Instituto Politécnico Nacional

Práctica 02: Implementación y Evaluación del Algoritmo de Dijkstra

Adan Torres Gutierrez

1 de diciembre de 2023

1. Introducción

En este reporte se plasmara la manera en que fue implementamos el algoritmo de Dijkstra en Python para la resolución del problema de encontrar caminos mínimos, el cual es muy recurrente y tiene distintas utilidades como en la implementación de redes de transporte.

2. Desarrollo

Primeramente creamos la clase Graph como es mencionado en el documento de la practica, en el que definimos las vértices y las aristas, como sus listas para agregarlos y sus funciones para ser llamados. Además de importar la biblioteca time para mas tarde.

```python
import time

class Graph:
    def __init__(self):
        # Conjunto de vertices
        self.nodes = set()
        # conjunto de aristas
        self.edges = {}

    def add_vertex(self, valor):
        # Agregar un nuevo vértice al grafo
        self.nodes.add(valor)
        # Inicializar la lista de aristas para el vértice agregado
        self.edges[valor] = []

    def add_edge(self, devertice, avertice, peso):
        # Agregar una nueva arista al grafo
        self.edges[devertice].append((avertice, peso))
        self.edges[avertice].append((devertice, peso))
```

Figura 1: primera parte del codigo.

Seguidamente de esto implementamos el algoritmo Dijkstra en el que definimos la distancia de las vértices como distancia y la inicializamos en 0 y todas las de las demás vértices su distancia a infinito y definimos al predecesor de las vértices como padre y iniciamos en nulo porque aun no sabemos cual es la vértice predecesora. Definimos una lista donde se guardara las vértices que no hemos visitado esta tiene el nombre de unvisited con el algoritmo Dijkstra, creamos un while para visitar estas vértices y elegir la que tiene menor distancia entre si, dentro de este while también se actualizaran los datos para guardar el camino mas corto. Al salir de este while definimos una función para cambiar el predecesor de cada vértice hacia la vértice destino para a si seguir a la siguiente vértice que no hemos visitado y volver a actualizar los datos hasta que no hayamos quedado sin vértices que visitar en el código.

En esta parte del código definimos una función para medir el tiempo que

```python
def dijkstra(graph, start):
    # Inicializar distancias y padres
    distancia = {node: float('infinity') for node in graph.nodes}
    distancia[start] = 0
    padre = {node: None for node in graph.nodes}

    # Lista de vértices no visitados
    unvisited = list(graph.nodes)

    while unvisited:
        # Seleccionar el vértice no visitado con distancia mínima
        current = min(unvisited, key=lambda node: distancia[node])
        unvisited.remove(current)

        # Actualizar las distancias de los vértices vecinos
        for vecino, peso in graph.edges[current]:
            new_distancia = distancia[current] + peso

            if new_distancia < distancia[vecino]:
                distancia[vecino] = new_distancia
                padre[vecino] = current

    return distancia, padre

def get_path(padre, destination):
    path = [destination]
    while padre[destination] is not None:
        destination = padre[destination]
        path.insert(0, destination)
    return path
```

Figura 2: Algoritmo Dijkstra.

tarda el codigo. Aquí ya podemos ver el final del código en el que con las funciones add vertex y add edge añado vértices y aristas con sus respectivos pesos o distancia entre cada vértice, así como mandar a imprimir todos los resultados junto con estos el tiempo en que tarda el código.

```python
def measure_time():
    # Código para medir el tiempo de ejecución
    start_time = time.time()

    grafo = Graph()
    grafo.add_vertex("A")
    grafo.add_vertex("B")
    grafo.add_vertex("C")
    grafo.add_edge("A", "B", 3)
    grafo.add_edge("A", "C", 5)
    grafo.add_edge("B", "C", 2)

    start_node = 'A'
    results, padre = dijkstra(grafo, start_node)

    for node, distancia in results.items():
        if node != start_node:
            path = get_path(padre, node)
            print(f"El camino más corto de {start_node} a {node}: {path}, con distancia de: {distancia}")

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"Tiempo de ejecución: {elapsed_time} segundos")

measure_time()
```

Figura 3: Parte final del codigo.

3. Resultados

Aquí presentare un par de imágenes donde se ven los resultados de este código.

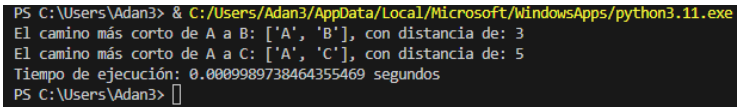

Figura 4: Resultado 1.

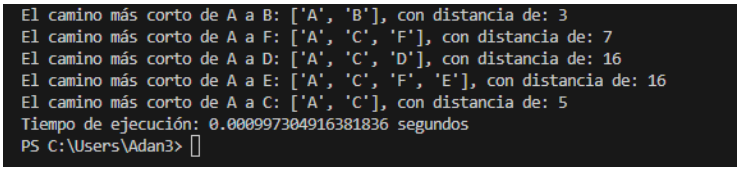

Figura 5: Resultado 2.

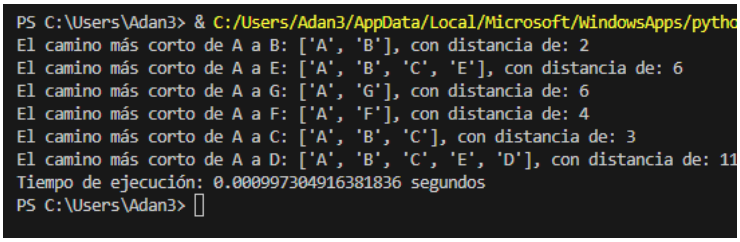

Figura 6: Resultado 3.

4. Conclusiones

En conclusión, el código escrito implementa el algoritmo de Dijkstra correctamente para un grafo y nos da todos los caminos para llegara cualquier vértice del grafo, cumpliendo su funcion.