Reporte de practica: Algoritmo Dijkstra

Nombre: Diana Carolina Lozoya Pérez **Matricula:**1725012

Materia: Matemáticas Computacionales **Grupo:**001 **Fecha:** 20/10/17

Resumen: Este reporte tiene como objetivo mostrar y describir un algoritmo llamado Dijkstra, posteriormente hacer 5 pruebas de este algoritmo y mostrar los resultados en forma de tabla, pues este algoritmo es muy importante para aquellos que desean solucionar problemas de rutas o caminos más favorables bajo distancia o ciertas variables.

I. ¿Qué es el algoritmo de Dijkstra?

Este algoritmo se le atribuye a Edsger Wybe Dijkstra de origen holandés, quien anuncio su algoritmo de caminos mínimos en 1959, hoy conocido como ‘’el algoritmo de Dijkstra’’.

Este algoritmo es eficiente para la determinación más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Este se encarga de formar un camino a partir de otro ya existente. Además, nótese que en este algoritmo los ‘’pesos’’ deben de ser positivos.

El algoritmo Dijkstra trabaja por etapas bajo el principio de la optimalidad, tomando en cada etapa la mejor solución sin considerar consecuencias futuras; donde el óptimo encontrado puede modificarse posteriormente si surge una solución mejor. El algoritmo devuelve en realidad el peso mínimo, no el camino mínimo propiamente dicho.

II. Funcionamiento de Dijkstra

Cabe recordar que este algoritmo utiliza un grafo para su funcionamiento, por lo que previamente se debe de tener definido un grafo.

a) Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de esos vértices evaluaremos sus adyacentes, como Dijkstra usa una técnica ***greedy*** – *La técnica greedy utiliza el principio de que para que un camino sea óptimo, todos los caminos que contiene también deben ser óptimos*- entre todos los vértices adyacentes, buscamos el que esté más cerca de nuestro punto origen, lo tomamos como punto intermedio y vemos si podemos llegar más rápido a través de este vértice a los demás.
b) Después escogemos al siguiente más cercano (con las distancias ya actualizadas) y repetimos el proceso. Esto lo hacemos hasta que el vértice no utilizado más cercano sea nuestro destino. Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vértice se le conoce como *relajación*.

Nota:

Dijkstra es muy similar a BFS, si recordamos BFS usaba una Cola para el recorrido para el caso de Dijkstra usaremos una Cola de Prioridad o Heap, este Heap debe tener la propiedad de Min-Heap es decir cada vez que extraiga un elemento del Heap me debe devolver el de menor valor, en nuestro caso dicho valor será el peso acumulado en los nodos.

III. Pseudocódigo de Dijkstra

Considérese la distancia[i] como la distancia más corta del vértice origen ingresado al vértice i.

Entonces, podemos describir el pseudocódigo como:

```
método Dijkstra(Grafo, origen):
    creamos una cola de prioridad Q
    agregamos origen a la cola de prioridad Q
    mientras Q no este vacío:
        sacamos un elemento de la cola Q llamado u
        si u ya fue visitado continuo sacando elementos de Q
        marcamos como visitado u
        para cada vértice v adyacente a u en el Grafo:
            sea w el peso entre vértices (u, v)
            si v no ha sido visitado:
                relajación (u, v, w)

método relajación( actual , adyacente , peso ):
    si distancia [ actual ] + peso < distancia[ adyacente ]
       distancia [ adyacente ] = distancia[ actual ] + peso
       agregamos adyacente a la cola de prioridad Q
```

IV. Código de Dijkstra en Python

```python
def shortest(self, v):
    q = [(0, v, ())]
    dist = dict()
    visited = set()
    while len(q) > 0:
        (l, u, p) = heappop(q)
        if u not in visited:
            visited.add(u)
            dist[u] = (l,u,list(flatten(p))[::-1] + [u])
        p = (u, p)
        for n in self.vecinos[u]:
            if n not in visited:
                el = self.E[(u,n)]
                heappush(q, (l + el, n, p))
    return dist
```

V. Muestra de funcionamiento de Dijkstra

A continuación, se muestra el uso de 5 grafos para probar el funcionamiento de Dijkstra, dentro de cada prueba se mencionará que consideramos para realizarla así como el número de nodos y aristas, además del código que se uso y el resultado que se obtuvo al ejecutarlo y por último la interpretación de este resultado en forma de tabla.

a) Prueba 1

Considérese como el nodo inicial a 'a'.

Programa:

```
#5 nodos,10 aristas
print("Primer prueba 5 nodos, 10 aristas")

g=Grafo()
g.conecta('a','b',2)
g.conecta('a','c',4)
g.conecta('a','d',6)
g.conecta('a','e',8)
g.conecta('a','b',10)
g.conecta('b','c',12)
g.conecta('b','d',14)
g.conecta('b','e',16)
g.conecta('c','e',18)
g.conecta('c','d',20)

print(g.shortest('a'))
```

Resultado del Programa:

```
Primer prueba 5 nodos, 10 aristas
{'a': (0, 'a', ['a']), 'c': (4, 'c', ['a', 'c']), 'd': (6, 'd', ['a', 'd']), 'e': (8, 'e', ['a', 'e']),
 'b': (10, 'b', ['a', 'b'])}
```

Tabla de Interpretación de Resultados:

Nodo	Camino	Distancia
a	a-a	0
c	a-c	4
d	a-d	6
e	a-e	8
b	a-b	10

b) Prueba 2

Considérese como el nodo inicial a ‘a’.

Programa:

```
#10 nodos,20 aristas
print("Segunda prueba 10 nodos,20 aristas")

f=Grafo()
f.conecta('a','b',3)
f.conecta('a','c',9)
f.conecta('a','d',12)
f.conecta('a','e',7)
f.conecta('a','f',4)
f.conecta('a','g',3)
f.conecta('a','h',2)
f.conecta('a','i',1)
f.conecta('a','j',5)
f.conecta('b','c',11)
f.conecta('b','d',8)
f.conecta('c','d',9)
f.conecta('d','e',4)
f.conecta('e','f',7)
f.conecta('f','g',6)
f.conecta('g','h',1)
f.conecta('h','e',10)
f.conecta('i','j',14)
f.conecta('j','h',4)
f.conecta('j','g',5)
f.conecta('c','e',5)

print(f.shortest('a'))
```

Resultado del Programa:

```
Segunda prueba 10 nodos,20 aristas
{'a': (0, 'a', ['a']), 'i': (1, 'i', ['a', 'i']), 'h': (2, 'h', ['a', 'h']), 'b': (3, 'b', ['a', 'b']),
 'g': (3, 'g', ['a', 'g']), 'f': (4, 'f', ['a', 'f']), 'j': (5, 'j', ['a', 'j']), 'e': (7, 'e', ['a', '
e']), 'c': (9, 'c', ['a', 'c']), 'd': (11, 'd', ['a', 'b', 'd'])}
```

Tabla de Interpretación de Resultados:

Nodo	Camino	Distancia
a	a-a	0
i	a-i	1
h	a-h	2
b	a-b	3
g	a-g	3
f	a-f	4
j	a-j	5
e	a-e	7
c	a-c	9
d	b-d	11

c) *Prueba 3*

Considérese como el nodo inicial a ‘a’.

Programa:

```
#15 nodos,30 aristas
print("Tercera prueba 15 nodos, 30 aristas")
h=Grafo()
h.conecta('a','b',4)
h.conecta('a','c',8)
h.conecta('a','d',7)
h.conecta('a','e',5)
h.conecta('a','f',3)
h.conecta('a','g',2)
h.conecta('a','h',1)
h.conecta('b','i',9)
h.conecta('c','j',10)
h.conecta('d','k',11)
h.conecta('e','l',8)
h.conecta('f','m',4)
h.conecta('j','n',7)
h.conecta('a','ñ',9)
h.conecta('i','k',11)
h.conecta('b','b',3)
h.conecta('c','e',5)
h.conecta('d','i',6)
h.conecta('e','h',7)
h.conecta('f','c',10)
h.conecta('g','e',12)
h.conecta('h','f',15)
h.conecta('i','b',9)
h.conecta('j','c',3)
h.conecta('k','j',2)
h.conecta('l','f',1)
h.conecta('m','i',5)
h.conecta('n','k',8)
h.conecta('ñ','d',9)
h.conecta('a','m',13)

print(h.shortest('a'))
```

Resultado del Programa:

```
Tercera prueba 15 nodos, 30 aristas
{'a': (0, 'a', ['a']), 'h': (1, 'h', ['a', 'h']), 'g': (2, 'g', ['a', 'g']), 'f': (3, 'f', ['a', 'f']),
 'b': (4, 'b', ['a', 'b']), 'l': (4, 'l', ['a', 'f', 'l']), 'e': (5, 'e', ['a', 'e']), 'd': (7, 'd', ['
a', 'd']), 'm': (7, 'm', ['a', 'f', 'm']), 'c': (8, 'c', ['a', 'c']), 'ñ': (9, 'ñ', ['a', 'ñ']), 'j': (
11, 'j', ['a', 'c', 'j']), 'i': (12, 'i', ['a', 'f', 'm', 'i']), 'k': (13, 'k', ['a', 'c', 'j', 'k']),
'n': (18, 'n', ['a', 'c', 'j', 'n'])}
```

Tabla de Interpretación de Resultados:

Nodo	Camino	Distancia
a	a-a	0
h	a-h	1
g	a-g	2
f	a-f	3
b	a-b	7
l	a-f-l	4
e	a-e	5
d	a-d	7
m	a-f-m	7

c	a-c	8
ñ	a-ñ	9
j	a-c-j	11
i	a-f-m-i	12
k	a-c-j-k	13
n	a-c-j-n	18

d) *Prueba 4*

Considérese como el nodo inicial a ‘a’.

Programa:

```
#20 nodos, 40 aristas)
print("Cuarta prueba 20 nodos, 40 aristas")

m=Grafo()
m.conecta('a','b',5)
m.conecta('a','c',2)
m.conecta('a','d',3)
m.conecta('c','e',4)
m.conecta('a','f',2)
m.conecta('a','g',5)
m.conecta('g','h',9)
m.conecta('a','i',10)
m.conecta('k','j',21)
m.conecta('a','k',34)
m.conecta('a','l',7)
m.conecta('c','m',8)
m.conecta('r','n',9)
m.conecta('a','ñ',12)
m.conecta('b','o',11)
m.conecta('c','p',2)
m.conecta('d','q',1)
m.conecta('e','r',5)
m.conecta('f','s',6)
m.conecta('g','m',7)
m.conecta('h','n',9)
m.conecta('i','l',11)
m.conecta('j','o',10)
m.conecta('m','p',12)
m.conecta('o','q',2)
m.conecta('s','r',3)
m.conecta('l','s',4)
m.conecta('k','e',5)
m.conecta('j','f',4)
m.conecta('b','g',7)
m.conecta('e','h',8)
m.conecta('d','i',12)
m.conecta('r','j',2)
m.conecta('s','k',5)
m.conecta('p','l',6)

m.conecta('n','m',7)
m.conecta('g','n',8)
m.conecta('f','ñ',9)
m.conecta('e','o',10)
m.conecta('s','p',1)

print(m.shortest('a'))
```

Resultados del Programa:

```
Cuarta prueba 20 nodos, 40 aristas
{'a': (0, 'a', ['a']), 'c': (2, 'c', ['a', 'c']), 'f': (2, 'f', ['a', 'f']), 'd': (3, 'd', ['a', 'd']),
 'p': (4, 'p', ['a', 'c', 'p']), 'q': (4, 'q', ['a', 'd', 'q']), 'b': (5, 'b', ['a', 'b']), 'g': (5, 'g
', ['a', 'g']), 's': (5, 's', ['a', 'c', 'p', 's']), 'e': (6, 'e', ['a', 'c', 'e']), 'j': (6, 'j', ['a'
, 'f', 'j']), 'o': (6, 'o', ['a', 'd', 'q', 'o']), 'l': (7, 'l', ['a', 'l']), 'r': (8, 'r', ['a', 'f',
'j', 'r']), 'i': (10, 'i', ['a', 'i']), 'k': (10, 'k', ['a', 'c', 'p', 's', 'k']), 'm': (10, 'm', ['a',
 'c', 'm']), 'ñ': (11, 'ñ', ['a', 'f', 'ñ']), 'n': (13, 'n', ['a', 'g', 'n']), 'h': (14, 'h', ['a', 'c'
, 'e', 'h'])}
```

Tabla de Interpretación de Resultados:

Nodo	Camino	Distancia
a	a-a	0
c	a-c	2
f	a-f	2
d	a-d	3
p	a-c-p	4
q	a-d-q	4
b	a-b	5
g	a-g	5
s	a-c-p-s	5
e	a-c-e	6
j	a-f-j	6
o	a-d-q-o	6
l	a-l	7
r	a-f-j-r	8
i	a-i	10
k	a-c-p-s-k	10
m	a-c-m	10
ñ	a-f-ñ	11
n	a-g-n	13
h	a-c-e-h	14

e) *Prueba 5*

Considérese como el nodo inicial a ‘a’.

Programa:

```
#25 nodos, 50 aristas
print("Quinta prueba 50 nodos, 50 aristas")

r=Grafo()
r.conecta('a','b',5)
r.conecta('a','c',1)
r.conecta('a','d',2)
r.conecta('a','e',53)
r.conecta('a','f',4)
r.conecta('a','g',7)
r.conecta('a','h',9)
r.conecta('a','i',12)
r.conecta('a','j',8)
r.conecta('a','k',9)
r.conecta('a','l',2)
r.conecta('a','m',3)
r.conecta('b','n',5)
r.conecta('a','ñ',8)
r.conecta('s','o',6)
r.conecta('d','p',51)
r.conecta('d','q',4)
r.conecta('p','r',7)
r.conecta('o','s',9)
r.conecta('x','t',3)
r.conecta('x','u',4)
r.conecta('r','v',57)
r.conecta('k','w',7)
r.conecta('s','x',56)
r.conecta('j','b',6)
r.conecta('e','c',3)
r.conecta('b','d',2)
r.conecta('a','e',51)
r.conecta('a','f',1)
r.conecta('x','g',2)
r.conecta('w','h',4)
r.conecta('v','i',5)
r.conecta('u','j',8)
r.conecta('s','k',10)

r.conecta('r','l',11)
r.conecta('q','m',9)
r.conecta('p','n',12)
r.conecta('o','ñ',13)
r.conecta('n','o',14)
r.conecta('m','p',5)
r.conecta('l','q',8)
r.conecta('k','r',9)
r.conecta('j','s',2)
r.conecta('i','t',1)
r.conecta('h','u',4)
r.conecta('g','v',7)
r.conecta('f','w',3)
r.conecta('e','x',2)
r.conecta('d','e',20)
r.conecta('c','b',11)

print(r.shortest('a'))
```

Resultados del Programa:

```
Quinta prueba 50 nodos, 50 aristas
{'a': (0, 'a', ['a']), 'c': (1, 'c', ['a', 'c']), 'f': (1, 'f', ['a', 'f']), 'd': (2, 'd', ['a', 'd']),
 'l': (2, 'l', ['a', 'l']), 'm': (3, 'm', ['a', 'm']), 'b': (4, 'b', ['a', 'd', 'b']), 'e': (4, 'e', ['
a', 'c', 'e']), 'w': (4, 'w', ['a', 'f', 'w']), 'q': (6, 'q', ['a', 'd', 'q']), 'x': (6, 'x', ['a', 'c'
, 'e', 'x']), 'g': (7, 'g', ['a', 'g']), 'h': (8, 'h', ['a', 'f', 'w', 'h']), 'j': (8, 'j', ['a', 'j'])
, 'p': (8, 'p', ['a', 'm', 'p']), 'ñ': (8, 'ñ', ['a', 'ñ']), 'k': (9, 'k', ['a', 'k']), 'n': (9, 'n', [
'a', 'd', 'b', 'n']), 't': (9, 't', ['a', 'c', 'e', 'x', 't']), 'i': (10, 'i', ['a', 'c', 'e', 'x', 't'
, 'i']), 's': (10, 's', ['a', 'j', 's']), 'u': (10, 'u', ['a', 'c', 'e', 'x', 'u']), 'r': (13, 'r', ['a
', 'l', 'r']), 'v': (14, 'v', ['a', 'g', 'v']), 'o': (19, 'o', ['a', 'j', 's', 'o'])}
```

Tabla de Interpretación de Resultados:

Nodo	Camino	Distancia
a	a-a	0
c	a-c	1
f	a-f	1
d	a-d	2
l	a-l	2
m	a-m	3
b	a-d-b	4
e	a-c-e	4
w	a-f-w	4
q	a-d-q	6
x	a-c-e-x	6
g	a-g	7
h	a-f-w-h	8
j	a-j	8
p	a-m-p	8
ñ	a-ñ	8
k	a-k	9
n	a-d-b-n	9
t	a-c-e-x-t	9
i	a-c-e-x-t-i	10
s	a-j-s	10
u	a-c-e-x-u	10
r	a-l-r	13
v	a-g-v	14
o	a-j-s-o	19

VI. Conclusión

De manera general, el algoritmo de Dijkstra me pareció de mucha utilidad en la rama de las matemáticas ya que lo podemos aplicar en lo que ya conocíamos (los grafos), además de que este sencillo algoritmo lo podemos utilizar principalmente para resolver algún problema sobre una distribución de productos bajo una red de establecimientos comerciales, correos postales, encontrar el camino más corto de un lugar a otro lugar, etc.

Me parece que es una herramienta que a nosotros como matemáticos nos es de mucha ayuda, ya que principalmente para resolver este tipo de problemas, nosotros optaríamos por comenzar a hacer combinaciones y probar cada una de ellas, posteriormente cerciorarnos de que es la opción más optima y por ultimo exponer los resultados, sin embargo, el hecho de que un matemático cuente con esto, representa una gran ayuda, ya que nosotros podemos hacer todo ese trabajo en menos tiempo del que normalmente nos tomaría, es decir, solo en cuestión de segundos sabríamos cual es nuestra ruta o camino más corto.

Por lo tanto, considero que este algoritmo nos será de gran ayuda más si nos vamos a dedicar a las matemáticas aplicadas, pues este es un claro ejemplo de la combinación de la lógica matemática con la programación. Cabe destacar que debemos de seguir innovando cada vez más este tipo de algoritmos, de esta manera lograremos facilitar muchas cosas y ayudaremos a que el avance de la tecnología sea más rápido.

De manera personal, me gusto trabajar con Dijkstra, aunque al principio fue confuso por la manera en la que debía de estructurarlo, el hecho de haberlo comprendido me facilito bastante poder entender y analizar el código del mismo.

Como observación adicional, el código completo de todo el programa que fue utilizado para elaborar este reporte se encuentra en el repositorio con el nombre de ‘’Código Dijkstra’’.