NOMBRES:

Sebastián Santos

Gino Fuschetto

ASIGNATURA:

Algoritmos Clásicos

PROFESOR/A:

Freddy P

TEMA:

Proyecto final

FECHA DE ENTREGA:

04-12-2025

INFORME FINAL

ESTRUCTURACIÓN DEL CÓDIGO:

Básicamente para la creación del proyecto implementamos el grafo como un HashMap, donde las entradas son objetos tipo estacion, esta clase Estación tiene atributos descriptivos como el nombre y la zona, algunos para ubicación en el pane del Mapa como los son la latitud (posicion x) y la longitud (posición y) pero lo más importante es que guardan información como la velocidad y su costo base que sirven para temas de ponderación de las rutas.

Justamente las rutas constituyen la otra gran parte del mapa y es que en el Hashmap los values son la lista de rutas que tiene esa estación, las rutas también tienen datos identificativos y datos como la distancia o tiempo que sirven para la ponderación.

Patrón de diseño utilizado: Bueno a nuestro entender usamos el patrón singleton ya que tenemos una clase GrafoTransporte que guarda el mapa y tiene los métodos referentes a él. Aunque sí es cierto que también contamos con el servicio que guarda tanto esta instancia como una lista de las estaciones y rutas para ser guardadas en la base de datos

ALGORITMOS IMPLEMENTADOS:

El paquete util de nuestro proyecto guarda todos los algoritmos que se implementaron, **los que no son mínimos sino ya conocidos y especificados para este proyecto. Entre estos están:**

Floyd Warshall: Este es el algoritmo para crear la matriz de distancias que guarda la distancia de cada estación a cada estación en su respectiva posición [Estacion Actual][EstacionDestino]. Es O(n^3) por naturaleza, lo primero que hace es llenar todas las casillas de la matriz con valores infinitos, luego recorre entre las rutas de cada estación para poner su valor directo entre el nodo del índice actual del bucle principal y cada estación que es el destino de las rutas que tiene esa misma estación. De esa manera ya solo queda verificar con un triple bucle si existen uno o múltiples valores k intermedios que hagan un camino más corto en base solo a la distancia (osea este es el parámetro que utiliza).

Dijkstra

Implementa búsqueda de rutas óptimas según cuatro criterios seleccionables: distancia, tiempo, costo y transbordos. Utiliza una cola de prioridad que ordena los caminos por valor acumulado y cantidad de transbordos como criterio de desempate. El algoritmo explora los vecinos de cada estación, evaluando si los nuevos caminos encontrados mejoran los almacenados en el top 3 por estación, garantizando optimalidad respecto al criterio elegido.

Bellman-Ford

Adaptación del algoritmo clásico optimizada para encontrar el Top 3 de rutas más económicas. Itera n-1 veces sobre todas las rutas, actualizando los mejores caminos conocidos hacia cada estación. Cuenta transbordos automáticamente detectando cambios de línea entre rutas consecutivas. Incluye detección de ciclos negativos en una iteración adicional, elevando excepción si se encuentran. Finalmente reconstruye los tres mejores caminos completos desde origen a destino.

Prim

Aunque no esté en uso dentro del programa, la implementación del algoritmo de Prim se basa en crear un árbol de expansión mínima que cubra todos los vértices, en este caso estaciones, con la menor cantidad de ponderación posible. El algoritmo crea otro grafo con todas las estaciones del grafo original y después comenzando desde una estación arbitraria, va seleccionando la ruta con menor ponderación, asegurándose en cada iteración de elegir una ruta que no cree un ciclo, seguirá este proceso hasta añadir n-1 rutas, donde n representa la cantidad de estaciones.

Kruskal

Tampoco está en un uso en el programa, pero similar al de Prim, el algoritmo de Kruskal es un algoritmo que crea un árbol de expansión mínima. Comienza igual guardando todas las estaciones en un grafo nuevo. Este se diferencia al coger y guardar todas las rutas del grafo en una lista y organizarlas basadas en ponderación mínima. Se implementó una clase auxiliar Unión, que es un Disjoint Union Set, el cual se usa para detectar ciclos de forma eficiente. El algoritmo repite eligiendo la ruta con menor ponderación cada vez hasta llegar a n-1 rutas, donde n representa la cantidad de estaciones.

Clases de Utilidad

Las clases DatoCamino y Caminos encapsulan la lógica compartida de los algoritmos. DatoCamino es una estructura que almacena información completa de un camino parcial (estación actual, costo acumulado, predecesor, transbordos, línea anterior, tipo de estación) implementando Comparable para ordenamiento automático. Caminos agrupa métodos auxiliares como reconstrucción de caminos completos a partir de datos parciales y creación de objetos ResultadoRuta con métricas calculadas.

Randomización

Clase de utilidad que simula eventos aleatorios en el sistema de transporte. Genera números aleatorios del 1 al 100 y los clasifica en tres categorías: choques (20%), eventos variados (95%) o ausencia de eventos. Su método calcularEvento() combina ambas operaciones en una única invocación, permitiendo modelar incidencias realistas en las rutas.

DECISIONES DE DISEÑO:

Decidimos usar esa idea del hashmap porque era la forma más manejable de abordar el proyecto, decidimos seccionar bien todo el proyecto, osea las funciones o algoritmos en el paquete util, el codigo que define las entidades en el paquete estructura y el paquete visual que cuenta con los controllers, las clases que abren los fxml y el principal controller + el main que corre la app.

Si hablamos de diseño visual decidimos hacer una barra vertical con cada una de las funciones que ofrece nuestro código. Usamos colores como el verde, blanco, amarillo ya que usualmente los relacionamos con los mapas de la vida real y asi.

La ventana principal aparte de con la barra, tiene a su derecha el mapa, que no es más que un pane, donde por cuestiones de no gustarnos ninguna ninguna librería, decidimos simplemente, aprovechando que ya teníamos los apartados de longitud y latitud, crear círculos en un inicio de radio 15 “unidades en la pantalla” representando las estaciones, aunque luego la cambiamos por los logotipos de cada tipo de estación + el color que se le asigna a la misma. Las rutas la representamos con líneas a las cuales les creamos una flecha usando triángulos para indicar la dirección, hay un botón de actualizar que es la que llama a la clase randomización pasa el numero a la función que actualiza el tiempo por evento y sale

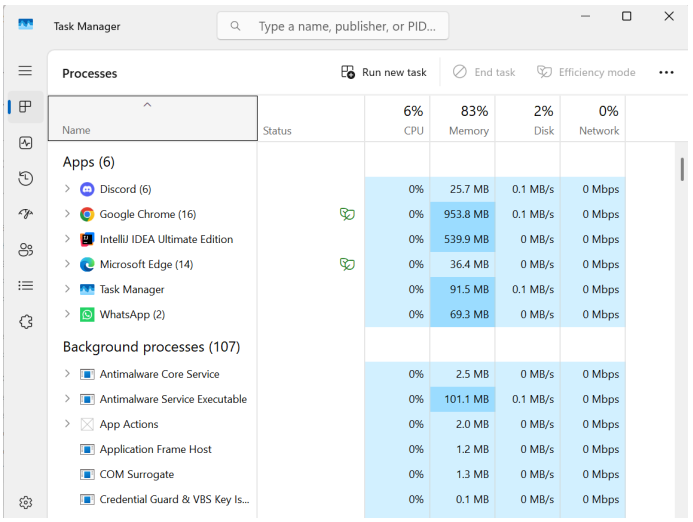
la respectiva alerta de en qué ruta paso según qué evento y se actualiza su tiempo ya que se atrasa. Los resultados son dos comboBox donde eliges tu origen y tu destino y si hay ruta con otro mapa que tiene al lado esa ventana pues se resalta el camino y en un cuadro a la izquierda salen los detalles más específicos de la ruta que elegiste entre las 3 mejores que había.

PRUEBAS REALIZADAS:

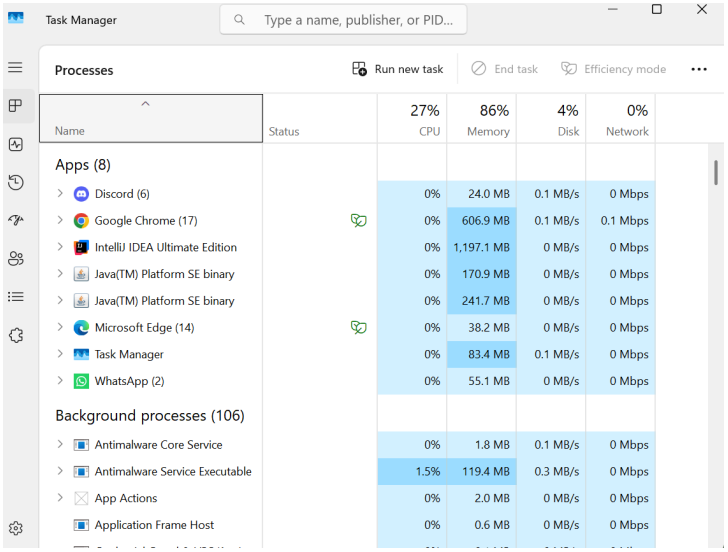
Primero todo funciona bien, verificamos cada validación, lo cual nos ayudó a ver que por ejemplo al principio los spinners te dejaba poner texto y tomaba ese texto como valor cuando tenía que ser numérico y así, osea nos enfocamos en que todo funcionase al 100 por 100.

En cuanto al rendimiento con grandes volúmenes de datos ingresados el task manager marca que estamos usando como promedio 1gb-1,2gb de memoria, pero hay que considerar que de normal sin correrlo solo por abrir el intelliJ nos marca 800 (supongo que se basa en también otros proyectos y en general el peso del editor de código), en cuanto a CPU no suele llegar a un aumento más de 22% (con datos de volumen grande y con el algoritmo más demandante (Floyd + dibujarMapaCompleto), para operaciones normales de adición, listado, eliminación no suele llegar al 4% de aumento , y para búsqueda alrededor del 9% de aumento.

PRE-EJECUCIÓN



POST-EJECUCIÓN (Segundos después de realizar un proceso con gran volumen de datos)



Ojo en las fotos es todo el task manager pero se puede evidenciar que al estar cerrado está en 6% y crece un 21% en la cpu tras varias operaciones realizadas + la cantidad de datos cargados

NOTA: LE DEBEMOS LAMENTABLEMENTE LA FOTO DE LA CPU CUANDO SE HACEN LOS DISTINTOS PROCESOS EN SU PRECISO INSTANTE, NO SABÍAMOS SI ERAN REQUERIDOS PERO ES DIFÍCIL TIRAR FOTO A ESO DEBIDO A QUE SON PROCESOS QUE NO SE QUEDAN EJECUTÁNDOSE SINO QUE SE EJECUTAN Y TERMINAN EN FRACCIONES DE SEGUNDO