Introducción y presentación del problema

Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas que pueden ser orientados o no. Típicamente, un grafo se representa mediante una serie de puntos (los vértices) conectados por líneas (las aristas). Si en un grafo toda arista es dirigida se denominará "grafo dirigido" ya que entre un par de vértices existe una relación unívoca.

El recorrido de un grafo se realiza desde un vértice v dado, visitando el resto de vértices a través de caminos de v, si el grafo no es conexo, desde v no se podrán visitar todos los vértices del grafo. Los vértices se irán marcando como “visitados”, se recorrerán siguiendo las aristas del grafo, y se usarán estructuras adicionales para ir gestionando los vértices pendientes de ser procesados.

El ambiente gráfico o interfaz gráfica es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

TkInter (de TK Interface) es un módulo que nos permite construir interfaces gráficas de usuario multiplataforma en Python utilizando el toolkit Tk. Python incluye este módulo por defecto, lo que hace que sea un tookit muy popular. TkInter, además, es robusto, maduro y muy sencillo de aprender y de utilizar, contando con una amplia documentación.

Con estas bases y, considerando que actualmente la Ciudad de México se encuentra dentro de las 10 ciudades más pobladas del mundo, por lo que la demanda de usuarios del transporte público es muy grande, y debido al servicio poco efectivo de los medios de transporte y al escaso mantenimiento que se les da, el trasladarnos de un lugar a otro puede ser una tarea que nos tome bastante tiempo, por lo tanto se creó un programa que represente las líneas del metro de la Cd. De México, definiendo como nodos a las estaciones y las vías entre dos estaciones como aristas.

El programa puede calcular la ruta más corta entre dos estaciones ya sea en número de estaciones o en tiempo de viaje, puede agregar o eliminar estaciones, modificar el nombre de las estaciones o el tiempo entre dos nodos, así como adyacencias a una arista.

También carga la red del metro de un archivo y guarda esta misma, después de ser modificada, en otro archivo.

Definición intuitiva (ejemplo didáctico) y definición formal

Tomando como base la red del metro de la Cd. De México podemos imaginar que nuestro programa funciona de la siguiente forma:

Si necesitamos viajar de Xola a Candelaria las dos formas más sencillas de llegar sería primero, Xola - Viaducto - Chabacano - Jamaica - Fray Servando - Candelaria o como segunda opción Xola – Viaducto – Chabacano - San Antonio Abad - Pino Suarez – Merced – Candelaria, contando el número de estaciones podemos ver que la primera estación es más corta en cuanto al número de estaciones pero si queremos ahorrar tiempo de viaje, la segunda opción es la más viable (los tiempos que se toman para esta afirmación son los establecidos en nuestro programa).

Si quisiéramos ir por nuestra primera opción, tendríamos que modificar los tiempos de viaje de las estaciones para que sean menores que los de las estaciones de la segunda opción. Para esto necesitaríamos dar click en el botón “Modificar atributos” y después en “Cambiar tiempo de traslado”.

Si quisiéramos definir un nuevo nombre para la estación ermita, sólo tenemos que dar click en el botón “Modificar atributos” y después en “Cambiar nombre” y escribir el nombre de la estación que queremos cambiar y el nuevo nombre y una vez realizado esto, se notificará que la hemos cambiado.

Si decidiéramos eliminar una estación, como portales, estaríamos dejando a Ermita, General Anaya y Taxqueña unidas, pero si quisiéramos llegar a ellas desde cualquier estación diferente a estas tres sería imposible ya que no existe una vía que nos conecte a ellas.

Pero si no queremos eliminar Portales, pero si dejar inalcanzables a Ermita, General Anaya y Taxqueña podemos eliminar la vía entre Portales y Ermita en “Modificar atributos” y “Eliminar Adyacencia”. Si después nos damos cuenta que cometimos un error al eliminar esa adyacencia, podemos volver a ponerla en “Agregar Adyacencia”.

Y si queremos una forma más fácil de llegar de taxqueña a Barranca del Muerto haciendo una escala solamente, podemos agregar una estación entre estas dos.

Formalmente, tenemos una gráfica dirigida y conexa en la que cada estación se representa como un nodo y las vías se representan como aristas; cada nodo tiene los atributos de visitado (inicializado como False), costo (inicializado como “inf”), antecesor, una lista de ‘vecinos’ que contendrá a los nodos que están unidos con una arista a este, nivel y, por último, un id (que será el nombre de nuestra estación).

Esta gráfica se carga de unos archivos que contiene toda la información necesaria de los nodos, los nodos mismos y las aristas, así como los datos para crear el ambiente gráfico en el cuál se creará una ventana y un canvas para dibujar sobre este las líneas y círculos necesarios para dibujar nuestra red del metro de la Cd. De México y en la parte derecha el que será nuestro menú.

Para buscar el camino más corto entre dos nodos es necesario revisar que los nodos ingresados existan y aplicando el algoritmo de BFS lo encontramos, gráficamente deberemos ir cambiando el color de las aristas que marquen el camino, así como imprimir los nombres en un cuadro que creamos del lado de nuestro menú.

Para buscar el camino más corto en tiempo aplicamos los algoritmos de DFS y Dijkstra para encontrarlo y marcamos el camino tal y como se hizo anteriormente, pero se imprimirá el tiempo en minutos que tardaremos en llegar al final de nuestro cuadro.

Para eliminar un nodo n revisamos que este exista iterando en el diccionario de estaciones hasta que el String recibido coincida con un elemento del diccionario, iteramos sobre sus vecinos para encontrar a dos de ellos que tengan el mismo número en el atributo línea y eliminamos al nodo n de la lista de vecinos del par que coincidieron en la línea. Finalmente eliminamos el nodo del diccionario. Y en la gráfica eliminamos el círculo, y con este las aristas.

Para agregar un nodo verificamos que el nodo no exista y que la información sea adecuada según lo solicitado (el tiempo sea una cadena de dígitos), después revisamos si se ingresará el nodo entre dos estaciones o sólo con una conexión a una estación, obtenemos las coordenadas de la estación a la que se agregará la nueva y creamos el círculo y la línea (Para una estación). De otra forma, obtenemos las coordenadas de ambas estaciones, después probamos si estas directamente conectadas, por si la nueva estación estará entre dos estaciones contiguas,

revisamos a que línea pertenecen ambas estaciones y buscamos en esa línea la posición de ambas estaciones, agregamos el nodo, agregamos las aristas, y la dibujamos en el ambiente gráfico el círculo y las dos líneas.

Para modificar los nodos tenemos 4 situaciones:

En cambiar el nombre es necesario buscar el nodo para verificar que exista, entramos a los vecinos del nodo para cambiar el nombre de la estación que se quiere cambiar, es decir, actualizar el nombre. Entramos a los vecinos de los vecinos para cambiar el nombre de la estación a modificar, luego creamos uno nuevo, igualamos ambos nodos y borramos el antiguo.

Para modificar el tiempo verificamos que ambas estaciones existan y que el tiempo ingresado sea una cadena de dígitos, si cumple esto iteramos sobre los vecinos de la primera estación hasta encontrar a la segunda y modificamos el tiempo, además marcamos que hicimos asignando a la variable prueba con True, después verificamos que esta variable sea True y, por tanto, que el primer proceso se ejecutó correctamente y realizamos la misma iteración, pero con la estación 2. Finalmente creamos un cuadro, en el ambiente gráfico, con un mensaje de que el cambió se ha realizado.

Para agregar una adyacencia pedimos la estación y su nueva vecina, así como el tiempo de viaje que existirá entre ellas, verificamos que las dos estaciones existan y llamamos al método agregarArista con la información solicitada, obteneos las coordenadas de ambas estaciones y creamos la línea.

Para eliminar la adyacencia, verificamos que las estaciones existan y después comprobamos que estas estén unidas por una arista y eliminamos las estaciones de los vecinos del otro y eliminamos la arista del gráfico.

El botón salir nos permite finalizar la ejecución, así como decidir si se descartarán o no las modificaciones realizadas de forma definitiva.

Descripción de él o los algoritmos implementados en el proyecto

El recorrido BFS parte de un nodo inicial que será la raíz, luego ve los adyacentes a ese nodo y los agrega en una cola, como la prioridad de una cola es FIFO, los siguientes nodos a evaluar serán los adyacentes previamente insertados. No se pueden visitar 2 veces el mismo nodo, ya que podríamos terminar en un ciclo infinito o simplemente no hallar el punto deseado en el menor número de pasos.

El recorrido DFS visita los vértices de un grafo de forma ordenada, pero no uniforme, es decir, expande todos los vértices que se van visitando en un camino concreto hasta que no haya más vértices que visitar en dicho camino, después regresa de forma recurrente (backtracking), de modo que se repite el paso anterior en cada uno de los vecinos de los vértices que se estén procesando en el camino actual.

El algoritmo dijkstra resuelve un problema por etapas eligiendo la opción óptima local en cada paso con la esperanza de alcanzar la solución óptima global, es decir, encuentra caminos de coste mínimo desde un vértice en un grafo recorriendo un camino desde el vértice s, selecciona el vértice v que tenga la distancia menor entre los vértices no visitados, y declara el camino más corto de s a v con distancia igual a la distancia de del predecesor p más el peso de la arista.

Un ordenamiento topológico es un ordenamiento lineal de todos sus vértices tal que, si el grafo no es acíclico, entonces no es posible un ordenamiento lineal. Puede verse como un ordenamiento de sus vértices a lo largo de una línea horizontal tal que todos los arcos directos van desde la izquierda a la derecha.

El algoritmo de minheap consiste en almacenar todos los elementos del vector a ordenar y luego extraer el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado, dejando siempre en la cima al menor elemento de todos los almacenados en él.

Explicación del código del proyecto

Clase Estacion:

En el constructor se crean las variables que necesitamos y que utilizamos en las prácticas relacionadas a árboles, agregando dos más: una variable booleana transborde y una lista llamada línea.
Posee un solo método que es agregarVecino.

Clase RedMetro:

Esta clase posee los algoritmos necesarios en el proyecto. Los métodos sobresalientes en esta clase son:

- leerRedMetro: Recibe como parámetros dos archivos, donde se encuentran las estaciones de la red y las conexiones entre estas, o sea, las aristas.
- resetEstaciones: Se itera sobre la lista de todos los elementos que tienen como tag “black” y se regresa el color original de la línea. Se

recorren todos los vértices de la red reiniciando los valores de los atributos de cada estación.

- agregarAdy: Agrega una conexión entre cualesquiera dos estaciones que no estén conectadas directamente. Prueba que ambas estaciones existan en la red, si existen, se agrega una arista entre ellos.
  Se obtienen las coordenadas de los óvalos de estas estaciones iterando sobre la lista de ítems que tenga a la estación y con un condicional. Se crea una línea entre ambos óvalos a la que se le asigna como tags ambas estaciones y el color de la línea de la primera estación ingresada.
- eliminarAdy: Prueba que ambas estaciones estén directamente conectadas, de ser así se elimina la arista entre estas, y se elimina la línea que tenga como tags a ambas estaciones.
- Agregar: Se crea un ovalo en el punto medio entre los óvalos de la estación anterior y la posterior a la nueva. Si hay línea entre la anterior y la posterior se elimina y se crean las líneas entre anterior-nueva-anterior.
- modNombreEstacion: se crea una nueva estación con el nombre deseado, se agregan aristas entre la estación con el nombre deseado y los vecinos de la estación con el nombre que se quiere cambiar y se elimina la estación con el nombre antiguo.
- modificarTiempo: se itera sobre los vecinos de la primera estación hasta encontrar la referencia a la segunda estación. Ahí se cambia el peso de la arista por el nuevo valor.
- Agregar: Si la estación nueva solo tendrá un vecino la estación se cea en un espacio aleatorio del canvas, mientras que en el diccionario solo se agrega esta estación y se agregan los vecinos correspondientes. Si se desean dos vecinos, esta se agrega en el punto medio entre estas dos en el canvas.
- Eliminar: elimina una estación de la red. Itera sobre los vecinos de la estación y se van eliminando estas referencias. Finalmente se elimina la estación del diccionario y del canvas.
- caminoMasCortoEstaciones: Calcula el camino más corto en estaciones de la estación *a* a la *b*.
  A *b* se le asigna un nivel 0. Se agrega *b* a una cola donde se marca como visitado. Mientras la cola tenga elementos se extrae el primer elemento y se itera sobre sus vecinos, si este no ha sido visitado se agrega a la cola, se marca como visitado y se le aumenta su nivel en una unidad al elemento extraído de la cola.
  Se agrega la estación *a b* otra lista que guarda el camino a seguir y una variable guarda el valor del nivel de a. Mientras nivel sea diferente de cero se busca en los vecinos de b al elemento con nivel menor en una unidad a la variable nivel, se decrementa nivel y ahora se busca en los vecinos de este vecino.
  Se itera sobre los elementos de esta última lista y se obtienen los tags pertenecientes a ese elemento, al que se le cambia el color de la línea

por “black”. Se hace para resaltar en el canvas el camino a seguir entre las estaciones dadas. Retorna la lista que guarda el camino de *a* a *b.*

- dijkstra: Calcula el camino con menor tiempo de la estación *a* a *b*.
  El costo de *a* será igual a cero y sus vecinos se agregan a una lista al tiempo que se le asigna costo de la arista entre ellos y se marca como su antecesor a *a*. Mientras la lista tenga elementos se extrae el de menor tiempo, se itera sobre sus vecinos y se prueba si esta estación tiene transbordo; si no lo tiene se prueba que el costo de esta estación sea menor al del elemento extraído más el tiempo de la arista, de ser así se se marca a este como su antecesor y se le asigna como costo el tiempo de la suma. Si no se ha visitado se agrega a la cola.
  En caso de tener transbordo, se prueba que tomarTransbordo sea diferente a cero, y se hace lo mismo que si no tuviera transbordo. Si es igual a cero se prueba que el vecino del elemento vecino tenga menor costo que la suma del elemento extraído más el peso de la arista más 5 minutos de transbordo y se marca como antecesor del vecino al elemento extraído. Si no ha sido visitado se agrega a la lista. Finalmente se elimina el elemento extraído de la lista y se marca como visitado.
  Se obtiene el camino a seguir invocando el método camino al que se le mandan como parámetros *b* y una lista donde guarda el camino. Se itera sobre esta lista y a sus elementos se les cambia el tag de color que tienen por “black” para resaltar sobre le canvas el camino a seguir entre *a* y *b*. Retorna la lista que guarda el camino de *a* a *b*.
- tomarTransbordo: en una variable contador se guarda el número de elementos idénticos entre las listas de las líneas a las que pertenece la estación antecesora a la estación que se extrajo de la lista en dijkstra y las líneas a las que pertenece uno de los vecinos de tal estación extraída. Retorna el valor de contador. Si contador es igual a cero, significa que estas estaciones son de diferente línea, por tanto, se tiene que tomar un transbordo en la estación extraído para llegar a su vecina.
- camino: es un método recursivo, en donde el caso base es que la estación recibida tenga un antecesor diferente a menos 1, si se cumple se invoca a camino mandando como parámetro el antecesor a la estación y una lista. Retorna una lista.

<u>Clase RedDelMetro:</u>

Esta clase hereda de la clase RedMetro y es la clase que posee la GUI del proyecto.

- __init__: En su constructor se invoca leerRedMetro, después crea un objeto de la clase Tk donde se crea un Canvas y un Frame. En el Canvas se muestra la red y en el Frame las opciones que el programa posee para manipular la red. Aquí también se invoca el método crearLineas.

- agregarE: prueba que los datos ingresados sean correctos, y de serlo se invoca a agregar.
- eliminarE: prueba que la estación dada sea válida y de ser así se itera sobre sus vecinos y se eliminan las aristas entre estos. Finalmente se elimina del diccionario y del canvas.
- impresionCaminoCorto: se crea una Listbox que se pueda manipular mediante una Scrollbar; en el Listbox se muestran las estaciones a seguir entre las estaciones dadas. Se muestra la lista que retorna el método de caminoMasCortoEstaciones
- impresionCaminoTiempo: igual que el método anterior, pero esta muestra la lista que retorna dijkstra en la que se agrega el tiempo estimado a la llegada a la estación deseada.
- cargarRedNueva: Posee 5 widgets para la lectura delos archivos de los que se desea cargar una red diferente, y un botón con el comando de invocar a cargarR.
- cargarR: se crea un objeto nuevo de la clase RedDelMetro, ahora con los archivos de la red que se desea.
- crearLineas: Abre los archivos recibidos como parámetros que son donde se encuentran las especificaciones de los grafos dibujados sobre en canvas. Se itera sobre las líneas de cada archivo mientras se va creando determinada figura, con las especificaciones leídas, sobre el canvas.
- guardarRed: Si se desea guardar la red se crean los archivos donde se van a guardar los elementos del canvas y las modificaciones al diccionario de la red, guardándolos en diferentes archivos con el formato ideal para una próxima carga de datos.