Estructura del Grafo

Vertice:

En las especificaciones del proyecto, se pide la utilización de TDAs y diccionarios JSON. Tomando esto en cuenta, la clase vertice tendría dos atributos: su nombre y sus aristas en un diccionario. Las aristas se guardarían como el nombre del otro vertice como llave y el peso (ya calculado) como su valor. Para ~~crear una~~ agregar una arista al vertice, se utilizaría una función que reciba el nombre del vertice arista y el peso entre estos dos nodos. La función agregaría los datos en el diccionario (JSON) como se mencionó previamente. La otra función que tendría la clase vertice sería la de obtener las aristas, esto se haría al ingresar cada elemento en un arreglo para tener un mejor acceso para el uso que se le desee dar.

Grafo:

La clase grafo tendría como único atributo el diccionario de vertices que contiene. Ya que cada vertice contiene su diccionario de aristas, en el ~~agregado~~ agregado de vertices se agregaría el ~~nombre~~ nombre del vertice como llave y como valor una instancia de la clase Vertice. Con esta lógica, al agregar una arista con un vertice en especifico, se accesaría el vertice con su llave (el nombre del vertice) en el diccionario de vertices para así poder utilizar sus metodos y atributos.

Para encontra todas las rutas entre dos ~~vert~~ vertices, se debe de mantener un registro de los vertices ya recorridos para evitar ciclos. Luego de investigar sobre el recorrido de grafos, y asimismo acordandose del recorrido de árboles, la mejor opción es la de ~~B Dep~~ "Depth First Search".

Este algoritmo sigue los siguientes pasos (editados para el grafo):

1.) Agregar el vertice actual a los visitados y al camino actual
2.) Si el vertice actual es el que se desea encontrar, ~~retornar~~ agregar el camino actual a la lista de caminos
3.) Recorrer las aristas del vertice actual, en cada iteración:
   i.) Si el vertice arista no ha sido visitado, llamar la función otra vez utilizando el vertice arista actual como vertice actual.
4.) Eliminar el vertice actual de los visitados y el camino actual para seguir revisando otros posibles caminos.
5.) Si la lista del camino actual está vacía, devolver la lista de todos los posibles caminos.

Una vez logrado el algoritmo anterior, se ordenarían de menor a mayor peso para su facil recorrido en la creación de la tabla.

Gui

Boton cargar archivo: este boton se encarga de llamar a la funcion Cargar Archivo

Def Cargar Archivo: Este carga el archivo de texto tanto en el Gui (dialog box) y en consola (para poder usar los datos) para hacer esto primero:

utilizamos una funcion que viene en PyQT5 llamada QFileDialog box. primero usamos una funcion llamada get open file dialog luego en una variable filename guardaremos el archivo cargado en esta variable tambien definimos el Tipo de archivos que admitira el programa en este caso solo necesitamos archivos con la extension .txt

Tabla ASCII

Con la función de obtener los caminos obtendremos ese arreglo y recorrerlo y crear otro arreglo el cual va ir almacenando los elementos como si fuera un texto con sus saltos de linea y sus tabulado para poder regresarlos con un join para que el textEdit de la ventana ya que solo lee cadenas.

* Descripción

- La función de crear una tabla recibira un arreglo el cual sera recorrido y a cada uno de los elementos se le agregara saltos de linea con sus tabulados y hacer su "append" para luego solo unir los con el join y espacios vacios e implementarlo en el text edit.

Clase Loader

→ Metodos:
* Load
* ArrayToGraph
* Weight

• Load :

Recorre un arreglo generado por un archivo de texto plano con el formato

'A':
• 'B':
• • Usuarios: 1
• • Medio: 'WIFI'
• • Distancia: 3
• • Ancho De Banda: 24
• • Trafico: 18
'B':
• 'B':
• • Usuarios: 1
• • medio 'WIFI'
• • Distancia: 3
• • Ancho De Banda: 24
• • Trafico: 18

Donde '•' es un tabulado.
load es una funcion recursiva que contiene como parametros 'array', 'initValue', 'corrent Vertex'

initValue Va recorriendo el arreglo siguiendo un patron, si el initValue del array contiene un ':' entonces es un Vertice si no es una arista y se Va recorriendo sumandole 1 ó 6 para pasar al siguiente Vertice o arista

Luego si es una arista se calcula el peso utilizando [:,:] para copiar los valores numericos y el medio de los 5 elementos que tiene la arista. Luego se genera el edge con el currentVertex.