MANUAL TECNICO Y DE COMPLEJIDADES

GIAN CARLOS FIGUEROA

JORGE ALEJANDRO AGUIRRE

LUZ ENITH GUERRERO

DOCENTE

PROYECTO

UNIVERSIDAD DE CALDAS

FACULTAD DE INGENIERÍAS

INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

ANALISIS Y DISEÑO DE ALGORITMOS

MANIZALES CALDAS

MAYO DEL 2017

MANUAL TECNICO

|  |  |  |  |
| --- | --- | --- | --- |
| Clase | Descripción | Métodos |  |
| AlgoritmosRuta | Contiene el algoritmo de Floyd Warshall, que permite calcular la ruta mínima entre los nodos de un grafo. | * LlenarPesos: Permite llenar la matriz de pesos para calcular el mínimo camino. * floydWarshall: Contiene las operaciones propias de la filosofía del algoritmo de Floyd warshall para calcular caminos minimos. * obtenerCamino: Apartir de la matriz calculada de Floyd Warshall, se obtiene el camino de nodos. | * FloydWarshall: Posee tres ciclos, cada uno va hasta la cantidad de nodos del grafo, por lo tanto su peor caso es: * Obtener camino en su peor caso, puede obtener un camino que involucre todos los nodos, por lo tanto su complejidad es: para el mejor caso es si solo debe recorrer una arista por lo tanto su complejidad es: * Complejidad de los demás métodos: . |
| AreaItems | Contiene una lista de los elementos que puede seleccionar el usuario para construir la ciudad e interrupciones | * CargarComponente: Llena una lista con los componentes que puede seleccionar el usuario en la interfaz para construir la ciudad. | * CargarComponentes: No posee ciclos. Complejidad: . * Complejidad de los demás métodos: . |
| Item | Elemento con atributos como área y tipo de imagen que puede seleccionar el usuario para construir la ciudad. |  | * Complejidad de los métodos: |
| Componente | Representa cada elemento con los que se puede construir la ciudad por lo tanto contiene un nombre, una ruta de la imagen, una área y un id si se considera nodo. |  | * Complejidad de los métodos: |
| Interrupción | Indica que un componente es del tipo interrupción, almacena el componente que está por debajo de la interrupción para recuperarlo después de eliminar la interrupción |  | * Complejidad de los métodos: |
| Ciudad | Contiene una matriz de los componentes que tiene la ciudad, cantidad de nodos, y lista de interrupciones. | * EsCalle, esCarretera, esVia, esViaCortada, esCruce: Verifica si el tipo de componente pertenece a una calle, carretera o cruce. * MarcarNodosAdyacentes: Cuando el usuario crea una interrupción en la ciudad, se ejecuta este método para marcar a los componentes adyacentes a la interrupción como nodo. * EliminarNodosAdyacentes: Al eliminarse una interrupción se deben eliminar los nodos adyacentes marcados a la misma. * actualizarCiudad: Cuando se carga la ciudad desde un archivo de texto, se debe reconstruir sus propiedades para los vehículos. * actualizarCiudadPersonas: Cuando se carga la ciudad desde un archivo de texto, se debe reconstruir sus propiedades para los peatones * modificarNodos: modifica los id de los nodos de menor a mayor. (Para un mejor debug). * marcarNodo: Asigna a un componente un id de nodo. | * EsCalle, esCarreter, es vida, esCruce: Son condicionales para establecer el tipo de componente: Complejidad es: * MarcarNodosAdyacentes: condicionales para verificar si se debe marcar como nodo un componente. Complejidad: * EliminarNodosAdyacentes: Para desmarcar los nodos se hacen uso de condiciones. Complejidad: * ActualizarCiudad, actualizarCiudadPersonas: Establece los tipos de componentes en la ciudad por lo tanto recorre la matriz de la ciudad su complejidad es: * ModificarNodos: Para reorganizar los nodos es necesario recorrer la matriz de la ciudad por lo tanto la complejidad es: * marcarNodo: Le agrega un nodo a un componente: Complejidad: . * Otros metodos complejidad: . * Otros métodos: |
| EntidadMovimiento | Contiene la ruta de la imagen de la entidad que se va a mover, el área y posee una lista de aristas que es el camino que el carro tomara. |  | * Complejidad de los métodos: . |
| CarroMovimiento | Contiene la ciudad, el grafo la ubicación origen y los nodos de destinos | * Iniciar: Permite dar inicio al hilo para que se empieze a mover el carro. * Parar: Detiene el hilo de movimiento. * Run: Permite cambiar las coordenadas de movimiento del vehículo según el camino que tiene asignado. * idNodoComponente: Permite obtener el id del nodo de la última ubicación del carro. * buscarCamino: Permite obtener caminos automáticos. * PuedoPasar: Verifica si es posible pasar por un cruce (Semaforos inteligentes). * ReconstruirUbicacion y reconstruirDestinos: Permite obtener los id de nodos de los destinos que fueron cambiados cuando el usuario creo alguna interrupción en la ciudad. * RecuperarDestinos: Recupera los id de destinos según los cambios en la ciudad. | * Run: Si el camino recorre todo el grafo, la complejidad en el peor caso es: y para el mejor caso es recorrer una arista es decir . * idNodoComponente: Devuelve el id de nodo de un componente. Complejidad es: . * buscarCamino: En el peor caso si el nodo objetivo para encontrar aristas tiene adyacencia con todos los demás nodos entonces su complejidad es de: . * puedoPasar: Se verifica a través de condicionales si en un cruce existen vehiculos: Complejidad . * ReconstruirUbicacion: Se asigna a una variable ubicación el componente donde está ubicado el vehiculo. Complejidad: . * reconstruirDestinos y recuperarDestinos: Recorren la matriz de la ciudad, además de los destinos del vehiculo, los destinos en el peor caso serían todos los nodos de la ciudad por lo tanto la complejidad en el peor caso es . Y para el mejor caso sería un solo destino entonces: . |
| Persona Movimiento | Contiene la ciudad, el grafo no dirigido, el origen y la lista de destinos. | * Iniciar: Permite dar inicio al hilo para que se empiece a mover la persona. * Parar: Detiene el hilo de movimiento. * Run: Permite cambiar las coordenadas de movimiento de la persona según el camino que tiene asignado. * buscarCamino: Permite obtener caminos automáticos. * PuedoPasar: Verifica si es posible pasar por una cebra. * ExisteCarro: Permite conocer si existe un carro en alguna posición de la ciudad. | * Run: Si el camino recorre todo el grafo, la complejidad en el peor caso es: y para el mejor caso es recorrer una arista es decir . * buscarCamino: En el peor caso si el nodo objetivo para encontrar aristas tiene adyacencia con todos los demás nodos entonces su complejidad es de: . * puedoPasar: Se verifica a través de condicionales si en un cruce existen vehiculos: Complejidad . * Otros metodos: Complejidad . |
| Arista | Contiene las coordenadas de inicio y fin de la arista y una lista de componente que conforman la arista como también la velocidad a la que se puede transitar, longitud y tráfico de la arista. | Arista: Prepara las posiciones iniciales y finales de la arista, además de la dirección, velocidad y lista de componentes. | * Arista: Para definir la dirección de la arista se hace uso de condicionales, no existen bucles por lo tanto la complejidad es: . |
| RutaCorta | Hereda de AlgoritmoRuta, Permite calcular la ruta más corta sobrescribiendo el método llenarPesos con la longitud de cada uno de las aristas. | * LlenarPesosGrafoDirigido: A partir del grafo dirigido para los vehículos se obtiene la longitud en la que se puede transitar en cada arista. * LlenarPesosGrafoNoDirigido: A partir del grafo no dirigido para las personas se obtiene la velocidad en la que se puede transitar en cada arista. | LLenarPesosGrafoDirigido, llenarPesosGrafoNoDirigido: Se utiliza dos bucles para recorrer el grafo que es una matriz de nodos de adyacencias para poder llenar la matriz de pesos y con esto obtener la ruta corta. Complejidad: . |
| RutaVeloz | Hereda de AlgoritmoRuta, Permite calcular la ruta más corta sobrescribiendo el método llenarPesos con la velocidad de cada una de las aristas. | * LlenarPesosGrafoDirigido: A partir del grafo dirigido para los vehículos se obtiene la velocidad en la que se puede transitar en cada arista. | LLenarPesosGrafoDirigido: Se utiliza dos bucles para recorrer el grafo que es una matriz de nodos de adyacencias para poder llenar la matriz de pesos y con esto obtener la ruta veloz. Complejidad: . |
| RutaTrafico | Hereda de AlgoritmoRuta, Permite calcular la ruta más corta sobrescribiendo el método llenarPesos con la cantidad de tráfico que hay en cada una de las aristas. | * LlenarPesosGrafoDirigido: A partir del grafo dirigido para los vehiculos se obtiene el número de carros que existen en las aristas | LLenarPesosGrafoDirigido: Se utiliza dos bucles para recorrer el grafo que es una matriz de nodos de adyacencias para poder llenar la matriz de pesos y con esto obtener la ruta con menor trafico. Complejidad: . |
| GrafoDirigido | Permite obtener un grafo dirigido a partir de la ciudad, para el cálculo de rutas para los vehiculos. | * CrearGrafo: a partir de un objeto ciudad, se calculan el número de nodos, y de aristas para un grafoDirigido. * CrearArista: Permite crear una arista a partir de los puntos de inicio y final y la lista de componentes. * AnadirArista: se asigna a los nodos al componente de inicio y final y se añade la arista al grafo. * verArista: Permite establecer la dirección de cada arista. | * CrearGrafo: Con dos ciclos realiza un recorrido a la matriz de la ciudad para construir el grafo. Complejidad . * crearArista: Construye apartir de parámetros una arista, complejidad: . * AñadirArista: Es una serie de condicionales para calcular la dirección de la arista dentro de la ciudad. Su complejidad es: . |
| GrafoNoDirigido | Permite obtener un grafo no dirigido a partir de la ciudad, para el cálculo de rutas para las personas. | * CrearGrafo: A partir de un objeto ciudad se calculan el número de nodos y de aristas para construir el grafo no dirigido. * CompletarGrafo: Permite establecer las calles paralelas por donde puede circular la persona, además de los cruces. Lo anterior con aristas. * anadirArista: Asigna nodos al inicio y final de la aristay se añaden al grafo. * verArista: Permite establecer el tipo de arista y su dirección. | * CrearGrafo: Con dos ciclos realiza un recorrido a la matriz de la ciudad para construir el grafo no dirigido. Complejidad . * CompletarGrafo: Recorre la matriz ciudad para completar el grafo por lo tanto su complejidad es: * crearArista: Construye apartir de parámetros una arista, complejidad: . * AñadirArista: Es una serie de condicionales para calcular la dirección de la arista dentro de la ciudad por donde circularan los peatones. Su complejidad es: . |
| FormVentana | En conjunto con panelVentana, permite ingresar un nuevo vehiculo o peaton, inicializar la ciudad con los componentes, permite cargar la ciudad o inicializarla como nueva. | * GraficarCiudad: Permite establecer los parámetros de ancho y alto de la ciudad como también del área de ítems. * guardarCiudad: Permite guardar la ciudad que se tiene construida hasta el momento. * cargarCiudad: Apartir de un archivo de texto es posible reconstruir una ciudad previamente guardada. * ingresarCarro: Permite ingresar un carro a la ciudad e iniciar recorrido de forma automática. * ingresarPersona: Permite ingresar un peatón a la ciudad e iniciar el recorrido de forma automática. | * GraficarCiudad: Permite darle propiedades a la ciudad. Su complejidad es: . * guardarCiudad: Es necesario recorrer la ciudad por lo tanto su complejidad es: . * cargarCiudad: Es necesario reconstruir la ciudad por lo tanto recorrer la matriz de ciudad: su Complejidad es: . * IngresarCarro: añade a una lista de vehiculos: Complejidad . * IngresarCarro: añade un peaton a una lista de personas: Complejidad . |
| PanelVentana | Permite establecer los eventos de ruta corta, ruta de menor tráfico y de mayor velocidad, añadir y remover componentes para la ciudad e interrupciones. | * modificarGrafo: Permite enviar parámetros como el grafo, la ciudad, el tipo de camino seleccionado, los destinos y obtiene el camino según el seleccionado. Para asignárselo al vehículo. * modificarGrafoPersonas: Permite enviar parámetros como el grafo no dirigido completo, la ciudad, los destinos y obtiene el camino mínimo. Para asignárselo al peatón. * destinoCercaPersonas: Algoritmo de backtracking que permite obtener todos los caminos hacia los distintos destino y seleccionar el más corto, está apoyado por el método: buscarCaminoCercaPersonas; se asigna a los peatones. * DestinoSecuenciaPersonas: Permite obtener el camino más corto, pasando por los destinos en secuencia para asignárselo a los peatones. * destinosCerca: Algoritmo de backtracking que permite obtener todos los caminos hacia los distintos destino y seleccionarlo según el tipo de camino (corto, menor tráfico y mayor velocidad), está apoyado por el método: buscarCaminoCerca; este es asignado a los vehículos. * destinosSecuencia: Permite obtener un camino según el tipo seleccionado (más corto, menor tráfico, y mayor velocidad), pasando por los destinos en secuencia; este es asignado a los vehículos. * obtenerTrafico: Recorre el grafo que es la representación de la ciudad en busca de la cantidad de vehículos que están transitando por cada una de las aristas. * PaintComponent: Permite pintar los componentes en la ciudad | * modificarGrafo, modificarGrafoPersonas: Llaman a los métodos destinosCerca, o destinosSecuencia por lo tanto la complejidad es: * destinosCercaPersonas: Invoca al método buscarCaminoCerca que es exponencial por lo tanto su complejidad es: * buscarCaminoCercaPersonas: permite obtener el camino minimo para los peatones de cada uno de los nodos de destino hacia los mismo. Dado por la ecucacion de recurrencia = * destinosCerca: Invoca al método buscarCaminoCerca que es exponencial por lo tanto su complejidad es: * buscarCaminoCerca: permite obtener el camino minimo para los vehículos de cada uno de los nodos de destino hacia los mismo. Dado por la ecucacion de recurrencia * destinoSecuenciaPersonas: Se hace un recorrido de los nodos de destino invocando al método obtener camino de complejidad: . Si los destinos invlucra a n nodos entonces la complejidad es: . * destinosSecuencia: Se hace un recorrido de los nodos de destino invocando a el método obtener camino de complejidad: . Si los destinos invlucra a n nodos entonces la complejidad es: . * obtenerTrafico: Recorre el grafo que es una matriz de ayacencia y además la recorre para cada carro en la ciudad por lo tanto su complejidad es: . * PaintComponent: Recorre la matriz ciudad para dibujar los componente su complejidad es: . |

Complejidad total: