UNIDAD PROFESIONAL INTERDICIPLINARIA DE INGENIERÍA CAMPUS ZACATECAS.

Práctica 3

Análisis y diseño de algoritmos

Autor:
Emilio de Jesús Ureño Padilla

Fecha:
01-Diciembre-2023

1 Introducción

En esta práctica se nos pide la implementación del Algoritmo de Dijkstra para la resolución de problemas de grafos de una manera sencilla y eficiente; este algoritmo se usa mucho en la planificación en rutas de mapas, en redes de comunicación para encontrar la ruta más eficiente entre dos nodos, y en problemas logísticos donde se busca minimizar el costo de transporte.

2 Desarrollo

2.1 Implementación

Con el algoritmo de Dijkstra, puedes encontrar la ruta más corta o el camino más corto entre los nodos de un grafo. Específicamente, puedes encontrar el camino más corto desde un nodo (llamado el nodo de origen) a todos los otros nodos del grafo, generando un árbol del camino más corto.
Funciona de la siguiente manera dentro del código:

- Definición de la clase Grafo:

  La clase Grafo representa un grafo dirigido ponderado. Tiene un conjunto de vértices (self.vertices) y un diccionario de aristas (self.aristas).

```
class Grafo:
    def __init__(self):
        self.vertices = set()
        self.aristas = {}
```

- Método agregar vertice:

  Agrega un vértice al grafo. Inicializa la lista de aristas para ese vértice como vacía.

```
def agregar_vertice(self, vertice):
    self.vertices.add(vertice)
    self.aristas[vertice] = []
```

- Método agregar arista:

  Agrega una arista ponderada entre dos vértices al grafo.

```
def agregar_arista(self, inicio, fin, peso):
    self.aristas[inicio].append((fin, peso))
```

- Función dijkstra:

  Recibe el grafo y un vértice de inicio como parámetros.
  Inicializa un diccionario distancias con distancias iniciales infinitas para cada vértice, excepto el vértice de inicio que tiene distancia cero.
  Utiliza una cola de prioridad (prioridad) para explorar los vértices en orden de distancia.
  Mientras haya vértices en la cola de prioridad, se extrae el vértice con la menor distancia actual.
  Para cada vecino del vértice actual, se actualiza la distancia si se encuentra un camino más corto.
  El bucle continúa hasta que se hayan explorado todos los vértices alcanzables.

```python
def dijkstra(grafo, inicio):
    distancias = {vertice: float('inf') for vertice in grafo.vertices}
    distancias[inicio] = 0
    prioridad = [(0, inicio)]

    while prioridad:
        distancia_actual, vertice_actual = heapq.heappop(prioridad)

        if distancia_actual > distancias[vertice_actual]:
            continue

        for vecino, peso in grafo.aristas[vertice_actual]:
            distancia = distancia_actual + peso

            if distancia < distancias[vecino]:
                distancias[vecino] = distancia
                heapq.heappush(prioridad, (distancia, vecino))

    return distancias
```

2.2 Resultados

- Ejemplo 1

```python
g1 = Grafo()
g1.agregar_vertice('A')
g1.agregar_vertice('B')
g1.agregar_vertice('C')
g1.agregar_arista('A', 'B', 3)
g1.agregar_arista('B', 'C', 2)
g1.agregar_arista('A', 'C', 5)
```

```
Caminos mínimos desde A:
Vértice C: Distancia 5
Vértice B: Distancia 3
Vértice A: Distancia 0
Tiempo de ejecución: 0.0 segundos
```

- Ejemplo 2

```
g2 = Grafo()
g2.agregar_vertice('A')
g2.agregar_vertice('B')
g2.agregar_vertice('C')
g2.agregar_vertice('D')
g2.agregar_arista('A', 'B', 1)
g2.agregar_arista('B', 'C', 2)
g2.agregar_arista('C', 'D', 4)
g2.agregar_arista('A', 'D', 7)
```

```
Caminos mínimos desde A:
Vértice D: Distancia 7
Vértice A: Distancia 0
Vértice C: Distancia 3
Vértice B: Distancia 1
Tiempo de ejecución: 0.0 segundos
```

- Ejemplo 3

```
g3 = Grafo()
g3.agregar_vertice('A')
g3.agregar_vertice('B')
g3.agregar_vertice('C')
g3.agregar_vertice('D')
g3.agregar_vertice('E')
g3.agregar_arista('A', 'B', 4)
g3.agregar_arista('A', 'C', 2)
g3.agregar_arista('C', 'B', 5)
g3.agregar_arista('B', 'D', 10)
g3.agregar_arista('C', 'D', 1)
g3.agregar_arista('D', 'E', 3)
```

```
Caminos mínimos desde A:
Vértice B: Distancia 4
Vértice C: Distancia 2
Vértice D: Distancia 3
Vértice E: Distancia 6
Vértice A: Distancia 0
Tiempo de ejecución: 0.0009999275207519531 segundos
```

- Ejemplo 4

```
g4 = Grafo()
g4.agregar_vertice('A')
g4.agregar_vertice('B')
g4.agregar_vertice('C')
g4.agregar_vertice('D')
g4.agregar_vertice('E')
g4.agregar_arista('A', 'B', 5)
g4.agregar_arista('B', 'C', 8)
g4.agregar_arista('A', 'C', 12)
g4.agregar_arista('C', 'D', 4)
g4.agregar_arista('B', 'D', 10)
g4.agregar_arista('D', 'E', 6)
g4.agregar_arista('A', 'E', 15)
```

```
Caminos mínimos desde A:
Vértice A: Distancia 0
Vértice E: Distancia 15
Vértice B: Distancia 5
Vértice C: Distancia 12
Vértice D: Distancia 15
Tiempo de ejecución: 0.0 segundos
```

- Ejemplo 5

```
g5 = Grafo()
g5.agregar_vertice('A')
g5.agregar_vertice('B')
g5.agregar_vertice('C')
g5.agregar_vertice('D')
g5.agregar_vertice('E')
g5.agregar_vertice('F')
g5.agregar_vertice('G')
g5.agregar_vertice('H')
g5.agregar_vertice('I')
g5.agregar_arista('A', 'B', 3)
g5.agregar_arista('B', 'C', 7)
g5.agregar_arista('C', 'D', 10)
g5.agregar_arista('D', 'E', 2)
g5.agregar_arista('E', 'F', 6)
g5.agregar_arista('F', 'G', 8)
g5.agregar_arista('G', 'H', 5)
g5.agregar_arista('H', 'I', 4)
g5.agregar_arista('A', 'I', 15)
```

```
Caminos mínimos desde A:
Vértice G: Distancia 36
Vértice E: Distancia 22
Vértice H: Distancia 41
Vértice A: Distancia 0
Vértice F: Distancia 28
Vértice I: Distancia 15
Vértice B: Distancia 3
Vértice D: Distancia 20
Vértice C: Distancia 10
Tiempo de ejecución: 0.0 segundos
```

3 Conclusión

Después de investigar varios usos que se le pueden dar a este código el que se puede encontrar más común poe así decirlo es en el uso que se le da en las aplicaciones de mapas, esto nos ayuda a encontrar el camino más rápido posible, no sé que uso se le puede dar fuera de estos ámbitos de querer encontrar el camino más corto pero es un algortimo entrtenido de utilizar gracias a los grafos que se implementan.

4 Referencias

Navone, E. C. (2023, 2 agosto). Algoritmo de la ruta más corta de Dijkstra - Introducción gráfica y detallada. freeCodeCamp.org.
`https://acortar.link/C8M1D5`