Reporte de algoritmo de Dijkstra.

María Luisa Borrego Riojas
Universidad Autónoma de Nuevo León
Facultad de Ciencias Físico Matemáticas
Matemáticas Computacionales

En este reporte se buscará mostrar y analizar el algoritmo de Dijkstra, cómo se comporta con diferentes grafos y la manera en la que es implementado en Python.

1. Dijkstra.

En pocas palabras, el algoritmo de Dijkstra nos ayuda a, dado un grafo en donde las aristas tienen un valor determinado, encontrar el camino con menor valor para poder llegar desde un punto de partida (nodo inicial) hasta el resto de los nodos. Una manera sencilla para hacer de esto algo más práctico y funcional, es pensar en los nodos como destinos, en donde queremos pasar a cada destino gastando la menor cantidad de dinero, o recorriendo la menor distancia, etc. (el valor de las aristas).

Para lograrlo, el algoritmo de Dijkstra tiene 4 pasos básicos:

1. Se define un nodo inicial. Nótese que este nodo es el punto de partida, por lo que la distancia de este nodo a el mismo es de 0.
2. Dentro de las aristas a las que se tienen acceso, se toma la que tenga menor distancia.
3. A partir de la arista que se eligió en el paso 2, se expanden los nodos que estén adyacentes a ella, y en caso de que no hayan sido visitados previamente (o que

el valor de las aristas haya sido mayor que el que se tiene actualmente) se agrega al conjunto de caminos con la nueva distancia recorrida. De esta manera se actualizará cada que encuentre un camino mejor para cada nodo.

4. El proceso se repite hasta terminar con todos los caminos posibles.

Esto en un formato para Python sería algo así:

```
def shortest(self, v): # Dijkstra's algorithm
        q = [(0, v, ())] # arreglo "q" de las "Tuplas" de lo que se va a almacenar donde 0 es la distancia, v el nodo y () el "camino" hacia él
        dist = dict() #diccionario de distancias
        visited = set() #Conjunto de visitados
        while len(q) > 0: #mientras exista un nodo pendiente
            (l, u, p) = heappop(q) # Se toma la tupla con la distancia menor
            if u not in visited: # si no lo hemos visitado
                visited.add(u) #se agrega a visitados
                dist[u] = (l,u,list(flatten(p))[::-1] + [u])
     #agrega al diccionario
            p = (u, p) #Tupla del nodo y el camino
            for n in self.vecinos[u]: #Para cada hijo del nodo actual
                if n not in visited: #si no lo hemos visitado
                    el = self.E[(u,n)] #se toma la distancia del nodo actual hacia el nodo hijo
                    heappush(q, (l + el, n, p)) #Se agrega al arreglo "q" la distancia actual + la distancia hacia el nodo hijo, el nodo hijo n hacia donde se va, y el camino
         return dist #regresa el diccionario de distancias
```

Aplicación.

Se aplicará el algoritmo a 5 grafos diferentes, de diferentes tamaños, y se explicará cómo funciona el algoritmo con cada uno de ellos.

Grafo 1. (5 nodos/10 aristas)

Vértices:
1 2 3 4 5

Aristas:

(1, 2):	13	(1, 5):	33
(3, 2):	31	(1, 4):	22
(1, 3):	6	(4, 3):	8
(4, 5):	25	(4, 2):	2
(5, 2):	7	(5, 3):	1

Dijkstra = {1: (0, [1]), 2: (13, [1, 2]), 3: (6, [1, 3]), 4: (14, [1, 3, 4]), 5: (7, [1, 3, 5])}

En este caso, el algoritmo nos informa que, partiendo del nodo 1:

a) Para llegar al nodo 1, se recorre una distancia de 0, pasando por 1.
b) Para llegar al nodo 2, se recorre una distancia de 13, pasando por 1-2.
c) Para llegar al nodo 3, se recorre una distancia de 6, pasando por 1-3.
d) Para llegar al nodo 4, se recorre una distancia de 14, pasando por 1-3-4.
e) Para llegar al nodo 5, se recorre una distancia de 7, pasando por 1-3-5.

Grafo 2. (10 nodos/20 aristas)

Vértices:
1 2 3 4 5 6 7 8 9 10

Aristas:

(1, 3):	71	(10, 8):	6
(2, 8):	5	(10, 7):	18
(10, 6):	40	(7, 6):	29
(5, 4):	49	(5, 7):	8
(2, 1):	320	(8, 6):	37
(6, 2):	3	(4, 1):	87
(9, 4):	2	(10, 9):	19
(5, 1):	17	(9, 7):	22
(7, 2):	31	(3, 2):	54
(10, 4):	28	(4, 3):	16

Dijkstra = {1: (0, [1]), 2: (54, [1, 5, 7, 10, 8, 2]), 3: (65, [1, 5, 7, 9, 4, 3]), 4: (49, [1, 5, 7, 9, 4]), 5: (17, [1, 5]), 6: (54, [1, 5, 7, 6]), 7: (25, [1, 5, 7]), 8: (49, [1, 5, 7, 10, 8]), 9: (47, [1, 5, 7, 9]), 10: (43, [1, 5, 7, 10])}

En este caso, el algoritmo nos informa que, partiendo del nodo 1:

a) Para llegar al nodo 1, se recorre una distancia de 0, pasando por 1.
b) Para llegar al nodo 2, se recorre una distancia de 54, pasando por 1-5-7-10-8-2.
c) Para llegar al nodo 3, se recorre una distancia de 65, pasando por 1-5-7-9-4-3.
d) Para llegar al nodo 4, se recorre una distancia de 49, pasando por 1-5-7-9-4.
e) Para llegar al nodo 5, se recorre una distancia de 17, pasando por 1-5.
f) Para llegar al nodo 6, se recorre una distancia de 54, pasando por 1-5-7-6.
g) Para llegar al nodo 7, se recorre una distancia de 25, pasando por 1-5-7.
h) Para llegar al nodo 8, se recorre una distancia de 49, pasando por 1-5-7-10-8.
i) Para llegar al nodo 9, se recorre una distancia de 47, pasando por 1-5-7-9.
j) Para llegar al nodo 10, se recorre una distancia de 43, pasando por 1-5-7-10.

Grafo 3. (15 nodos/30 aristas)

Vértices:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Aristas:

(1, 3):	71	(7, 2):	31	(5, 7):	8
(2, 8):	5	(10, 4):	87	(8, 6):	37
(11, 2):	11	(10, 8):	6	(4, 1):	87
(10, 6):	40	(14, 5):	6	(10, 9):	19
(5, 4):	49	(11, 5):	1	(9, 7):	22
(2, 1):	320	(10, 7):	18	(3, 2):	54
(6, 2):	3	(13, 10):	13	(12, 11):	22
(9, 4):	2	(7, 6):	29	(15, 14):	19
(7, 11):	10	(1, 14):	20	(8, 14):	5
(5, 1):	17	(2, 13):	14	(4, 3):	16

Dijkstra = {1: (65, [3, 4, 9, 7, 5, 1]), 2: (48, [3, 4, 9, 10, 8, 2]), 3: (0, [3]), 4: (16, [3, 4]), 5: (48, [3, 4, 9, 7, 5]), 6: (51, [3, 4, 9, 10, 8, 2, 6]), 7: (40, [3, 4, 9, 7]), 8: (43, [3, 4, 9, 10, 8]), 9: (18, [3, 4, 9]), 10: (37, [3, 4, 9, 10]), 11: (49, [3, 4, 9, 7, 5, 11]), 12: (71, [3, 4, 9, 7, 5, 11, 12]), 13: (50, [3, 4, 9, 10, 13]), 14: (48, [3, 4, 9, 10, 8, 14]), 15: (67, [3, 4, 9, 10, 8, 14, 15])}

En este caso, el algoritmo nos informa que, partiendo del nodo 3:

a) Para llegar al nodo 1, se recorre una distancia de 65, pasando por 3, 4, 9, 7, 5, 1.
b) Para llegar al nodo 2, se recorre una distancia de 48, pasando por 3, 4, 9, 10, 8, 2.
c) Para llegar al nodo 3, se recorre una distancia de 0, pasando por 3.
d) Para llegar al nodo 4, se recorre una distancia de 16, pasando por 3, 4.
e) Para llegar al nodo 5, se recorre una distancia de 48, pasando por 3, 4, 9, 7, 5.
f) Para llegar al nodo 6, se recorre una distancia de 51, pasando por 3, 4, 9, 10, 8, 2, 6.
g) Para llegar al nodo 7, se recorre una distancia de 40, pasando por 3, 4, 9, 7.
h) Para llegar al nodo 8, se recorre una distancia de 43, pasando por 3, 4, 9, 10, 8.
i) Para llegar al nodo 9, se recorre una distancia de 18, pasando por 3, 4, 9.
j) Para llegar al nodo 10, se recorre una distancia de 37, pasando por 3, 4, 9, 10.

k) Para llegar al nodo 11, se recorre una distancia de 49, pasando por 3, 4, 9, 7, 5, 11.
l) Para llegar al nodo 12, se recorre una distancia de 71, pasando por 3, 4, 9, 7, 5, 11, 12.
m) Para llegar al nodo 13, se recorre una distancia de 50, pasando por 3, 4, 9, 10, 13.
n) Para llegar al nodo 14, se recorre una distancia de 48, pasando por 3, 4, 9, 10, 8, 14.
o) Para llegar al nodo 15, se recorre una distancia de 67, pasando por 3, 4, 9, 10, 8, 14, 15.

Grafo 4. (20 nodos/40 aristas)

Vértices:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Aristas:

(1, 3):	0	(5, 1):	10	(8, 6):	16
(18, 15):	15	(15, 5):	5	(4, 1):	18
(2, 8):	13	(19, 11):	19	(10, 9):	6
(11, 2):	1	(7, 2):	15	(9, 7):	8
(10, 6):	4	(10, 4):	7	(5, 4):	17
(17, 3):	1	(10, 8):	12	(19, 17):	13
(2, 1):	11	(14, 5):	11	(2, 20):	8
(15, 1):	2	(11, 5):	15	(14, 15):	18
(19, 15):	18	(10, 7):	3	(12, 11):	9
(6, 2):	3	(13, 10):	4	(9, 18):	6
(19, 14):	8	(7, 6):	5	(8, 14):	10
(9, 4):	14	(1, 14):	3	(4, 3):	3
(7, 11):	10	(2, 13):	14		
(16, 13):	20	(5, 7):	20		

Dijkstra = {1: (7, [5, 15, 1]), 2: (16, [5, 11, 2]), 3: (7, [5, 15, 1, 3]), 4: (10, [5, 15, 1, 3, 4]), 5: (0, [5]), 6: (19, [5, 11, 2, 6]), 7: (20, [5, 7]), 8: (20, [5, 15, 1, 14, 8]), 9: (23, [5, 15, 1, 3, 4, 10, 9]), 10: (17, [5, 15, 1, 3, 4, 10]), 11: (15, [5, 11]), 12: (24, [5, 11, 12]), 13: (21, [5, 15, 1, 3, 4, 10, 13]), 14: (10, [5, 15, 1, 14]), 15: (5, [5, 15]), 16: (41, [5, 15, 1, 3, 4, 10, 13, 16]), 17: (8, [5, 15, 1, 3, 17]), 18: (20, [5, 15, 18]), 19: (18, [5, 15, 1, 14, 19]), 20: (24, [5, 11, 2, 20])}

En este caso, el algoritmo nos informa que, partiendo del nodo 5:

a) Para llegar al nodo 1, se recorre una distancia de 7, pasando por 5, 15, 1.
b) Para llegar al nodo 2, se recorre una distancia de 16, pasando por 5, 11, 2.
c) Para llegar al nodo 3, se recorre una distancia de 7, pasando por 5, 15, 1, 3.
d) Para llegar al nodo 4, se recorre una distancia de 10, pasando por 5, 15, 1, 3, 4.
e) Para llegar al nodo 5, se recorre una distancia de 0, pasando por 5.
f) Para llegar al nodo 6, se recorre una distancia de 19, pasando por 5, 11, 2, 6.
g) Para llegar al nodo 7, se recorre una distancia de 20, pasando por 5, 7.
h) Para llegar al nodo 8, se recorre una distancia de 20, pasando por 5, 15, 1, 14, 8.
i) Para llegar al nodo 9, se recorre una distancia de 23, pasando por 5, 15, 1, 3, 4, 10, 9.
j) Para llegar al nodo 10, se recorre una distancia de 17, pasando por 5, 15, 1, 3, 4, 10.
k) Para llegar al nodo 11, se recorre una distancia de 15, pasando por 5, 11.
l) Para llegar al nodo 12, se recorre una distancia de 24, pasando por 5, 11, 12.
m) Para llegar al nodo 13, se recorre una distancia de 21, pasando por 5, 15, 1, 3, 4, 10, 13.
n) Para llegar al nodo 14, se recorre una distancia de 10, pasando por 5, 15, 1, 14.
o) Para llegar al nodo 15, se recorre una distancia de 5, pasando por 5, 15.
p) Para llegar al nodo 16, se recorre una distancia de 41, pasando por 5, 15, 1, 3, 4, 10, 13, 16.
q) Para llegar al nodo 17, se recorre una distancia de 8, pasando por 5, 15, 1, 3, 17.
r) Para llegar al nodo 18, se recorre una distancia de 20, pasando por 5, 15, 18.
s) Para llegar al nodo 19, se recorre una distancia de 18, pasando por 5, 15, 1, 14, 19.
t) Para llegar al nodo 20, se recorre una distancia de 24, pasando por 5, 11, 2, 20.

Grafo 5. (25 nodos/50 aristas)

Vértices:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

Aristas:

(15, 1):	2	(7, 5):	20	(4, 1):	18
(21, 8):	8	(14, 15):	18	(10, 9):	6
(21, 6):	20	(12, 11):	9	(9, 7):	8
(7, 22):	1	(3, 1):	0	(8, 25):	12
(6, 7):	5	(5, 14):	11	(24, 15):	8
(11, 5):	15	(16, 13):	20	(20, 5):	5
(10, 7):	3	(14, 8):	10	(11, 20):	18
(6, 10):	4	(18, 15):	15	(11, 25):	1
(19, 14):	8	(2, 1):	11	(23, 18):	6
(14, 1):	3	(17, 19):	13	(15, 19):	18
(18, 9):	6	(9, 4):	14	(1, 20):	2
(4, 10):	7	(7, 11):	10	(2, 13):	14
(2, 6):	3	(5, 1):	10	(2, 20):	8
(8, 2):	13	(3, 17):	1	(2, 25):	15
(4, 5):	17	(7, 24):	15	(3, 4):	3
(3, 23):	13	(11, 19):	19	(15, 5):	5
(10, 13):	4	(8, 6):	16		

Dijkstra = {1: (13, [7, 10, 4, 3, 1]), 2: (8, [7, 6, 2]), 3: (13, [7, 10, 4, 3]), 4: (10, [7, 10, 4]), 5: (20, [7, 5]), 6: (5, [7, 6]), 7: (0, [7]), 8: (21, [7, 6, 2, 8]), 9: (8, [7, 9]), 10: (3, [7, 10]), 11: (10, [7, 11]), 12: (19, [7, 11, 12]), 13: (7, [7, 10, 13]), 14: (16, [7, 10, 4, 3, 1, 14]), 15: (15, [7, 10, 4, 3, 1, 15]), 16: (27, [7, 10, 13, 16]), 17: (14, [7, 10, 4, 3, 17]), 18: (14, [7, 9, 18]), 19: (24, [7, 10, 4, 3, 1, 14, 19]), 20: (15, [7, 10, 4, 3, 1, 20]), 21: (25, [7, 6, 21]), 22: (1, [7, 22]), 23: (20, [7, 9, 18, 23]), 24: (15, [7, 24]), 25: (11, [7, 11, 25])}

En este caso, el algoritmo nos informa que, partiendo del nodo 7:

a) Para llegar al nodo 1, se recorre una distancia de 13, pasando por 7, 10, 4, 3, 1.
b) Para llegar al nodo 2, se recorre una distancia de 8, pasando por 7, 6, 2.
c) Para llegar al nodo 3, se recorre una distancia de 18, pasando por 7, 10, 4, 3.
d) Para llegar al nodo 4, se recorre una distancia de 10, pasando por 7, 10, 4.

e) Para llegar al nodo 5, se recorre una distancia de 20, pasando por 7, 5.
f) Para llegar al nodo 6, se recorre una distancia de 5, pasando por 7, 6.
g) Para llegar al nodo 7, se recorre una distancia de 0, pasando por 7.
h) Para llegar al nodo 8, se recorre una distancia de 21, pasando por 7, 6, 2, 8.
i) Para llegar al nodo 9, se recorre una distancia de 8, pasando por 7, 9.
j) Para llegar al nodo 10, se recorre una distancia de 3, pasando por 7, 10.
k) Para llegar al nodo 11, se recorre una distancia de 10, pasando por 7, 11.
l) Para llegar al nodo 12, se recorre una distancia de 19, pasando por 7, 11, 12, 12.
m) Para llegar al nodo 13, se recorre una distancia de 7, pasando por 7, 10, 13.
n) Para llegar al nodo 14, se recorre una distancia de 16, pasando por 7, 10, 4, 3, 1, 14.
o) Para llegar al nodo 15, se recorre una distancia de 15, pasando por 7, 10, 4, 3, 1, 15.
p) Para llegar al nodo 16, se recorre una distancia de 27, pasando por 7, 10, 13, 16.
q) Para llegar al nodo 17, se recorre una distancia de 14, pasando por 7, 10, 4, 3, 17.
r) Para llegar al nodo 18, se recorre una distancia de 14, pasando por 7, 9, 18.
s) Para llegar al nodo 19, se recorre una distancia de 24, pasando por 7, 10, 4, 3, 1, 14, 19.
t) Para llegar al nodo 20, se recorre una distancia de 15, pasando por 7, 10, 4, 3, 1, 20.
u) Para llegar al nodo 21, se recorre una distancia de 25, pasando por 7, 6, 21.
v) Para llegar al nodo 22, se recorre una distancia de 1, pasando por 7, 22.
w) Para llegar al nodo 23, se recorre una distancia de 20, pasando por 7, 9, 18, 23.
x) Para llegar al nodo 24, se recorre una distancia de 15, pasando por 7, 24.
y) Para llegar al nodo 25, se recorre una distancia de 11, pasando por 7, 11, 25.

Conclusiones.

Personalmente, el algoritmo de Dijkstra me ha resultado especialmente interesante, pues me pareció que tiene una implementación muy práctica y muy real. Me gustó la lógica que seguía y a pesar de que al momento de hacerlo aplicado en Python no lo vi tan claro como la teoría general, creo que es de los algoritmos más ingeniosos que se ha visto en la clase. Es un acercamiento importante entre Python y sus herramientas con nuestras vidas cotidianas.