Práctica de Estudio 7: **Grafos**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Jesus Cruz Navarro
Profesor: _____

Estructura de Datos y Algoritmos II
Asianatura: _____

01
Grupo: _____

07
No de Práctica(s): _____

Diego Santiago Gutierrez
Integrante(s): _____

No. de Equipo de cómputo _____

No. de Lista o Brigada: _____

Tercer Semestre
Semestre: _____

18/11/2020
Fecha de entrega: _____

Observaciones: _____

_____

CALIFICACIÓN: _____

Práctica de Estudio 7: **Grafos**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

REQUISITOS:

a) **Diseñar e implementar** las clases **Vértice** y **Grafo**, con los métodos AgregarVertice y AgregarArista, BreadthFirstSearch, y DepthFirstSearch como se vio en clase (usando como parámetros los nombres de los vértices, en lugar de pasar un objeto de tipo Vértice) y modificar la clase vértice para poder utilizar los algoritmos BFS y DFS.

b) **Diseñar e implementar** el método EncontrarCaminoBFS(nombreVerticeInicial, nombreVerticeFinal) que, utilizando el algoritmo Breadth Firsth Search. IMPRIMA los nombres de los vértices del camino que hay que seguir para llegar desde un vértice Inicial a un vértice Final. Esto se realiza iterando sobre los vértices Padre del nodo final, hasta que un vértice padre sea nulo. Además, imprima el número de estaciones de la ruta.
NOTA: Debe imprimir desde inicial al final (no al revés), incluyendo los vértices inicial y final.

c) **Diseñar e implementar** el método EncontrarCaminoDFS(nombreVerticeInicial, nombreVerticeFinal) que, utilizando el algoritmo Depth First Search (el pseudocódigo lo pueden encontrar en la práctica de la coordinación), IMPRIMA los nombres de los vértices del camino que hay que seguir para llegar desde un vértice Inicial a un vértice Final. Esto se realiza iterando sobre los vértices Padre del nodo final, hasta que un vértice padre sea nulo. Además, imprima el número de estaciones de la ruta.
NOTA: Debe imprimir desde inicial al final (no al revés), incluyendo los vértices inicial y final.

d) Desarrolle un programa que **genere un grafo con todas las líneas del Metro** (una lista con todas las estaciones de metro se encuentra en el archivo metro.py). Las estaciones serán los nodos del grafo y la interconexión entre cada estación con otras serán los vértices del grafo.

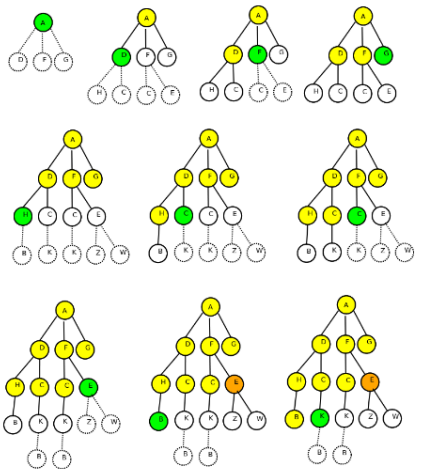
e) Imprima para los **Caminos encontrados con BFS y DFS** para los siguientes trayectos:
   a. Aquiles Serdán – Iztapalapa
   b. San Antonio – Aragón
   c. Vallejo - insurgentes

Nota 1: Para este ejercicio, es improbable que salgan la misma ruta entre las mismas estaciones usando DPS y BFS, sobre todo en los trayectos planteados. Depende de su implementación, como hayan agregado las líneas y el trayecto en particular (si hay transbordos, si son líneas adyacentes, etc.), para que den la misma ruta. Pongan en las conclusiones cual creen que es el patrón que sigue DPS para generar los caminos (a diferencia de BFS, que debe dar la ruta más corta).

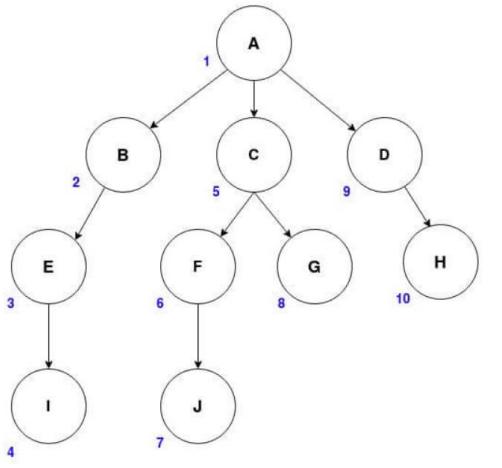
Práctica de Estudio 7: **Grafos**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

INTRODUCCIÓN:

Búsqueda en Anchura (BFS): La idea principal consiste en visitar todos los nodos que hay a profundidad i antes de pasar a visitar aquellos que hay a profundidad i+1. Es decir, tras visitar un nodo, pasamos a visitar a sus hermanos antes que a sus hijos



Búsqueda en Profundidad (DFS): Al igual que en el caso de la búsqueda en anchura,) también puede ser vista como un proceso por niveles, pero con la diferencia de que, tras visitar un nodo, se visitan sus hijos antes que sus hermanos, por lo que el algoritmo tiende a bajar por las ramas del árbol hacia las hojas antes de visitar cada una de las ramas posibles.



Ambas técnicas constituyen métodos sistemáticos para visitar todos los vértices y arcos del grafo, exactamente una vez y en un orden específico predeterminado, por lo cual podríamos decir que estos algoritmos simplemente nos permiten hacer recorridos controlados dentro del grafo con algún propósito.

DESARROLLO:

a)

```python
class Vertex: #Creamos la clase Vertex
    def __init__(self, name): #Constructor que recibe el nombre
        self.name=name #Nombre
        self.neighbors = [] #Vecinos
        self.parentNode  = None #Padre de los nodos
        self.color = "" #Color es el estado (Visitado, proceso, no visitado)
        self.distance = None #Distancia del nodo de un punto a otro

    def __str__(self): #Metodos de impresion str
        return str(self.name)

    def __repr__(self): #Metodo de impresion repr objetos
        return str(self.name)

    def addNeighbor(self, nameVertex): #Creamos metodo que añade un nuevo vecino
                                       #vertex es el nombre

        if nameVertex in self.neighbors: #Si ya existe el nodo, indica el vertice
            print('Vecino ya existe', nameVertex.name , "En el vertice: " , self.name )
            return

        self.neighbors.append(nameVertex) #Si no existe se incerta el vecino en la lista
        return
```

Clase Vertex:

La clase tiene sus atributos, y un metodo que agrega los vecinos a una lista.

```python
from Vertex import Vertex #Exportamos el Vertex

class Grafo: #Creamos la clase Grafo

    def __init__(self): #Atributos de la clase···

    def __str__(self): #Metdo para imprimir como cadena

    def __repr__(self): #Metodo para imprimir objetos

    def addVertex(self, nameVertex): #Metodo que añade

    def addBranch(self, nameVertex1, nameVertex2):#Meto

    def BFS (self, nameFirstNodo): #Metodo BFS que rec

    def RutaBFS(self, InitialNode, FinalNode): #Para c

    def DFS (self, nameFirstNodo): #Metodo de busqueda

    def DFSearch(self, actualNode):···

    def RutaDFS(self, InitialNode, FinalNode ): #Buscar
```

Clase Grafo

```python
def addVertex(self, nameVertex): #Metodo que añade un nodo
    if nameVertex in self.vertexs: #Verifica si el vertice ya existe en el diccionario de vertices
        print('Ya existe: ', nameVertex)
        return False

    self.vertexs[nameVertex] = Vertex(nameVertex)
    #Si no se cumple el if, se crea un vertex
    #tipo Vertex que recive el nombre del nodo
    #Crea una key con nameVertex y un value tipo Vertex con su nombre
    #{nameVertex: Vertex}
    return
```

Metodo addVertex

Este metodo agregará nodos, de forma que primero compará si existen (ya que no pueden repetirse), de no ser así se añaden.

```python
def addBranch(self, nameVertex1, nameVertex2):#Metodo que añade arista

    if not nameVertex1 in self.vertexs: #Si no existe el nodo 1
        print('Error al agregar arista. No existe le vertice 1', nameVertex1)
        return False

    if not nameVertex2 in self.vertexs: #Si no existe el nodo 2
        print('Error al agregar arista. No existe el vertice 2', nameVertex2)
        return False

    self.vertexs[nameVertex1].addNeighbor(self.vertexs[nameVertex2]) #A es vecino de B
    self.vertexs[nameVertex2].addNeighbor(self.vertexs[nameVertex1]) #B es vecino de A
    return
```

Metodo addBranch

Agregar arista conciste en que si existen los dos nodos a los que queremos agregar, entonces consideramos a estos como vecinos.

```python
def BFS (self, nameFirstNodo): #Metodo BFS que recibe un nodo para buscar
    for u in self.vertexs.values(): #Reseteamos todo el nodo, nuevecito queda
        u.color = "blanco"
        u.distance = None
        u.parentNode = None;

    if nameFirstNodo not in self.vertexs.keys(): #Verificamos si el nodo existe con su key del diccionario
        print("El nodo no es parte del arreglo") #Si no existe le do decimos
        return

    nodeInitial = self.vertexs[nameFirstNodo] #Key->NameFirstNode , value->nodeInitial
    nodeInitial.color = "gris" #Marcamos como en proceso
    nodeInitial.distance  = 0 #La distancia del primer nodo es 0

    q = list() #Creamos una lista que será utilizada como cola
    q.append(nodeInitial) #Insertamos en la cola el nodoInicial

    while (len(q) > 0): #Mientras este lleno la lista
        actualNode = q.pop(0) #Se saca el inicial, dando comportamiento de cola
        #actualNode = u
        #explorerNode = v
        for v in actualNode.neighbors: #Actual nodo es el primer elemento de la cola
                                       #Buscamos en sus vecinos

            if v.color == "blanco": #Si nodo vecino no ha sido explorado
                v.color = "gris" #Ya que ha sido explorado ahora es gris
                v.distance = actualNode.distance + 1 #Su distancia es uno mas que el padre
                v.parentNode = actualNode.name #Su padre será el nombre del nodo actual

                q.append(v)  #Insertamos a v en la cola

        self.vertexs[actualNode.name].color = "negro" #Volvemos negro el nodo una vez que haya sido explorado
```

Metodo BFS

Metodo BFS es un algoritmo que buscará por niveles, para ello inicia todos los atributos que necesitará en modo restored. Despues verificará si se encuentra el nodo que va a buscar, para posteriormente crear una lista que funcionará como cola, ya que irá sacando los primeros elementos de esta (FIFO) para poder ser comparados junto a sus vecinos, recorriendo capa por capa y finalmente, volviendo un nodo explorado si esto se ha completado.

```python
def DFS (self, nameFirstNodo): #Metodo de busqueda DFS
    for u in self.vertexs.values(): #Reseteamos todos los valores del diccionario
        u.color= "blanco"
        u.parentNode = None
        u.distance = None

    if nameFirstNodo not in self.vertexs.keys(): #Si no se encuentra el nombre en el diccionario
        print("No esta en el arreglo")
        return

    actualNode = self.vertexs[nameFirstNodo] #El nodo actual será el que estemos buscando
    actualNode.color = "gris" #Como sabemos que estamos ahí, lo volvemos gris
    actualNode.distance = 0  #Su distancia es 0 por ser el inicial
    self.DFSearch(actualNode) #Buscamos el camino con una función aparte
```

Metodo DFS

El metodo inicia reseteando todos los valores de los nodos, con el fin de recorrerlo en caso de algún problema. Después de realizar esto verificará que el arreglo se encuentre dentro del diccionario de vertexs, de ser así el nodo actual será el nombre del nodo que estemos buscando será por ende, un nodo explorado (gris), donde inicia un **DFSearch** sobre este

```python
def DFSearch(self, actualNode):

    for v in actualNode.neighbors:  #Recorremos en los vecinos del nodo actual
        if v.color == "blanco": #Si no han sido explorados
            v.parentNode = actualNode.name #El primer nodo vecino será el nombre del actual
            v.color = "gris" #Su color será gris ya que estando en el se explora
            v.distance = actualNode.distance + 1 #La distnacia será la del nodo actual más uno
            self.DFSearch(v) #Buscamos en el primer vecino del vecino de ahorita hasta que ya no haya mas

    actualNode.color= "negro" #El nodo actual se vuelve negro una vez acabando de explorar
    return
```

Metodo DFSearch

Este metodo es recursivo, es decir que se llama así mismo, tan necesario como sea, hasta llegar a una exploracion total. Lo que hace es que busca en los vecinos, si estos no han sido explorados asignamos al padre ahora como nodo a explorar, aumentamos la distancia en uno y actualizamos su estado a gris. Es aquí cuando DFSearch se llama así mismo y realizará esto hasta que llegue al final de un nodo, volviendo a un punto anterior y empezar a marcar como negro los nodos ya explorados, hasta ir con el siguiente vecino y repetir el proceso.

b)

```python
def RutaBFS(self, InitialNode, FinalNode): #Para conocer la ruta necesitamos de donde empezar
                                           #Y donde terminar

    self.BFS(InitialNode) #Hacemos BFC con el nodo inicial
    lines = list() #Creamos una lista
    actualNode = self.vertexs[FinalNode] #Nodo actual será el final de la busqueda

    while True:
        lines.append(actualNode.name) #Incertaremos en la lista el nombre del nodo
        if actualNode.parentNode == None: #Si llegamos None (sin hijos ya)
            break #Salimos
        actualNode = self.vertexs[actualNode.parentNode] #Pasamos al nodo padre de este
        #{padre: actualNode}

    print("------------------R U T A - B F S-----------------------------")
    print("\tPARADA: ", InitialNode, "\tDESTINO: ", FinalNode)
    print("\t\tTOTAL DE ESTACIONES: ", len(lines) )

    cadena = lines[len(lines)-1]
    for i in range(len(lines)-1, -1, -1):
        cadena += "-->" + lines[i]
    print("MOSTRANDO RUTA: ", cadena )
```

Metodo RutaBFS

Para realizar esta ruta, es necesario un punto de partida y un punto de llegada, despues de realizar una busqueda BFS del nodo incial, creamos una lista para ir almacenando los caminos que se recorrieron hasta llegar al destino. Empezamos de la ruta final a la ruta destino por fines prácticos y hasta que while deje de ser cierto, se acabará dicha busqueda e imprimios resultados de la busqueda.

c)

```python
def RutaDFS(self, InitialNode, FinalNode ): #Buscamos la ruta inicial a un punto final

    self.DFS(InitialNode) #LBuscamos con DFS el camino del nodo inicial
    lines = list()  #Creamos una lista que será usada como pila
    ActualNode = self.vertexs[FinalNode] #El nodo punto final será nuestro nodo actual

    while True: #Mientras haya elementos que incertar
        lines.append(ActualNode.name) #Incertamos el nodoACtual(final)
        if ActualNode.parentNode == None: #Si se llega al padre que apunta a null, salimos
            break
        ActualNode = self.vertexs[ActualNode.parentNode] #Hasta entonces buscaremos en el padre del padre del padre
        #{ActualNode.parentNode : actualNode}
    print("------------------R U T A - D F S-----------------------------")
    print("\tPARADA: ", InitialNode, "\tDESTINO: ", FinalNode)
    print("\t\tTOTAL DE ESTACIONES: ", len(lines) )

    cadena = lines[len(lines)-1]
    for i in range(len(lines)-1, -1, -1):
        cadena += "-->" + lines[i]
    print("MOSTRANDO RUTA: ", cadena )
```

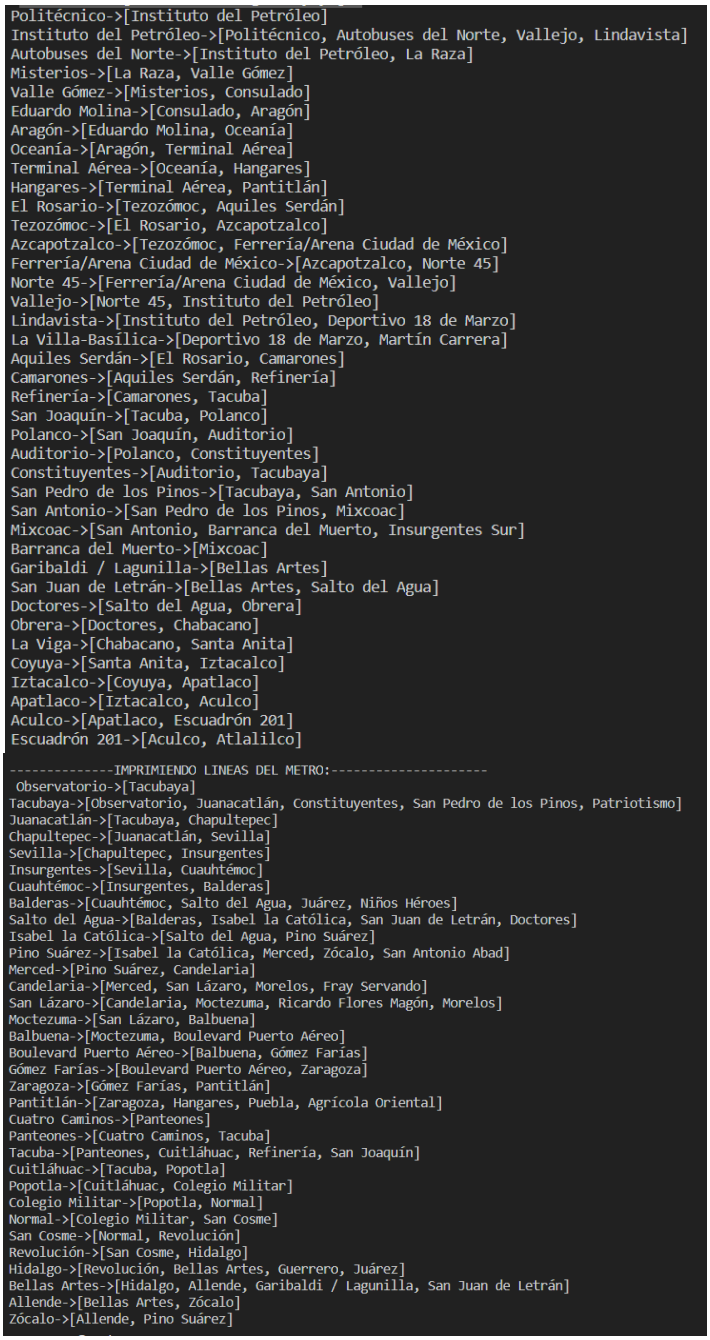
Práctica de Estudio 7: **Grafos**
Estructura de Datos y Algoritmos Grupo 01
Facultad de Ingeniería
Departamento de Computación

RutaDFS

Para realizar esta ruta, es necesario un punto de partida y un punto de llegada, despues de realizar una busqueda DFS del nodo incial, creamos una lista para ir almacenando los caminos que se recorrieron hasta llegar al destino. Empezamos de la ruta final a la ruta destino por fines prácticos y hasta que while deje de ser cierto, se acabará dicha busqueda e imprimios resultados de la busqueda. Lo mismo que BFS,

RESULTADOS:

d)



```
San Antonio Abad->[Pino Suárez, Chabacano]
Chabacano->[San Antonio Abad, Viaducto, Obrera, La Viga, Lázaro Cárdenas, Jamaica]
Viaducto->[Chabacano, Xola]
Xola->[Viaducto, Villa de Cortés]
Villa de Cortés->[Xola, Nativitas]
Nativitas->[Villa de Cortés, Portales]
Portales->[Nativitas, Ermita]
Ermita->[Portales, General Anaya, Eje Central, Mexicaltzingo]
General Anaya->[Ermita, Tasqueña]
Tasqueña->[General Anaya]
Indios Verdes->[Deportivo 18 de Marzo]
Deportivo 18 de Marzo->[Indios Verdes, Potrero, Lindavista, La Villa-Basílica]
Potrero->[Deportivo 18 de Marzo, La Raza]
La Raza->[Potrero, Tlatelolco, Autobuses del Norte, Misterios]
Tlatelolco->[La Raza, Guerrero]
Guerrero->[Tlatelolco, Hidalgo, Garibaldi, Buenavista]
Juárez->[Hidalgo, Balderas]
Niños Héroes->[Balderas, Hospital General]
Hospital General->[Niños Héroes, Centro Médico]
Centro Médico->[Hospital General, Etiopía/Plaza de la Transparencia, Chilpancingo, Lázaro Cárdenas]
Etiopía/Plaza de la Transparencia->[Centro Médico, Eugenia]
Eugenia->[Etiopía/Plaza de la Transparencia, División del Norte]
División del Norte->[Eugenia, Zapata]
Zapata->[División del Norte, Coyoacán, 20 de Noviembre, Parque de los Venados]
Coyoacán->[Zapata, Viveros/Derechos Humanos]
Viveros/Derechos Humanos->[Coyoacán, Miguel Ángel de Quevedo]
Miguel Ángel de Quevedo->[Viveros/Derechos Humanos, Copilco]
Copilco->[Miguel Ángel de Quevedo, Universidad]
Universidad->[Copilco]
Martín Carrera->[Talismán, La Villa-Basílica]
Talismán->[Martín Carrera, Bondojito]
Bondojito->[Talismán, Consulado]
Consulado->[Bondojito, Canal del Norte, Valle Gómez, Eduardo Molina]
Canal del Norte->[Consulado, Morelos]
Morelos->[Canal del Norte, Candelaria, San Lázaro, Tepito]
Fray Servando->[Candelaria, Jamaica]
Jamaica->[Fray Servando, Santa Anita, Chabacano, Mixiuhca]
Santa Anita->[Jamaica, La Viga, Coyuya]
```

```
Atlalilco->[Escuadrón 201, Iztapalapa, Mexicaltzingo, Culhuacán]
Iztapalapa->[Atlalilco, Cerro de la Estrella]
Cerro de la Estrella->[Iztapalapa, UAM-I]
UAM-I->[Cerro de la Estrella, Constitución de 1917]
Constitución de 1917->[UAM-I]
Patriotismo->[Tacubaya, Chilpancingo]
Chilpancingo->[Patriotismo, Centro Médico]
Lázaro Cárdenas->[Centro Médico, Chabacano]
Mixiuhca->[Jamaica, Velódromo]
Velódromo->[Mixiuhca, Ciudad Deportiva]
Ciudad Deportiva->[Velódromo, Puebla]
Puebla->[Ciudad Deportiva, Pantitlán]
Agrícola Oriental->[Pantitlán, Canal de San Juan]
Canal de San Juan->[Agrícola Oriental, Tepalcates]
Tepalcates->[Canal de San Juan, Guelatao]
Guelatao->[Tepalcates, Peñón Viejo]
Peñón Viejo->[Guelatao, Acatitla]
Acatitla->[Peñón Viejo, Santa Marta]
Santa Marta->[Acatitla, Los Reyes]
Los Reyes->[Santa Marta, La Paz]
La Paz->[Los Reyes]
Ciudad Azteca->[Plaza Aragón]
Plaza Aragón->[Ciudad Azteca, Olímpica]
Olímpica->[Plaza Aragón, Ecatepec]
Ecatepec->[Olímpica, Múzquiz]
Múzquiz->[Ecatepec, Río de los Remedios]
Río de los Remedios->[Múzquiz, Impulsora]
Impulsora->[Río de los Remedios, Nezahualcóyotl]
Nezahualcóyotl->[Impulsora, Villa de Aragón]
Villa de Aragón->[Nezahualcóyotl, Bosque de Aragón]
Bosque de Aragón->[Villa de Aragón, Deportivo Oceanía]
Deportivo Oceanía->[Bosque de Aragón, Romero Rubio]
Romero Rubio->[Deportivo Oceanía, Ricardo Flores Magón]
Ricardo Flores Magón->[Romero Rubio, San Lázaro]
Tepito->[Morelos, Lagunilla]
Lagunilla->[Tepito, Garibaldi]
```

```
Garibaldi->[Lagunilla, Guerrero]
Buenavista->[Guerrero]
Insurgentes Sur->[Mixcoac, 20 de Noviembre]
20 de Noviembre->[Insurgentes Sur, Zapata]
Parque de los Venados->[Zapata, Eje Central]
Eje Central->[Parque de los Venados, Ermita]
Mexicaltzingo->[Ermita, Atlalilco]
Culhuacán->[Atlalilco, San Andrés Tomatlán]
San Andrés Tomatlán->[Culhuacán, Lomas Estrella]
Lomas Estrella->[San Andrés Tomatlán, Calle 11]
Calle 11->[Lomas Estrella, Periférico Oriente]
Periférico Oriente->[Calle 11, Tezonco]
Tezonco->[Periférico Oriente, Olivos]
Olivos->[Tezonco, Nopalera]
Nopalera->[Olivos, Zapotitlán]
Zapotitlán->[Nopalera, Tlaltenco]
Tlaltenco->[Zapotitlán, Tláhuac]
Tláhuac->[Tlaltenco]
```

e)

```
----------R U T A - B F S----------
      PARADA: Vallejo      DESTINO: Insurgentes
            TOTAL DE ESTACIONES: 11
MOSTRANDO RUTA: Vallejo-->Vallejo-->Instituto del Petróleo-->Autobuses del Norte-->La Raza-->Tlatelolco-->Guerrero-->Hidalgo-->Juárez-->Balderas-->Cuauhtémoc-->Insurgentes
----------R U T A - D F S----------
      PARADA: Vallejo      DESTINO: Insurgentes
            TOTAL DE ESTACIONES: 26
MOSTRANDO RUTA: Vallejo-->Vallejo-->Norte 45-->Ferrería/Arena Ciudad de México-->Azcapotzalco-->Tezozómoc-->El Rosario-->Aquiles Serdán-->Camarones-->Refinería-->Tacuba-->Cuitláhuac-->Popotla-->Colegi
o Militar-->Normal-->San Cosme-->Revolución-->Hidalgo-->Bellas Artes-->Allende-->Zócalo-->Pino Suárez-->Isabel la Católica-->Salto del Agua-->Balderas-->Cuauhtémoc-->Insurgentes
      ##########CAMBIANDO DE PRUEBA##########
----------R U T A - B F S----------
      PARADA: Aquiles Serdán       DESTINO: Iztapalapa
            TOTAL DE ESTACIONES: 21
MOSTRANDO RUTA: Aquiles Serdán-->Aquiles Serdán-->Camarones-->Refinería-->Tacuba-->San Joaquín-->Polanco-->Auditorio-->Constituyentes-->Tacubaya-->San Pedro de los Pinos-->San Antonio-->Mixcoac-->Insu
rgentes Sur-->20 de Noviembre-->Zapata-->Parque de los Venados-->Eje Central-->Ermita-->Mexicaltzingo-->Atlalilco-->Iztapalapa
----------R U T A - D F S----------
      PARADA: Aquiles Serdán       DESTINO: Iztapalapa
            TOTAL DE ESTACIONES: 52
MOSTRANDO RUTA: Aquiles Serdán-->Aquiles Serdán-->El Rosario-->Tezozómoc-->Azcapotzalco-->Ferrería/Arena Ciudad de México-->Norte 45-->Vallejo-->Instituto del Petróleo-->Autobuses del Norte-->La Raza-
->Potrero-->Deportivo 18 de Marzo-->La Villa-Basílica-->Martín Carrera-->Talismán-->Bondojito-->Consulado-->Canal del Norte-->Morelos-->Candelaria-->Merced-->Pino Suárez-->Isabel la Católica-->Salto de
l Agua-->Balderas-->Cuauhtémoc-->Insurgentes-->Sevilla-->Chapultepec-->Juanacatlán-->Tacubaya-->San Pedro de los Pinos-->San Antonio-->Mixcoac-->Insurgentes Sur-->20 de Noviembre-->Zapata-->División de
l Norte-->Eugenia-->Etiopía/Plaza de la Transparencia-->Centro Médico-->Lázaro Cárdenas-->Chabacano-->Viaducto-->Xola-->Villa de Cortés-->Nativitas-->Portales-->Ermita-->Mexicaltzingo-->Atlalilco-->Izt
apalapa
      ##########CAMBIANDO DE PRUEBA##########
----------R U T A - B F S----------
      PARADA: San Antonio    DESTINO: Aragón
            TOTAL DE ESTACIONES: 16
MOSTRANDO RUTA: San Antonio-->San Antonio-->San Pedro de los Pinos-->Tacubaya-->Patriotismo-->Chilpancingo-->Centro Médico-->Lázaro Cárdenas-->Chabacano-->Jamaica-->Fray Servando-->Candelaria-->Morelo
s-->Canal del Norte-->Consulado-->Eduardo Molina-->Aragón
----------R U T A - D F S----------
      PARADA: San Antonio    DESTINO: Aragón
            TOTAL DE ESTACIONES: 25
MOSTRANDO RUTA: San Antonio-->San Antonio-->San Pedro de los Pinos-->Tacubaya-->Juanacatlán-->Chapultepec-->Sevilla-->Insurgentes-->Cuauhtémoc-->Balderas-->Salto del Agua-->Isabel la Católica-->Pino S
uárez-->Merced-->Candelaria-->San Lázaro-->Moctezuma-->Balbuena-->Boulevard Puerto Aéreo-->Gómez Farías-->Zaragoza-->Pantitlán-->Hangares-->Terminal Aérea-->Oceanía-->Aragón
```

CONCLUCIONES:

El algoritmo BFS es **completo**, es decir, si existe solución, este algoritmo la encuentra. Más aún, es **óptimo**, en el sentido de que si hay solución, encuentra una de las soluciones a distancia mínima de la raíz.
En cambio, DFS no es ni óptimo ni completo. No es óptimo porque si existe más de una solución, podría encontrar la primera que estuviese a un nivel de profundidad mayor, y para ver que no es completo es necesario irse a ejemplos en los que el espacio de búsqueda fuese infinito

1. La búsqueda en anchura se recomienda cuando lo que se necesita es buscar el camino más corto en grafos no ponderados.
2. La búsqueda en profundidad se puede utilizar para detectar ciclos en un grafo, determinar si un grafo es conexo o no y cuántas componentes conexas tiene, determinar puntos de articulación y biconexión de grafos, entre otras cosas.
3. Cuando no tiene importancia el orden en que visitemos los nodos y aristas del grafo, se puede usar cualquiera de los algoritmos de búsqueda BFS y DFS son una de las herramientas básicas a la hora de trabajar con grafos. No sólo podremos usarlos para recorrer grafos o buscar elementos, sino que también podemos adaptarlos y mejorarlos para resolver de manera eficiente cualquier tipo de situaciones que podamos moldear como un grafo o un arbol.