Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria de Ingeniería

Campus Zacatecas.

Ejercicio - Examen:

"Algoritmo de Dijkstra'.

Materia y Grupo:

Análisis y Diseño de Algoritmos. 3CM2.

Profesora a cargo:

Erika Paloma Sánchez-Femat.

Alumno:

Jorge Ulises Zapata Arteaga.

1 Introducción.

En este documento se estudiará la implementación del algoritmo de Dijkstra, el cual consiste en buscar el camino más corto para llegar de un vértice a otro en un grafo ponderado, en este caso será dirigido.
Para la documentación se usarán 5 casos de prueba para verificar el funcionamiento del algoritmo. Además se calculará el tiempo de ejecución de cada caso y su complejidad.

2 Desarrollo.

I) Define una función dijkstra que tome el grafo y un vértice de inicio como entrada y devuelva los caminos mínimos desde el vértice de inicio a todos los demás vértices. Puedes utilizar un diccionario para mantener un registro de las distancias mínimas.
II) Crea un grafo de ejemplo utilizando la clase Graph y agrega algunos vértices y aristas con pesos. Ejemplo:

```
grafo = Grafo()
grafo.addVertex("A")
grafo.addVertex("B")
grafo.addVertex("C")
grafo.addEdge("A", "B", 3)
grafo.addEdge("A", "C", 5)
grafo.addEdge("B", "C", 2)
```

III) Utiliza la función dijkstra para encontrar los caminos mínimos desde un vértice de inicio dado. Puedes imprimir los resultados para verificar que el algoritmo funcione correctamente.

- **Proceso de implementación del algoritmo.**

Paso 1: Iniciamos definiendo una clase 'Grafo' con su respectivo constructor y a su vez declaramos un diccionario para guardar los vértices y las aristas del grafo (self.vertices).
Paso 2: Definimos una función *"agregarvertice"* para agregar un vértice al grafo, para esto verificamos primero si el vértice que vamos a agregar no existe en el grafo, esto es: "si el vertice no está en el diccionario de vértices" (if vertice not in self.vertices). Si esto se cumple entonces lo agregamos (self.vertices[vertice] = []).
Paso 3.- A continuación definimos una función *"agregararista"*que nos agrega una arista ponderado (con un peso) entre dos vértices, pero para que pueda agregar la arista, los vértices deben existir en el grafo, esto es: "si el vértice "desdevertice" y el vértice "hastavertice" están en el diccionario de vértices" (if desdevertice in self.vertices and haciavertice in self.vertices:). Si esto se cumple, entonces agrega el arista entre esos dos vértices con el peso designado.
Paso 4.- Definimos una nueva función *"dijkstra"* que recibe

como parámetros el grafo y un vértice de inicio.
Paso 5.- en esta función definimos un nuevo diccionario "distancias" para llevar un registro de las distancias mínimas desde el vértice de inicio hasta los demás vértices. Este diccionario, inicialmente los valores los asignamos como infinito, ya que al inicio del algoritmo no conocemos las distancias mínimas desde el vértice de inicio hasta los demás vértices. Esto nos sirve para establecer que, inicialmente, no hay un camino conocido desde el vértice de inicio hasta los demás vértices. Pero, hasta que se "explore" o se llegue al vértice de destino se encontrará una distancia real que obviamente será menor que infinito, por lo tanto la podremos actualizar (distancias = vertice: float('infinity') for vertice in grafo.vertices).
Paso 6.- Al mismo tiempo, establecemos en el diccionario que la distancia hacia el vértice de inicio es 0 ya que, mientras no haya un camino explícito (arista) en el grafo que conecte un vértice consigo mismo, la distancia de ese vértice hasta él mismo es siempre 0.(distancias[inicio] = 0).
Paso 7.- Inicializamos la cola de prioridad, esta se define como una lista que guarda tuplas de la forma (n, inicio), cada tupla representa la distancia acumulada y el vértice actual. En este caso, estamos considerando que la distancia acumulada desde el vértice de inicio hasta él mismo es cero. Por lo tanto tenemos: (colaprioridad = [(0, inicio)]).
Paso 8: A partir de este paso empieza la magia de la

comparación y el corazón del algoritmo:

- **Inicio del bucle principal (while colaprioridad):**

El bucle se ejecuta mientras la cola de prioridad colaprioridad no esté vacía. La cola de prioridad contiene tuplas donde el primer elemento es la distancia acumulada y el segundo es el vértice actual.

- **Extracción del vértice con menor distancia (heapq.heappop(colaprioridad)):**

Se utiliza heapq.heappop para extraer el elemento con la menor distancia acumulada de la cola de prioridad. La tupla resultante se obtiene en distanciaactual y verticeaactual.

- **Verificamos la distancia actual (if distanciaactual > distancias[verticeactual]: continue):**

Se verifica si la distancia actual (extraída de la cola de prioridad) es mayor que la distancia registrada en distancias para el vértice actual. Si es así, se ignora esta iteración del bucle (continue), ya que se ha encontrado un camino más corto en una iteración anterior.

- **Probamos los vértices adyacentes (for adyacente, peso in grafo.vertices[verticeaactual]:):**

Se itera sobre los vértices adyacentes al vértice actual. grafo.vertices[verticeactual] es la lista de aristas salientes del vértice actual, donde cada arista es una tupla (adyacente, peso).

- **Calculamos la distancia acumulada (distancia = distanciaaactual + peso):**

Para cada vértice adyacente, se calcula la distancia acumulada desde el vértice de inicio a través del vértice actual. La distancia acumulada es la distancia actual más el peso de la arista que conecta el vértice actual con el vértice adyacente.

- **Comparación y actualización de distancias (if distancia < distancias[adyacente]:)**

Se compara la distancia acumulada recién calculada con la distancia registrada en distancias para el vértice adyacente. Si la nueva distancia es menor, se actualiza la distancia registrada en distancias y se agrega la nueva distancia y vértice a la cola de prioridad usando heapq.heappush.

Paso 9: Después de ejecutar Dijkstra, se reemplazan los valores infinitos en el diccionario distancias con el mensaje "Infinito", simulando el símbolo del infinito en el proceso ya conocido Dijkstra, se logra con esta.
Paso 10: Para finalizar, se plantean los casos de prueba y así verificamos el funcionamiento del algoritmo.
Tomando en cuenta el siguiente grafo, Figure 2:

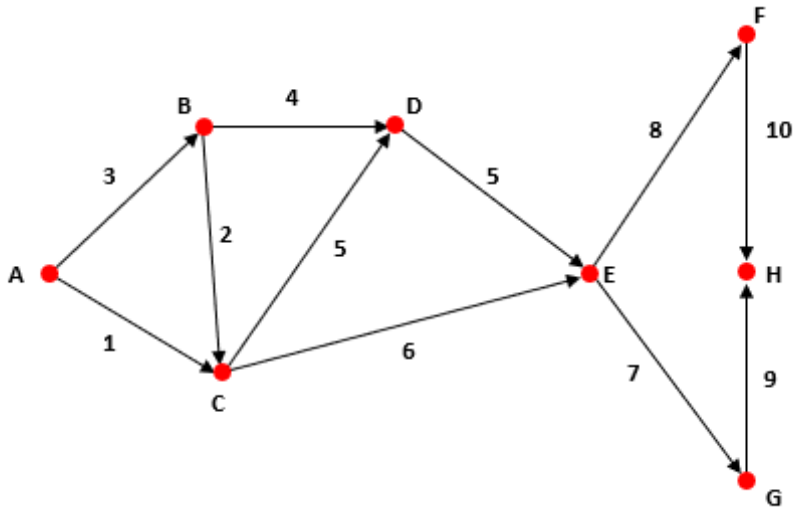

Figure 2: Grafo de ejemplo.

El primer caso de prueba es agarrar "A" como el vértice de inicio. Por lo tanto usando el simple inspección para encontrar los caminos mínimos: De A a A = 0, de A a B = 3, de A a C = 1, de A a D = 6 (primero pasa a C y se pasa a D 1+5), de A a E = 7(se va a C y llega a E 1+6), de A a F = 15(se va a C, pasa a E y llega a F 1+6+8), de A a G = 14(se va a C, pasa a E y llega a G 1+6+7), de A a H = 23(se va a C, pasa a E, pasa a G

y llega a H 1+6+7+9). Nos da como resultado A(A=0, B=3, C=1, D=6, E=7, F=15, G=14, H=23).

El segundo caso de prueba es agarrar "B" como el vértice de inicio. Por lo tanto, usando simple inspección para encontrar los caminos mínimos: de B a A = ∞ debido a que no hay camino de B a A, de B a B = 0, de B a C = 2, de B a D = 4, de B a E = 8(se va a C y llega a E 2+6), de B a F = 16(se va a C, pasa a E y llega a F 2+6+8), de B a G = 15(se va a C, pasa a E y llega a G 2+6+7), de B a H = 24(se va a C, pasa a E, pasa a G y llega a H 2+6+7+9). Nos da como resultado B(A=∞, B=0, C=2, D=4, E=8, F=16, G=15, H=24).

El tercer caso de prueba es agarrar "C" como el vértice de inicio. Por lo tanto, usando simple inspección para encontrar los caminos mínimos: de C a A = ∞, de C a B = ∞, de C a C = 0, de C a D = 5, de C a E = 6, de C a F = 14(se va a E y llega a F 6+8), de C a G = 13(se va a E y llega a G 6+7), de C a H = 22(se va a E, pasa a G y llega a H 6+7+9). Nos da como resultado C(A=∞, B= ∞, C=0, D=5, E=6, F=14, G=13, H=22).

El cuarto caso de prueba es agarrar "D" como el vértice de inicio. Por lo tanto, usando simple inspección para encontrar los caminos mínimos: de D a A = ∞, de D a B = ∞, de D a C = ∞, de D a D = 0, de D a E = 5, de

D a F = 13(se va a E y llega a F 5+8), de D a G = 12(se va a E y llega a G 5+7), de D a H = 21(se va a E, pasa a G y llega a H 5+7+9). Nos da como resultado D(A=∞, B= ∞, C=∞, D=0, E=5, F=13, G=12, H=21).

El quinto caso de prueba es agarrar "E" como el vértice de inicio. Por lo tanto, usando simple inspección para encontrar los caminos mínimos: de E a A = ∞, de E a B = ∞, de E a C = ∞, de E a D = ∞, de E a E = 0, de E a F = 8, de E a G = 7, de E a H = 16(se va a G y llega a H 7+9)
A continuación se muestra la salida en consola corriendo el algoritmo y nos damos cuenta que los resultados analizados y los arrojados son correctos. Figure 3.

```
CASOS DE PRUEBA ALGORITMO DIJKSTRA:

Caminos mínimos desde A: {'A': 0, 'B': 3, 'C': 1, 'D': 6, 'E': 7, 'F': 15, 'G': 14, 'H': 23}

Caminos mínimos desde B: {'A': 'Infinito', 'B': 0, 'C': 2, 'D': 4, 'E': 8, 'F': 16, 'G': 15, 'H':
    24}

Caminos mínimos desde C: {'A': 'Infinito', 'B': 'Infinito', 'C': 0, 'D': 5, 'E': 6, 'F': 14, 'G':
    13, 'H': 22}

Caminos mínimos desde D: {'A': 'Infinito', 'B': 'Infinito', 'C': 'Infinito', 'D': 0, 'E': 5, 'F':
    13, 'G': 12, 'H': 21}

Caminos mínimos desde E: {'A': 'Infinito', 'B': 'Infinito', 'C': 'Infinito', 'D': 'Infinito', 'E'
    : 0, 'F': 8, 'G': 7, 'H': 16}
```

Figure 3: Grafo de ejemplo.

- **Complejidad del algoritmo.**

Con lo que analizamos previamente podemos decir que el algoritmo de Dijkstra tiene una complejidad $O(n^2)$,

donde n es el número de vértices en el grafo; sin embargo, cabe aclarar que no es eficiente para grafos muy grandes, solo es eficiente para un número considerable de vértices.

3 Conclusión.

Como podemos ver en la documentación de la práctica, el Algoritmo de Dijkstra, para situciones cotidianas, nos ayuda a encontrar el camino más corto para un grafo, y en cuanto a la programación nos ayudó esta vez a aplicar conceptos previos del uso de clases, funciones y diccionarios en python. Haciendo más sencillo la representación de las salidas de consola en los programas. Pudiendo concluir que se realizó con éxito la práctica.

4 Referencias Bibliográficas.

https://www.udb.edu.sv/udbfiles/recursosguias/informatica-ingenieria/programacion-iv/2019/ii/guia-10.pdf

http://bioinfo.uib.es/ joemiro/aenui/procJenui/ProcWeb/actas2001/saalg223.pdf

https://kodifi.co/el-algoritmo-de-dijkstra/