REPORTE DE DIJKSTRA

MATEMÁTICAS COMPUTACIONALES

BENNY OZIEL ZAPATA VALLES

¿Qué es al algoritmo Dijsktra?

Este algoritmo es usado para encontrar la ruta más corta para llegar de nuestro nodo inicial a nuestro nodo deseado, todo esto dentro del concepto de grafos. Para que el algoritmo puede realizar lo anterior mencionado se necesita que, dentro del grafo, haya cierto conjunto de aristas con un peso, el cual, al compararse con otras decidiremos la recorrido con menor número de peso. Cabe destacar que este algoritmo sólo funciona con pesos positivos, si en nuestro grafo hay elemento negativos se necesitaría usar otro.

¿Cómo funciona?

Primero marca todos los vértices como no utilizados (o no visitados). Partiendo del nodo principal, del nodo origen, evaluamos las aristas para buscar el que tenga un camino más corto, es decir el que tenga un menor peso, se toma como punto medio para después volver a comparar para ver si podemos llegar más rápido de esta arista a los demás nodos. Una vez escogido el nodo más cercano, se repite el proceso, para así, siempre escoger las aristas con el menor peso posible. Esto se hace hasta llegar a nuestro nodo destino.

Como observación extra, podemos añadir que Dijsktra usa una técnica greedy, esta técnica está bajo la filosofía de *'Que para un camino sea óptimo, todos los caminos que contiene también deben ser óptimos'.*

En este caso, nuestro algoritmo de Dijsktra está elaborado de la siguiente manera:

```
from
heapq
import
heappop,
heappush
            def flatten(L):
                while len(L) > 0:
                    yield L[0]
                    L = L[1]
            class Grafo:

                def __init__(self):
                    self.V = set() # un conjunto
                    self.E = dict() # un mapeo de pesos de aristas
                    self.vecinos = dict() # un mapeo

                def agrega(self, v):
                    self.V.add(v)
                    if not v in self.vecinos: # vecindad de v
                        self.vecinos[v] = set() # inicialmente no tiene
            nada

                def conecta(self, v, u, peso=1):
                    self.agrega(v)
                    self.agrega(u)
                    self.E[(v, u)] = self.E[(u, v)] = peso # en ambos
            sentidos
                    self.vecinos[v].add(u)
                    self.vecinos[u].add(v)

                def complemento(self):
                    comp= Grafo()
                    for v in self.V:
                        for w in self.V:
                            if v != w and (v, w) not in self.E:
                                comp.conecta(v, w, 1)
```

```
        return comp
    def shortest(self, v): # Dijkstra's algorithm
        q = [(0, v, ())] # arreglo "q" de las "Tuplas" de lo
que se va a almacenar dondo 0 es la distancia, v el nodo y ()
el "camino" hacia el
        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 acutal hacia el nodo hijo
                    heappush(q, (l + el, n, p)) #Se agrega al
arreglo "q" la distancia actual mas la ditanacia hacia el nodo
hijo, el nodo hijo n hacia donde se va, y el camino
        return dist #regresa el diccionario de distancias


g= Grafo()
g.conecta('a','b', 1)
g.conecta('a','c', 1)
g.conecta('a','d', 1)
g.conecta('a','e', 1)
g.conecta('c','e', 1)
g.conecta('c','f', 10)
g.conecta('b','f', 1)
print(g.shortest('c'))
```

Para ejemplificar el funcionamiento de este algoritmo tenemos los siguientes ejemplos de grafos:

GRAFO 1

5 nodos- 10 aristas

Nodo	Peso	Final de Recorrido	Camino recorrido
T	4	t	s-c-t
S	0	s	s
B	2	b	s-b
C	1	c	s-c
D	1	d	s-c-d

Por simplicidad y falta de tiempo, para los siguientes grafos se utilizará una anotación diferente donde:

'Nodo': (Peso, Final, Camino o recorrido hecho).

GRAFO 2

<u>10 nodos- 20 aristas</u>

'a': (2, 'a', ['s', 'a']),

'c': (4, 'c', ['s', 'a', 'b', 'c']),

'b': (3, 'b', ['s', 'a', 'b']),

'e': (8, 'e', ['s', 'a', 'e']),

'd': (7, 'd', ['s', 'a', 'b', 'c', 'd']),

'g':(8, 'g', ['s', 'f', 'g']),

'f': (0, 'f', ['s', 'f']),

'h': (7, 'h', ['s', 'h']),

's': (0, 's', ['s'])

GRAFO 3

<u>15 NODOS- 30 ARISTAS</u>

'a': (21, 'a', ['e', 'i', 'o', 'a']),

'e': (0, 'e', ['e']),

'd': (31, 'd', ['e', 'i', 'o', 'a', 's', 'd']),

'g': (40, 'g', ['e', 'i', 'o', 'a', 's', 'd', 'f', 'g']),

'f': (35, 'f', ['e', 'i', 'o', 'a', 's', 'd', 'f']),

'i': (7, 'i', ['e', 'i']),

'h': (46, 'h', ['e', 'i', 'o', 'a', 's', 'd', 'f', 'g', 'h']),

'o': (13, 'o', ['e', 'i', 'o']),

'q': (16, 'q', ['e', 't', 'y', 'q']),

's': (28, 's', ['e', 'i', 'o', 'a', 's']),

'r': (12, 'r', ['e', 't', 'r']),

'u': (12, 'u', ['e', 'i', 'u']),

't': (7, 't', ['e', 't']),

'w': (13, 'w', ['e', 't', 'w']),

'y': (14, 'y', ['e', 't', 'y'])}

GRAFO 4

<u>20 NODOS- 40 ARISTAS</u>

'a': (10, 'a', ['e', 'q', 'a']),

'e': (0, 'e', ['e']),

'd': (12, 'd', ['e', 'l', 'd']),

'g': (6, 'g', ['e', 'g']),

'f': (11, 'f', ['e', 'l', 'f']),

'i': (13, 'i', ['e', 'l', 'i']),

'h': (7, 'h', ['e', 't', 'u', 'h']),

'k': (10, 'k', ['e', 'g', 'k']),

'j': (13, 'j', ['e', 't', 'u', 'h', 'j']),

'm': (21, 'm', ['e', 't', 'u', 'h', 'r', 'm']),

'l': (6, 'l', ['e', 'l']),

'o': (16, 'o', ['e', 't', 'u', 'h', 'o']),

'n': (16, 'n', ['e', 'l', 'f', 'n']),

'q': (7, 'q', ['e', 'q']),

'p': (13, 'p', ['e', 'g', 'p']),

's': (20, 's', ['e', 'g', 'p', 's']),

'r': (14, 'r', ['e', 't', 'u', 'h', 'r']),

'u': (7, 'u', ['e', 't', 'u']),

't': (7, 't', ['e', 't']),

'w': (16, 'w', ['e', 'l', 'f', 'w']),

'y': (17, 'y', ['e', 'l', 'f', 'y']

GRAFO 5

<u>25 NODOS – 50 ARISTAS</u>

'a': (18, 'a', ['e', 'v', 's', 'r', 'a']),

'c': (15, 'c', ['e', 'v', 'c']),

'b': (23, 'b', ['e', 'v', 's', 'r', 'b']),

'e': (0, 'e', ['e']),

'd': (17, 'd', ['e', 'v', 's', 'j', 'f', 'u', 'd']),

'g': (20, 'g', ['e', 'v', 's', 'j', 'f', 'g']),

'f': (12, 'f', ['e', 'v', 's', 'j', 'f']),

'i': (23, 'i', ['e', 'v', 's', 'r', 'i']),

'h': (23, 'h', ['e', 'v', 's', 'j', 'f', 'u', 'd', 'h']),

'k': (27, 'k', ['e', 'v', 's', 'j', 'f', 't', 'k']),

'j': (12, 'j', ['e', 'v', 's', 'j']),

'l': (19, 'l', ['e', 'v', 's', 'r', 'q', 'l']),

'o': (17, 'o', ['e', 'v', 's', 'j', 'f', 'o']),

'q': (17, 'q', ['e', 'v', 's', 'r', 'q']),

'p': (22, 'p', ['e', 'v', 's', 'j', 'f', 'u', 'd', 'p']),

's': (8, 's', ['e', 'v', 's']),

'r': (14, 'r', ['e', 'v', 's', 'r']),

'u': (17, 'u', ['e', 'v', 's', 'j', 'f', 'u']),

't': (17, 't', ['e', 'v', 's', 'j', 'f', 't']),

'w': (26, 'w', ['e', 'v', 's', 'r', 'i', 'w']),

'v': (8, 'v', ['e', 'v']),

```
 'y': (20, 'y', ['e', 'v', 's', 'j', 'y']),
'x': (31, 'x', ['e', 'v', 'c', 'z', 'x']),
 'z': (26, 'z', ['e', 'v', 'c', 'z'])
```