

Universidad Rafael Landívar

Facultad de ingeniería

Matemática Discreta II

Sección 01

Análisis e implementación del proyecto Practico INGUAT

Heizel Gonzalez 1202101

Contenido

Abstracto.....	3
Introducción	3
Método	3
Metodología de la implementación	3
Python.....	3
Networkx	4
Clase Graph.....	4
Análisis del Algoritmo de Dijkstra.....	5
Algoritmo Dijkstra en NetworkX.....	6
análisis del código Dijkstra en NetworkX	7
implementación ruta más corta en el proyecto	10
Manual del usuario	12
Funciones de administrador	12
Funciones de usuario:	16
Estructura de los archivos del proyecto	18

Abstracto

El objetivo del proyecto es para reactivar el turismo después de la pandemia, por lo que se propone la elaboración de un software basado en los requerimientos:

- Un usuario administrador que permita hacer lo siguiente:
 - Ingresar sitios turísticos
 - Ingresar carreteras a los sitios turísticos
 - Obtener estadísticas de sitios visitados
 - Ingresar alertas en una carretera
 - Eliminar la carretera
 - Ver listado de los sitios turísticos
 - Ver rutas más cortas
- Usuario
 - podrá ver listado de sitios turísticos
 - podrá ver la ruta sugerida o ruta más corta
 - mostrar algún alerta en la carretera

Introducción

Por cuestiones de la pandemia Covid-19 el INGUAT quiere reactivar el turismo interno en Guatemala por lo que necesitan un software que sirva para proponer sitios turísticos mostrando las rutas que mejor se ajustan y los tiempos para realizar los viajes.

Método

Metodología de la implementación

Se utilizó la metodología de Scrum, la cual se crearon diferentes tareas las cuales fueron integradas al proyecto y después se hicieron pruebas para verificar el funcionamiento del proyecto. Las funciones principales están divididas en las siguientes:

- funciones de Administrador
- funciones de Usuarios
- funciones de los grafos
- funciones adicionales como estadísticas.

Python

Es el lenguaje de programación que se escogió para este propósito, Python es un lenguaje que es interprete lo que significa que traduce el código en tiempo real, donde la indentación y el orden del código es crítico para que funcione bien, lo que es ideal para mantener y aprender un estándar en el código, además que tiene funciones y librerías que hacen que la programación y el número de líneas en la implementación disminuya significativamente, ahorrando tiempo en la programación y enfocarse en la funcionalidad del proyecto.



Networkx es una clase de Python que sirve para la implementación de redes, es un paquete para la creación manipulación y el estudio de estructuras, funciones, redes y grafos. Tiene implementados varios algoritmos estándares para el uso de grafos, puede manejar los nodos, y ha sido probado bastante por lo que la clase es confiable en su uso, también beneficia en la creación de los prototipos de redes. Se puede encontrar más información de esta clase en el siguiente link:

<https://networkx.org/>

este es el enlace para ver el código fuente:

<https://github.com/networkx/networkx/tree/main/networkx/classes>

Clase Graph

Networkx provee estructura de datos que permite el manejo de los grafos, la cual permite el uso de un objeto llamado nodo y a este nodo se le puede atribuir un valor llamado Edge que es lo que llamamos la arista. En el proyecto tenemos esta clase declarada como sigue:

```
G = nx.Graph()
```

Para los nodos estos se trabajan como etiquetas, se puede ver en la siguiente parte del código:

```
nx.draw(G,with_labels=True)
```

En el main cargamos los grafos con el siguiente código:

```
def cargaArchivoGrafos():
```

```
    try:
```

```
        file1 = open(archivo_grafos, 'r')
```

```
        Lines = file1.readlines()
```

```
        for line in Lines:
```

```
            x = line.replace('\n','').split(',')
```

```
            if (len(x)>0):
```

```
                G.add_edge(x[0],x[1],weight = int(x[2]))
```

```
    except:
```

```
        print('hay un problema con el archivo de grafos')
```

```
    # salvarLog(usr,'error: existe un problema con el archivo de grafos.')
```

Entonces en la parte remarcada es donde nosotros agregamos el nodo, los parámetros son origen, destino y peso que serían las carreteras.

Para agregar los nodos tenemos la siguiente función:

```
def agregaNodo():  
    try:  
        etiqueta = str.upper(input('Ingrese Nombre del Sitio Turistico:'))  
        G.add_node(etiqueta)  
        addContadorJson(etiqueta)  
        grabarContador()  
    except ValueError:  
        print("No es un dato valido")  
        salvarLog(usr,'error: ha ingresado un dato no valido.')
```

la lógica de esto es que existen nodos, pero estos pueden tener o no carreteras conectadas, por eso se tiene una función para cargar las aristas y los nodos

Enlace del código fuente de la clase grafo:

<https://github.com/networkx/networkx/blob/main/networkx/classes/graph.py>

Análisis del Algoritmo de Dijkstra

El código del algoritmo se puede encontrar en el siguiente enlace:

https://github.com/networkx/networkx/tree/main/networkx/algorithms/shortest_paths

para Dijkstra específicamente este es el enlace:

https://github.com/networkx/networkx/blob/main/networkx/algorithms/shortest_paths/weighted.py

En la documentación de la clase podemos ver las siguientes definiciones:

Parámetros de la clase

- G: que es el grafo en Networkx
- source: que es el nodo origen
- target: que es el nodo destino o final
- weight: este es el peso o una función, en nuestra implementación es una cadena que es un número, además en la documentación interna se especifica que tiene que ser un número entero.

valores de retorno

- Path: el numero de nodos con la ruta mas corta.

Ejemplo:

```
G = nx.path_graph(5)
print(nx.dijkstra_path(G, 0, 4))
[0, 1, 2, 3, 4]
```

Algoritmo Dijkstra en NetworkX

Código completo Dijkstra en NetworkX

```
def _dijkstra_multisource(
    G, sources, weight, pred=None, paths=None, cutoff=None, target=None
):
    G_succ = G._succ if G.is_directed() else G._adj

    push = heappush
    pop = heappop
    dist = {} # dictionary of final distances
    seen = {}

    # fringe is heapq with 3-tuples (distance,c,node)
    # use the count c to avoid comparing nodes (may not be able to)
    c = count()
    fringe = []
    for source in sources:
        seen[source] = 0
        push(fringe, (0, next(c), source))
    while fringe:
        (d, _, v) = pop(fringe)
        if v in dist:
            continue # already searched this node.
        dist[v] = d
        if v == target:
            break
        for u, e in G_succ[v].items():
            cost = weight(v, u, e)
            if cost is None:
                continue
            vu_dist = dist[v] + cost
            if cutoff is not None:
                if vu_dist > cutoff:
                    continue
            if u in dist:
                u_dist = dist[u]
                if vu_dist < u_dist:
                    raise ValueError("Contradictory paths found:", "negative weights?")
```

```

    elif pred is not None and vu_dist == u_dist:
        pred[u].append(v)
    elif u not in seen or vu_dist < seen[u]:
        seen[u] = vu_dist
        push(fringe, (vu_dist, next(c), u))
        if paths is not None:
            paths[u] = paths[v] + [u]
        if pred is not None:
            pred[u] = [v]
    elif vu_dist == seen[u]:
        if pred is not None:
            pred[u].append(v)

# The optional predecessor and path dictionaries can be accessed
# by the caller via the pred and paths objects passed as arguments.
return dist

```

análisis del código Dijkstra en NetworkX

Este es el algoritmo que se ha demostrado en la clase:

```

para todos los vértices  $x \neq a$ 
     $L(x) = \infty$ 
 $T =$  conjunto de todos los vértices
//  $T$  es el conjunto de todos los vértices cuyas distancias más cortas desde  $a$ 
// no se han encontrado
while( $z \in T$ ) {
    seleccionar  $v \in T$  con  $L(v)$  mínimo
     $T = T - \{v\}$ 
    para cada  $x \in T$  adyacente a  $v$ 
         $L(x) = \min\{L(x), L(v) + w(v, x)\}$ 
    }
}

```

En el siguiente análisis se demuestra que este algoritmo es el que se implementó en Python y los tipos de datos que se utilizaron.

La función que realiza la búsqueda de la ruta más corta de Dijkstra es la siguiente función:

```

def_dijkstra_multisource(
    G, sources, weight, pred=None, paths=None, cutoff=None, target=None

```

);

Busca si es dirigido o si tiene nodos sucesores

```
G_succ = G_succ if G.is_directed() else G_adj
```

En esta parte se implementa una cola, Python tiene funciones predefinidas para pilas y colas para más detalles se puede ver el siguiente enlace: <https://docs.python.org/3/library/heapq.html>

push es que inserta un valor en la cola

```
push = heappush
```

pop saca el valor más pequeño de la data

```
pop = heappop
```

En Python los diccionarios son utilizados para guardar data en pares llave: valor, la cual esta ordenada

```
https://www.w3schools.com/python/python\_dictionaries.asp
```

```
dist = {} # dictionary of final distances
```

```
seen = {}
```

acá en el Código se explica que hay una franja que es un arreglo de 3 tuplas, distancia, c y el nodo donde c es un numero para evitar comparar nodos

```
# fringe is heapq with 3-tuples (distance,c,node)
```

```
# use the count c to avoid comparing nodes (may not be able to)
```

```
c = count()
```

```
fringe = []
```

En esta parte del código empieza el ciclo for donde se meten las variables en la cola, en la clase es lo que se muestra como T donde se meten todos los vértices para el análisis

```
for source in sources:
```

```
    seen[source] = 0
```

```
    push(fringe, (0, next(c), source))
```

En esta parte del código entramos en el ciclo que se muestra en la clase mientras (Z pertenece a T)

```
while fringe:
```

En esta parte del código sacamos el menor valor de la cola haciendo un pop

```
(d, _, v) = pop(fringe)
```

Acá se busca si el valor v ya ha sido evaluada si ha sido evaluada sigue:


```
if v in dist:
```

```
    continue # already searched this node.
```

En esta parte se analiza si el sub-nodo tiene el valor del nodo objetivo, que en este caso es target, pero en la clase lo vemos como z

```
    dist[v] = d
```

```
    if v == target:
```

En esta parte del código si ya fue incluido en la cola entonces se cierra el while y no se continua

```
        break
```

En esta parte del código se hace un recorrido y se evalúa el costo del camino, y va haciendo las sumas

```
    for u, e in G_succ[v].items():
```

En esta parte se tiene una función que se llama peso, se envía como parámetros los nodos

```
        cost = weight(v, u, e)
```

Si el costo es ninguno continua, es el equivalente al valor infinito que vimos en clase

```
        if cost is None:
```

```
            continue
```

En esta parte suma los valores de la distancia

```
        vu_dist = dist[v] + cost
```

En este if es si por ejemplo existe un numero máximo de nodos a recorrer

```
        if cutoff is not None:
```

```
            if vu_dist > cutoff:
```

```
                continue
```

```
        if u in dist:
```

```
            u_dist = dist[u]
```

En esta parte del código se verifica si la suma de las distancias es positiva, si no regresa un error

```
        if vu_dist < u_dist:
```

```
            raise ValueError("Contradictory paths found:", "negative weights?")
```

```
        elif pred is not None and vu_dist == u_dist:
```

```
            pred[u].append(v)
```

En esta parte del código se hizo la evaluación de si z ya se ha evaluado, de no ser así se continua, y el nodo evaluado se guarda en la cola

```
elif u not in seen or vu_dist < seen[u]:  
    seen[u] = vu_dist  
    push(fringe, (vu_dist, next(c), u))  
    if paths is not None:  
        paths[u] = paths[v] + [u]
```

En esta parte del Código de evaluar si existe un predecesor si no se agrega el nodo destino sino se le da un append a la cola para la siguiente evaluación

```
if pred is not None:  
    pred[u] = [v]  
elif vu_dist == seen[u]:  
    if pred is not None:  
        pred[u].append(v)
```

```
# The optional predecessor and path dictionaries can be accessed  
# by the caller via the pred and paths objects passed as arguments.  
return dist
```

implementación ruta más corta en el proyecto

La parte del código que implementa la búsqueda de la ruta más corta es la siguiente:

```
def rutaMasCorta():  
    origen = str.upper(input('Ingrese lugar de partida Origen:'))  
    destino = str.upper(input('Ingrese lugar de Destino:'))  
    incrementarContador(destino)  
    try: # esta es una excepción por si la ruta no existe, o el nodo no tiene carretera
```

Es en esta parte del código donde llamamos la función de la ruta mas corta de Dijkstra:

```
path = nx.shortest_path(G, source=origen, target=destino, weight=None, method='dijkstra')  
print ('ruta mas corta: ', path, ' con un total de sitios a recorrer: ', len(path))  
salvarLog(usr, 'info: se ha mostrado la ruta más corta.')  
kms = 0
```

```
G2 = nx.Graph()
```

```
alerta = False;
```

Se realiza un ciclo for para sumar la longitud de los pesos y se evalúa si tiene algún alerta, de tener alerta entonces se muestra en resultado del software.

```
for x in range(len(path)-1):
```

```
    if (G[path[x]][path[x + 1]]["weight"] == VALORBANDERA):
```

```
        print('*****')
```

```
        print('en esta ruta hay una alerta')
```

```
        print('ruta: ',path[x], path[x + 1], ' kilometros: ',G[path[x]][path[x + 1]]["weight"])
```

```
        print('*****')
```

```
    print('ruta: ',path[x], path[x + 1], ' kilometros: ',G[path[x]][path[x + 1]]["weight"])
```

```
    kms = kms + G[path[x]][path[x + 1]]["weight"]
```

```
#sumamos los kilometros con la ruta total
```

```
    G2.add_edge(path[x],path[x + 1],weight = G[path[x]][path[x + 1]]["weight"])
```

```
print('kilometros totales a recorrer: ',kms)
```

```
salvarLog(usr,'info: se han mostrado los kilometros a recorrer.')
```

```
#mostramos la ruta mas corta
```

```
nx.draw(G2,with_labels=True)
```

```
plt.savefig("graph2.png")
```

```
plt.show()
```

```
except nx.exception.NetworkXNoPath:
```

```
    print('no existe ruta entre: ', origen, ' ',destino)
```

Manual del usuario

Funciones de administrador

Para poder correr el software es importante instalar python, e instalar las librerías para compilación de c como se muestra en el siguiente enlace:

<https://visualstudio.microsoft.com/visual-cpp-build-tools/>

Se recomienda instalar Visual Studio Code, y darle click en donde dice c++ Extension:




Visual Studio Code

Lightweight, extensible code editor with a
C++ Extension.




[Learn more >](#)

Al hacer click en c++ Extension aparece lo siguiente:

<https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools>



C/C++

Microsoft  |  25,299,122 installs |  (456) | Free

C/C++ IntelliSense, debugging, and code browsing.

[Install](#) [Trouble Installing?](#)

Se le da instalar, se escoge visual studio para instalar la Liberia después de haber realizado la instalación se abre visual studio y en la terminal se corren los siguientes comandos:

```
#pip install networkx
#pip install matplotlib
```

Esto es para instalar las extensiones de python para que se puedan correr los grafos.

Después cuando se esta corriendo el script de python, se ingresa el usuario, el usuario administrador tiene las siguientes credenciales:

Usuario: root

Contraseña: toor

Al ingresar aparece el siguiente menú:

```
Ingrese usuario:root
Ingrese password:toor
1. crear usuario
2. Agregar sitio turistico
3. Agregar carreteras
4. remover carretera
5. agregar alertas
6. log de estadisticas sitios turisticos
7. ver sitios y carreteras ingresados
8. ruta mas corta
10. salir
Ingrese Opcion:
```

Para crear usuario presiona el numero 1 y luego enter, cuando se ha creado exitosamente aparece el siguiente mensaje:

```
Ingrese Opcion:1
Ingrese nonbre del usuario:pearljam
Ingrese password:evenflow
El usuario no se ha encontrado
El usuario pearljam fue creado
grabando usuarios cuando hay algun cambio
```

La opción 2 que es agregar un sitio turístico es para crear la etiqueta del nodo, téngase en cuenta que no se tiene una arista a algún nodo.

```
Ingrese Opcion:2
Ingrese Nombre del Sitio Turistico:Guatemala
grabando contador
salvando grafos
```

La opción 3 sirve para crear carreteras entre los nodos:

```
Ingrese Opcion:3
Ingrese sitio origen:Guatemala
Ingrese sitio destino:solola
Ingrese la distancia en kilometros:200
salvando grafos
salvando grafos
```

La opción 4 sirve para remover carreteras, por si se necesitara eliminar alguna ruta que no esté habilitada

```
Ingrese Opcion:4
Ingrese Nodo Origen:guatemala
Ingrese Nodo Destino:solola
```

La opción 5 agrega una alerta en la carretera, el alerta sirve para mostrar que existe algún inconveniente en la carretera, por lo que se le pone una cantidad grande al peso de la carretera, si existe alguna ruta alterna entonces va a buscar la ruta alterna mas corta, si no existe ruta alterna entonces va a mostrar la carretera pero con el alerta.

```
Ingrese Opcion:5
Ingrese sitio origen:guatemala
Ingrese sitio destino:solola
salvando grafos
salvando grafos
```

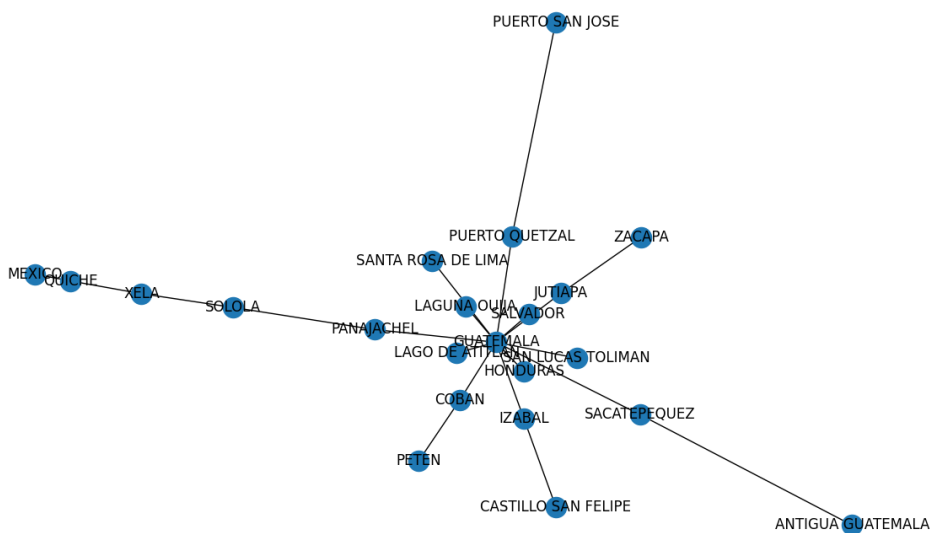
La opción 6 saca un log y estadísticas de los sitios turísticos mas visitados:

```
sitio: LAGUNA OUIJA      numero de consultas : 0
sitio: PUERTO QUETZAL    numero de consultas : 0
sitio: SAN LUCAS TOLIMAN numero de consultas : 0
sitio: SANTA ROSA DE LIMA numero de consultas : 0
sitio: SOLOLA            numero de consultas : 0
sitio: XELA              numero de consultas : 0
sitio: QUICHE            numero de consultas : 0
sitio: MEXICO             numero de consultas : 0
sitio: ZACAPA            numero de consultas : 0
sitio: PETEN             numero de consultas : 0
sitio: CASTILLO SAN FELIPE numero de consultas : 0
sitio: ANTIGUA GUATEMALA numero de consultas : 0
sitio: PUERTO QUETZAL    numero de consultas : 0
sitio: PUERTO SAN JOSE   numero de consultas : 0
```

La opción 7 muestra las carreteras que hay en el grafo mostrando los registros:

```
Ingrese Opcion:7
[('GUATEMALA', 'LAGO DE ATITLAN', {'weight': 450}), ('GUATEMALA', 'SALVADOR', {'weight': 300}), ('GUATEMALA', 'HONDURAS', {'weight': 350}), ('GUATEMALA', 'PANAJACHEL', {'weight': 150}), ('GUATEMALA', 'JUTIAPA', {'weight': 260}), ('GUATEMALA', 'COBAN', {'weight': 428}), ('GUATEMALA', 'IZABAL', {'weight': 265}), ('GUATEMALA', 'SACATEPEQUEZ', {'weight': 58}), ('GUATEMALA', 'LAGUNA QUIJA', {'weight': 255}), ('GUATEMALA', 'PUERTO QUETZAL', {'weight': 106}), ('GUATEMALA', 'SAN LUCAS TOLIMAN', {'weight': 149}), ('GUATEMALA', 'SANTA ROSA DE LIMA', {'weight': 73}), ('PANAJACHEL', 'SOLOLA', {'weight': 73}), ('JUTIAPA', 'ZACAPA', {'weight': 93}), ('COBAN', 'PETEN', {'weight': 158}), ('IZABAL', 'CASTILLO SAN FELIPE', {'weight': 76}), ('SACATEPEQUEZ', 'ANTIGUA GUATEMALA', {'weight': 9}), ('PUERTO QUETZAL', 'PUERTO SAN JOSE', {'weight': 10}), ('SOLOLA', 'XELA', {'weight': 150}), ('XELA', 'QUICHE', {'weight': 150}), ('QUICHE', 'MEXICO', {'weight': 700})]
[]
```

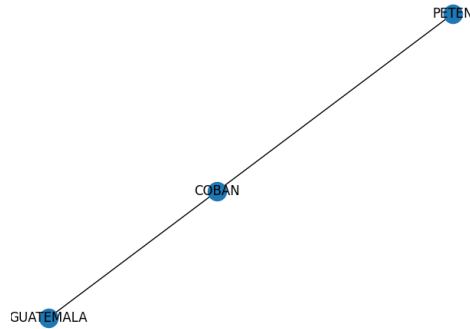
Mostrando el siguiente grafo con las rutas:



La opción 8 muestra la ruta mas corta, se pone el origen, destino y automáticamente muestra la ruta:

```
Ingrese Opcion:8
Ingrese lugar de partida Origen:guatemala
Ingrese lugar de Destino:peten
encontrado
[{'sitio': 'GUATEMALA', 'contador': 0}, {'sitio': 'LAGO DE ATITLAN', 'contador': 0}, {'sitio': 'PANAJACHEL', 'contador': 0}, {'sitio': 'JUTIAPA', 'contador': 0}, {'sitio': 'COBAN', 'contador': 0}, {'sitio': 'LAGUNA QUIJA', 'contador': 0}, {'sitio': 'PUERTO QUETZAL', 'contador': 0}, {'sitio': 'SOLOLA', 'contador': 0}, {'sitio': 'XELA', 'contador': 0}, {'sitio': 'QUICHE', 'contador': 0}, {'sitio': 'PETEN', 'contador': 1}, {'sitio': 'CASTILLO SAN FELIPE', 'contador': 0}, {'sitio': 'PUERTO SAN JOSE', 'contador': 0}]
ruta mas corta: ['GUATEMALA', 'COBAN', 'PETEN'] con un total de sitios a recorrer: 3
ruta: GUATEMALA COBAN kilometros: 428
ruta: COBAN PETEN kilometros: 158
kilometros totales a recorrer: 586
```

Y también muestra un grafo de la ruta más corta al destino



Funciones de usuario:

Se hace login con el usuario y se muestra el siguiente menú:

```

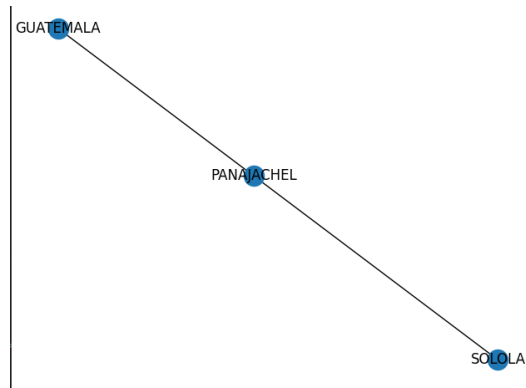
Ingrese usuario:heizel
Ingrese password:password
encontrado
1. obtener ruta del turistico a visitar
2. obtener ruta mas corta
4. ver todas las rutas
5. salir
Ingrese Opcion:
  
```

En la opción 1 se escoge la ruta del sitio turístico a visitar se solicita el lugar turístico y verifica si existe en el grafo de ser así pregunta el origen y el destino

```

Ingrese Opcion:1
Ingrese Nombre del Sitio Turistico:guatemala
buscando sitio turistico GUATEMALA
encontrado
Ingrese lugar de partida Origen:solola
Ingrese lugar de Destino:guatemala
encontrado
[{'sitio': 'GUATEMALA', 'contador': 1}, {'sitio': 'LAGO DE ATITLAN', 'contador': 0}, {'sitio': 'COBAN', 'contador': 0}, {'sitio': 'IZABAL', 'contador': 0}, {'sitio': 'SANTA ROSA DE LIMA', 'contador': 0}, {'sitio': 'SOLOLA', 'contador': 1}, {'sitio': 'CASTILLO SAN FELIPE', 'contador': 0}, {'sitio': 'GUATEMALA', 'contador': 1}]
ruta mas corta: ['SOLOLA', 'PANAJACHEL', 'GUATEMALA'] con un total de 223 kilometros
ruta: SOLOLA PANAJACHEL kilometros: 73
ruta: PANAJACHEL GUATEMALA kilometros: 150
kilometros totales a recorrer: 223
  
```

Y muestra un grafo de la ruta



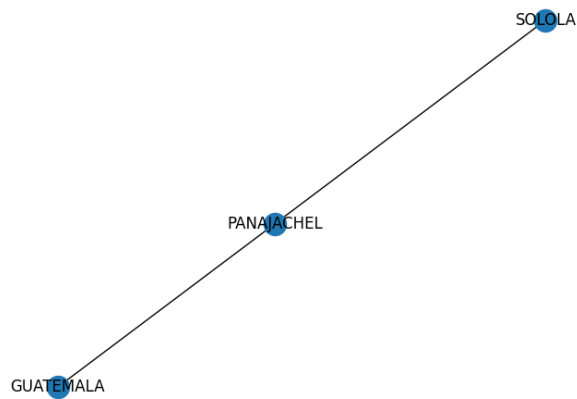
La opción 2 muestra la ruta mas corta solo tiene que poner origen y destino

```

5. salir
Ingrese Opcion:2
Ingrese lugar de partida Origen:guatemala
Ingrese lugar de Destino:solola
encontrado
[{'sitio': 'GUATEMALA', 'contador': 1}, {'sitio': 'LAGO DE ATITLAN', 'contador': 0}, {'sitio': 'PANAJACHEL', 'contador': 0}, {'sitio': 'JUTIAPA', 'contador': 0}, {'sitio': 'COBAN', 'contador': 0}, {'sitio': 'LAGUNA OUIJA', 'contador': 0}, {'sitio': 'PUERTO QUETZAL', 'contador': 0}, {'sitio': 'SOLOLA', 'contador': 1}, {'sitio': 'XELA', 'contador': 0}, {'sitio': 'QUICHE', 'contador': 0}, {'sitio': 'PETEN', 'contador': 1}, {'sitio': 'CASTILLO SAN FELIPE', 'contador': 0}, {'sitio': 'PUERTO SAN JOSE', 'contador': 0}]
ruta mas corta: ['GUATEMALA', 'PANAJACHEL', 'SOLOLA'] con un total de sitios a recorrer: 3
ruta: GUATEMALA PANAJACHEL kilometros: 150
ruta: PANAJACHEL SOLOLA kilometros: 73
kilometros totales a recorrer: 223

```

Y también muestra un grafo:



Estructura de los archivos del proyecto

Archivo grafos:

Guarda la información de los lugares turísticos, el nombre del archivo es grafos.csv y cada registro tiene el siguiente formato:

Origen,destino,peso

Donde peso son los kilómetros.

Archivo de contador de estadísticas

El archivo que guarda las estadísticas de los lugares turísticos en formato de json, y se ve de la siguiente forma:

```
[{"sitio": "GUATEMALA", "contador": 0}, {"sitio": "LAGO DE ATITLAN", "contador": 0}]
```

Tiene un campo llamado sitio, un campo contador, que se va incrementando mientras se agregan las consultas, el nombre del archivo es contador.json

Archivo de contador de usuarios

El archivo para usuarios se llama usuarios.json

```
[{"usuario": "root", "password": "toor", "tipo": "0"}, {"usuario": "heizel", "password": "password", "tipo": "1"}]
```

Tiene el campo usuario que es el nombre del usuario con el que se conecta, el campo password que es la contraseña, y el tipo de usuario, si es 0 es administrador y si es 1 es usuario de consulta.