UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FACULTAD DE CIENCIAS FÍSICO – MATEMÁTICAS

MATEMÁTICAS COMPUTACIONALES

“ALGORITMO DE DIJKSTRA”

Nombre: Diego Rubio Romero

Matrícula: 1738544

Carrera: Lic. En Matemáticas

Profesor: Lic. José Anastacio Hernández Saldaña

Grupo: 002

Fecha: Viernes 20 de Octubre del 2017

Lugar: Ciudad Universitaria

Introducción

En este reporte se analizará el algoritmo de Dijkstra, empezando por su definición, como funciona, y además veremos una serie de resultados, con varias prácticas hechas en clase.

Descripción del algoritmo

El algoritmo de Dijkstra consiste en determinar la ruta más corta y eficiente, en un grafo, desde un nodo inicial (origen, además con peso igual a cero) hacia los demás nodos, fijando un nodo y comparando con sus hijos la distancia entre ellos, tomando las distancias más cortas entre cada comparación, y así hasta terminar de comparar la distancia entre cada nodo posterior a ellos, fijando así el último nodo como un nodo final, para después éste determinar la ruta más corta.

Pseudo-Código:

Considerar distancia[i] como la distancia mas corta del vértice origen ingresado al vértice i.

```
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 ah sido visitado:
               Relajacion( u , v , w )

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

Código Fuente en Python:

```
from heapq import heappop, heappush
from copy import deepcopy


def flatten(L):
    while len(L) > 0:
        yield L[0]
        L = L[1]


class Grafo:


    def __init__(self):
        self.V = set()
        self.E = dict()
        self.vecinos = dict()


    def agrega(self, v):
        self.V.add(v)
        if not v in self.vecinos:
            self.vecinos[v] = set()


    def conecta(self, v, u, peso=1):
        self.agrega(v)
        self.agrega(u)
        self.E[(v, u)] = self.E[(u, v)] = peso
        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):
    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
```

Prueba del Algoritmo

A continuación analizaremos los resultados obtenidos, entre 5 Grafos con diferencia de 5 nodos entre cada uno y el doble número de aristas para cada uno, y a partir de aquí veremos lo que sucede con cada uno de ellos.

Grafo con 5 nodos y 10 aristas

```
g5= Grafo()
g5.conecta(1, 2, 16)
g5.conecta(1, 3,  49)
g5.conecta(1, 4, 12)
g5.conecta(1, 5,35)
g5.conecta(2, 5, 37)
g5.conecta(2, 4, 18)
g5.conecta(2, 3, 47)
g5.conecta(3, 4, 40)
g5.conecta(3, 5,  15)
g5.conecta(4, 5, 46)
print(g5.shortest(1))
```

Resultado:

1: (0, 1, [1]),

4: (12, 4, [1, 4]),

2: (16, 2, [1, 2]),

5: (35, 5, [1, 5]),

3: (49, 3, [1, 3])

Grafo con 10 nodos y 20 aristas

```
g10= Grafo()
g10.conecta(1, 10,   25)
g10.conecta(1, 2,    13)
g10.conecta(1, 7,    26)
g10.conecta(1, 9,    20)
g10.conecta(1, 3,   28)
g10.conecta(2, 4,    37)
g10.conecta(2, 10,   37)
g10.conecta(3, 6,    10)
g10.conecta(3, 5,    48)
g10.conecta(3, 7,    13)
g10.conecta(3, 10,   14)
g10.conecta(4, 7,    19)
g10.conecta(4, 9,    11)
g10.conecta(4, 10,   46)
g10.conecta(4, 5,    43)
g10.conecta(5, 10,   26)
g10.conecta(6, 10,   8)
g10.conecta(6, 7,    40)
g10.conecta(7, 8,    36)
g10.conecta(8, 10,  32)
print(g10.shortest(1))
```

Resultado:

1: (0, 1, [1]),

2: (13, 2, [1, 2]),

9: (20, 9, [1, 9]),

10: (25, 10, [1, 10]),

7: (26, 7, [1, 7]),

3: (28, 3, [1, 3]),

4: (31, 4, [1, 9, 4]),

6: (33, 6, [1, 10, 6]),

5: (51, 5, [1, 10, 5]),

8: (57, 8, [1, 10, 8])

Grafo con 15 nodos y 30 aristas

```
g15= Grafo()
g15.conecta(1, 15,   24)
g15.conecta(1, 11,   22)
g15.conecta(1, 3,    45)
g15.conecta(2, 7,    42)
g15.conecta(2, 10,   31)
g15.conecta(3, 6,    14)
g15.conecta(3, 13,   17)
g15.conecta(3, 15,   13)
g15.conecta(4, 14,   11)
g15.conecta(4, 13,   2)
g15.conecta(4, 9,   18)
g15.conecta(5, 15,   4)
g15.conecta(5, 11,   29)
g15.conecta(5, 7,   6)
g15.conecta(6, 11,   33)
g15.conecta(7, 13,   26)
g15.conecta(7, 10,   44)
g15.conecta(7, 11,   50)
g15.conecta(8, 14,   2)
g15.conecta(8, 13,   11)
g15.conecta(9, 15,   34)
g15.conecta(9, 14,   44)
```

```
g15.conecta(9, 10,   49)
g15.conecta(10, 15,  8)
g15.conecta(10, 14,  49)
g15.conecta(10, 13,  8)
g15.conecta(11, 15,  50)
g15.conecta(12, 15,  23)
g15.conecta(13, 14,  27)
g15.conecta(14, 15,  47)
print(g15.shortest(1))
```

Resultado:

1: (0, 1, [1]),

11: (22, 11, [1, 11]),

15: (24, 15, [1, 15]),

5: (28, 5, [1, 15, 5]),

10: (32, 10, [1, 15, 10]),

7: (34, 7, [1, 15, 5, 7]),

3: (37, 3, [1, 15, 3]),

13: (40, 13, [1, 15, 10, 13]),

4: (42, 4, [1, 15, 10, 13, 4]),

12: (47, 12, [1, 15, 12]),

6: (51, 6, [1, 15, 3, 6]),

8: (51, 8, [1, 15, 10, 13, 8]),

14: (53, 14, [1, 15, 10, 13, 4, 14]),

9: (58, 9, [1, 15, 9]),

2: (63, 2, [1, 15, 10, 2])

Grafo con 20 nodos y 40 aristas

```
g20= Grafo()
g20.conecta(1, 8,    23)
g20.conecta(1, 18,   1)
g20.conecta(2, 16,   18)
g20.conecta(2, 3,    19)
g20.conecta(2, 18,   15)
g20.conecta(3, 18,   50)
g20.conecta(3, 6,    8)
g20.conecta(4, 8,    43)
g20.conecta(4, 17,   32)
g20.conecta(5, 18,   1)
g20.conecta(6, 14,   43)
g20.conecta(6, 20,   45)
g20.conecta(7, 8,    13)
g20.conecta(7, 18,   9)
g20.conecta(7, 17,   42)
g20.conecta(8, 11,   21)
g20.conecta(8, 17,   37)
g20.conecta(9, 15,   20)
g20.conecta(9, 18,   11)
g20.conecta(9, 16,   2)
g20.conecta(10, 11,  14)
g20.conecta(10, 19,  29)
g20.conecta(11, 19,  26)
g20.conecta(11, 17,  45)
g20.conecta(11, 20,  50)
g20.conecta(12, 13,  37)
g20.conecta(12, 15,  12)
g20.conecta(13, 17,  5)
g20.conecta(13, 14,  15)
g20.conecta(14, 19,  3)
```

```
g20.conecta(14, 18,  28)
g20.conecta(15, 20,  11)
g20.conecta(15, 17,  43)
g20.conecta(15, 18,  3)
g20.conecta(16, 18,  16)
g20.conecta(16, 20,  48)
g20.conecta(17, 19,  1)
g20.conecta(18, 20,  50)
g20.conecta(18, 19,  4)
g20.conecta(19, 20,  12)
print(g20.shortest(1))
```

Resultado:

```
1: (0, 1, [1]),
18: (1, 18, [1, 18]),
5: (2, 5, [1, 18, 5]),
15: (4, 15, [1, 18, 15]),
19: (5, 19, [1, 18, 19]),
17: (6, 17, [1, 18, 19, 17]),
14: (8, 14, [1, 18, 19, 14]),
7: (10, 7, [1, 18, 7]),
13: (11, 13, [1, 18, 19, 17, 13]),
9: (12, 9, [1, 18, 9]),
16: (14, 16, [1, 18, 9, 16]),
20: (15, 20, [1, 18, 15, 20]),
2: (16, 2, [1, 18, 2]),
12: (16, 12, [1, 18, 15, 12]),
8: (23, 8, [1, 8]),
11: (31, 11, [1, 18, 19, 11]),
10: (34, 10, [1, 18, 19, 10]),
```

3: (35, 3, [1, 18, 2, 3]),

4: (38, 4, [1, 18, 19, 17, 4]),

6: (43, 6, [1, 18, 2, 3, 6])

Grafo con 25 nodos y 50 aristas

```
g25= Grafo()
g25.conecta(1, 2, 22)
g25.conecta(1, 21,   12)
g25.conecta(1, 20,   10)
g25.conecta(2, 15,   36)
g25.conecta(2, 20,   19)
g25.conecta(2, 25,   23)
g25.conecta(3, 13,   42)
g25.conecta(3, 15,   21)
g25.conecta(4, 11,   47)
g25.conecta(4, 14,   13)
g25.conecta(4, 6,    8)
g25.conecta(5, 25,   37)
g25.conecta(5, 8,    49)
g25.conecta(6, 13,   43)
g25.conecta(6, 19,   3)
g25.conecta(6, 23,   27)
g25.conecta(7, 18,   46)
g25.conecta(7, 12,   46)
g25.conecta(7, 11,   21)
g25.conecta(8, 17,   39)
g25.conecta(8, 13,   25)
g25.conecta(10, 24,  33)
g25.conecta(10, 18,  36)
g25.conecta(10, 15,  37)
g25.conecta(11, 13,  50)
```

```
g25.conecta(11, 19,  35)
g25.conecta(12, 13,  43)
g25.conecta(12, 20,  15)
g25.conecta(12, 25,  38)
g25.conecta(13, 22,  9)
g25.conecta(13, 15,  6)
g25.conecta(14, 22,  4)
g25.conecta(14, 16,  15)
g25.conecta(15, 17,  12)
g25.conecta(15, 20,  11)
g25.conecta(16, 25,  38)
g25.conecta(17, 24,  20)
g25.conecta(18, 19,  16)
g25.conecta(18, 20,  45)
g25.conecta(18, 21,  2)
g25.conecta(19, 24,  15)
g25.conecta(20, 23,  25)
g25.conecta(20, 21,  22)
g25.conecta(21, 23,  7)
g25.conecta(21, 22,  47)
g25.conecta(21, 25,  26)
g25.conecta(22, 23,  30)
g25.conecta(22, 25,  15)
g25.conecta(23, 25,  31)
g25.conecta(24, 25,  7)
print(g25.shortest(1))
```

Resultado:

```
1: (0, 1, [1]),
20: (10, 20, [1, 20]),
21: (12, 21, [1, 21]),
```

18: (14, 18, [1, 21, 18]),

23: (19, 23, [1, 21, 23]),

15: (21, 15, [1, 20, 15]),

2: (22, 2, [1, 2]),

12: (25, 12, [1, 20, 12]),

13: (27, 13, [1, 20, 15, 13]),

19: (30, 19, [1, 21, 18, 19]),

6: (33, 6, [1, 21, 18, 19, 6]),

17: (33, 17, [1, 20, 15, 17]),

22: (36, 22, [1, 20, 15, 13, 22]),

25: (38, 25, [1, 21, 25]),

14: (40, 14, [1, 20, 15, 13, 22, 14]),

4: (41, 4, [1, 21, 18, 19, 6, 4]),

3: (42, 3, [1, 20, 15, 3]),

24: (45, 24, [1, 21, 18, 19, 24]),

10: (50, 10, [1, 21, 18, 10]),

8: (52, 8, [1, 20, 15, 13, 8]),

16: (55, 16, [1, 20, 15, 13, 22, 14, 16]),

7: (60, 7, [1, 21, 18, 7]),

11: (65, 11, [1, 21, 18, 19, 11]),

5: (75, 5, [1, 21, 25, 5])

Conclusión

A pesar de que este existen diferentes algoritmos para hallar la ruta más corta entre los nodos de un grafo, el algoritmo de Dijkstra es muy eficiente y fácil de entender su funcionamiento (más no implementar en Python), pude observar que su funcionamiento consta de fijar un nodo inicial, y a partir de ahí preguntarse, ¿Quiénes son sus nodos adyacentes?, para después irse recorriendo a cada uno de estos, si es que hay, en dado caso de que los haya, se fija otro nodo y se repite el proceso, siempre y cuando exista una distancia más corta está será reemplazada, y así hasta terminar con todos los nodos, obtendremos la distancia más corta.