Actividad:

Integradora 2

Diego Mellado Oliveros A01655451
Iwalani Amador Piaga A01732251
Tonatiuh Reyes Huerta A01025459

Campus Santa Fe

Análisis y diseño de algoritmos avanzados

TC2038

Introducción a la situación problema:

A partir de la situación originada por el COVID-19, durante el año 2022 se vivió una pandemia que obligó a la población a tomar medidas de prevención para contener las consecuencias, una de estas medidas fue el enviar a todos a casa, retirando las actividades presenciales y obligando a la población a trabajar o realizar ciertas actividades en formato remoto, gracias a ello, la necesidad del uso de internet incrementó al ser un medio de comunicación a distancia, así que los proveedores de servicios de internet (ISP por sus siglas en inglés de Internet Service Provider) comenzaron a tener una gran responsabilidad, ya que gracias a ellos la mayor parte de actividades se mantuvieron durante la pandemia, transformándose en trabajo remoto, o home-office, misma modalidad que se implementó en instituciones educativas, cuestiones que aumentaron de gran manera la transmisión de datos en internet.

Bajo este panorama se formulan las siguientes interrogantes:
Si estuviera en nuestras manos mejorar los servicios de Internet en una población pequeña, ¿podríamos decidir cómo cablear los puntos más importantes de dicha población de tal forma que se utilice la menor cantidad de fibra óptica?
Asumiendo que tenemos varias formas de conectar dos nodos en la población,
Para una persona que tiene que ir a visitar todos los puntos de la red, ¿Cuál será la forma óptima de visitar todos los puntos de la red?
¿Podríamos analizar la cantidad máxima de información que puede pasar desde un nodo a otro ?
¿Podríamos analizar la factibilidad de conectar a la red un nuevo punto (una nueva localidad) en el mapa ?

Descripción del funcionamiento:

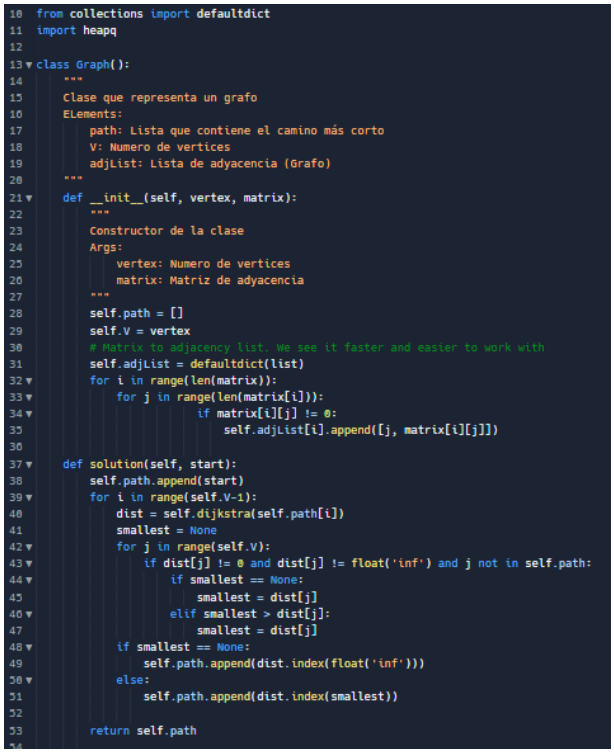
Con el propósito de poner en práctica lo aprendido durante el curso, en esta actividad integradora se integran contenidos de la actividad integradora 1 más los nuevos algoritmos aprendidos, para poder una solución a la problemática antes mencionada, para ello se diseñó un programa en Python que consiste en:

1. Leer un archivo de entrada que contiene la información de un grafo representado en forma de una matriz de adyacencias con grafos ponderados.
   Ya que se considera que el peso de cada arista es la distancia en kilómetros de una matriz de adyacencias con grafos ponderados, es decir, la distancia entre cada colonia por donde es factible incluir cableado.
2. Desplegar la forma más óptima para el cableado con fibra óptica, conectando todas las colonias entre sí.

Para dar solución a este problema que contiene muchas similitudes con el “problema del viajero”, se empleó el mayormente el algoritmo de búsqueda Dijkstra, ya que este nos permite utilizar grafos con peso, así como lo marca en las especificaciones del programa.
Dijkstra es un algoritmo que sirve para hallar el camino más corto entre nodos diferentes, consiste en ir explorando los caminos más cortos que parten desde un origen y llevan hacia los

demás destinos, cuando se recorre el grafo marcando como visitados cada nodo, el algoritmo se detiene, obteniendo así el camino más corto, comparando el costo de todos los nodos conectados para elegir el nodo con menor coste.

Capturas del programa:

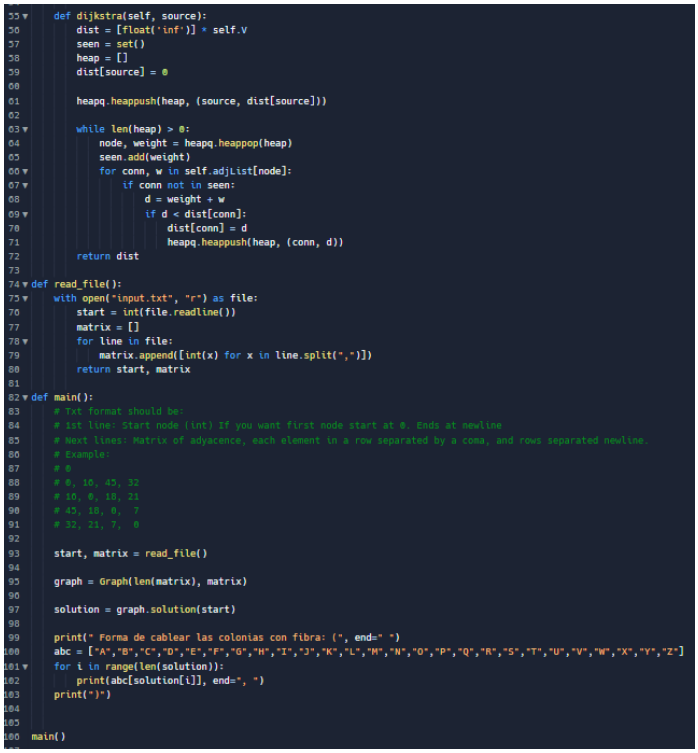


```python
from collections import defaultdict
import heapq

class Graph():
    """
    Clase que representa un grafo
    ELements:
        path: Lista que contiene el camino más corto
        V: Numero de vertices
        adjList: Lista de adyacencia (Grafo)
    """
    def __init__(self, vertex, matrix):
        """
        Constructor de la clase
        Args:
            vertex: Numero de vertices
            matrix: Matriz de adyacencia
        """
        self.path = []
        self.V = vertex
        # Matrix to adjacency list. We see it faster and easier to work with
        self.adjList = defaultdict(list)
        for i in range(len(matrix)):
            for j in range(len(matrix[i])):
                    if matrix[i][j] != 0:
                        self.adjList[i].append([j, matrix[i][j]])

    def solution(self, start):
        self.path.append(start)
        for i in range(self.V-1):
            dist = self.dijkstra(self.path[i])
            smallest = None
            for j in range(self.V):
                if dist[j] != 0 and dist[j] != float('inf') and j not in self.path:
                    if smallest == None:
                        smallest = dist[j]
                    elif smallest > dist[j]:
                        smallest = dist[j]
            if smallest == None:
                self.path.append(dist.index(float('inf')))
            else:
                self.path.append(dist.index(smallest))

        return self.path
```



```python
    def dijkstra(self, source):
        dist = [float('inf')] * self.V
        seen = set()
        heap = []
        dist[source] = 0

        heapq.heappush(heap, (source, dist[source]))

        while len(heap) > 0:
            node, weight = heapq.heappop(heap)
            seen.add(weight)
            for conn, w in self.adjList[node]:
                if conn not in seen:
                    d = weight + w
                    if d < dist[conn]:
                        dist[conn] = d
                        heapq.heappush(heap, (conn, d))
        return dist

def read_file():
    with open("input.txt", "r") as file:
        start = int(file.readline())
        matrix = []
        for line in file:
            matrix.append([int(x) for x in line.split(",")])
        return start, matrix

def main():
    # Txt format should be:
    # 1st line: Start node (int) if you want first node start at 0. Ends at newline
    # Next lines: Matrix of adyacence, each element in a row separated by a coma, and rows separated newline.
    # Example:
    # 0
    # 0, 10, 45, 32
    # 10, 0, 18, 21
    # 45, 18, 0,  7
    # 32, 21, 7,  0

    start, matrix = read_file()

    graph = Graph(len(matrix), matrix)

    solution = graph.solution(start)

    print(" Forma de cablear las colonias con fibra: (", end=" ")
    abc = ["A","B","C","D","E","F","G","H","I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"]
    for i in range(len(solution)):
        print(abc[solution[i]], end=", ")
    print(")")


main()
```

Al concluir exitosamente con la actividad se determinó que la complejidad del programa es O(ElogV) donde V es el número de vértices, y E es el número de aristas del grafo.

Tres aplicaciones (distintas de la presentada en esta situación) de los algoritmos empleados en esta actividad.

Ya que el algoritmo utilizado para resolver el problema fue Dijkstra, se pueden resolver distintos escenarios:

- Aplicaciones para sistemas de información geográficos: Donde se puede extraer información del terreno, dependiendo de sus características, por ejemplo, características curvilíneas de imágenes usando técnicas de minimización del camino, la imagen se puede representar como una matriz de puntos, cada uno con una intensidad y cada nodo puede ser un punto (píxel) de la imagen. Al final como en todas las representaciones graficas de los espacios geograficos el peso de las aristas seria la diferencia de color.
- Enrutamiento de aviones y tráfico aéreo: Con un agente de solicitudes de viaje, que puede ser ejecutado desde un software para hacer un programa de vuelos para los clientes, para ello el agente tendría acceso a una base de datos de los aeropuertos vinculados y los

vuelos que tienen, donde para distinguirlo será necesario contar con un ID que puede ser el número de vuelo, además de otros datos de identificación como el aeropuerto de origen y destino y los horarios de salida y de llegada. Con lo anterior, la aplicación puede ser utilizada para determinar una hora de llegada, pero no cualquiera, sino la más temprana para el destino dado un aeropuerto de origen y hora de inicio.

- Rodear una montaña: La escalada es un deporte que requiere riesgos y en muchas competiciones se busca llegar primero con el menor agotamiento posible, por ello es posible el utilizar este algoritmo para rodear una montaña o subir por el camino más corto conociendo los puntos, como muchas veces se tienen en las competencias para ir monitoreando la salud y los movimientos de cada competidor, donde cada punto de encuentro se convierta en un nodo a visitar, partiendo desde un punto de inicio, cada arista se convierte en un camino para llegar de punto A a B.
- Reconocimiento de lenguaje hablado: Se puede construir un grafo cuyos vértices correspondan a palabras posibles y las aristas unan palabras que pueden ir colocadas al lado de las otras. Si el peso de las aristas corresponde a la probabilidad de que estén así colocadas, el camino más corto en el grafo será la mejor interpretación de la frase.

Otros problemas “reales que puede resolverse con Dikstra son:

- Llegar a desde un punto de una ciudad hasta otro por el camino más rápido
- Conocer el camino más rápido que sigue la información a través de las neuronas.

Reflexión final:

Al utilizar el algoritmo de Dijkstra fue posible concluir la actividad integradora 2 de forma exitosa y nos permitió explorar un poco más sobre sus aplicaciones en situaciones reales, dándonos a comprender que un algoritmo que es diseñado para un problema específico, como en este caso que es encontrar el camino más corto, no siempre se queda en esa única aplicación, hay que convertir y pensar más allá, transformar la idea manteniendo la parte potencial de su funcionamiento.

Finalmente como equipo, con el curso comprendimos muchos algoritmos que serán de ayuda para nuestros próximos proyectos, algoritmos que aunque no fueron aplicados en esta situacion problema, estamos agradecidos de entender ya que aplicarlos de forma manual sería bastante complejo y confuso, quizás algunos algoritmos tengan implicaciones en escenarios más específicas, pero analizandolos, descubrimos que muchos de ellos son de valor y aplicables en problemas complejos que suceden en las industrias de logística.