos de datos correspondientes (String, int).

Creación de nodos y conexiones:

Se utiliza el método generarPar de la clase Grafo para crear dos nodos (origen y destino) si no existen en el grafo.

Se establece la conexión entre los nodos creados utilizando los valores de pesos (tiempo_vehiculo, tiempo_pie, consumo_gas, desgaste_persona, distancia).

Manejo de errores:

Se utiliza un bloque try-catch para capturar posibles errores de formato numérico en los datos del archivo.

En caso de error, se imprime un mensaje indicando la línea donde ocurrió el problema.