Reporte de Dijkstra

Haydee Judith Arriaga Ponce

Matricula: 1659539

l.judithmat@gmail.com

https://github.com/haydee10arriaga/1659539_Mat.Com

20 de octubre de 2017

En el presente reporte se tiene el objetivo de a partir de un análisis mostrar el algoritmo Dijkstra implementado en python, para ver su comportamiento en grafos por lo cual realizamos 5 pruebas mostrando los resultados de ahí que se toma conciencia de lo importante que es para solucionar caminos más favorables en cuestión de distancias.

1. Dijkstra descripción

El algoritmo de dijkstra determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos positivos.

Trabaja devolviendo en realidad el peso mínimo de las aristas, no el camino mínimo propiamente dicho ya que trabaja con el objetivo de llegar a lo más optimo, donde el óptimo encontrado puede llegar a modificarse si surge una solución mejor.

1.1 Como trabaja

Primero marcamos todos los vértices como no utilizados. El algoritmo parte de un vértice origen que será ingresado, a partir de ese vértice 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. 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 (relaxation).

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.

2. *Algoritmo en pseudocódigo*

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

```
1  método Dijkstra(Grafo,origen):
2      creamos una cola de prioridad Q
3      agregamos origen a la cola de prioridad Q
4      mientras Q no este vacío:
5          sacamos un elemento de la cola Q llamado u
6          si u ya fue visitado continuo sacando elementos de Q
7          marcamos como visitado u
8          para cada vértice v adyacente a u en el Grafo:
9              sea w el peso entre vértices ( u , v )
10             si v no ah sido visitado:
11                Relajacion( u , v , w )

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

3. *Complejidad*

Orden de complejidad del algoritmo: O(|V|2+|A|) = O(|V|2) sin utilizar cola de prioridad, O((|A|+|V|) log |V|) = O(|A| log |V|) utilizando cola de prioridad (por ejemplo un montículo). Por otro lado, si se utiliza un Montículo de Fibonacci, sería O(|V| log |V|+|A|).

La complejidad computacional del algoritmo de Dijkstra se puede calcular contando las operaciones realizadas:

- El algoritmo consiste en n-1 iteraciones como máximo. En cada iteración se añade un vértice al conjunto distinguido.
- En cada iteración se identifica el vértice con la menor etiqueta entre los que no están en Sk. El número de estas operaciones está acotado por n-1.
- Además se realizan una suma y una comparación para actualizar la etiqueta de cada uno de los vértices que no están en Sk.

Luego, en cada iteración se realizan a lo sumo 2(n-1) operaciones. Entonces tenemos: TEOREMA: El Algoritmo de Dijkstra realiza O(n2) operaciones (sumas y comparaciones) para determinar la longitud del camino más corto entre dos vértices de un grafo ponderado simple, conexo y no dirigido con n vértice.

4. *Implementación*

Este algoritmo de Dijkstra se realiza como un método de la clase de Grafos ya que estos nos permiten el manejo del mismo. A continuación, se muestra la clase de grafo que utilizamos en la act. para las 5 pruebas de los diferentes grafos y después el método de Dijkstra ya implementado donde se muestran los 5 grafos ya realizados.

```
8  #Grafo HJAP
9  class Grafo:
10
11      def __init__(self):
12          self.V = set() # un conjunto
13          self.E = dict() # un mapeo de pesos de aristas
14          self.vecinos = dict() # un mapeo
15
16      def agrega(self, v):
17          self.V.add(v)
18          if not v in self.vecinos: # vecindad de v
19              self.vecinos[v] = set() # inicialmente no tiene nada
20
21      def conecta(self, v, u, peso=1):
22          self.agrega(v)
23          self.agrega(u)
24          self.E[(v, u)] = self.E[(u, v)] = peso # en ambos sentidos
25          self.vecinos[v].add(u)
26          self.vecinos[u].add(v)
27
28      @property
29      def Complemento(self):
30          comp=Grafo()
31          for v in self.V:
32              for w in self.V:
33                  if v!=w and (v,w) not in self.E:
34                      comp.Conecta(v,w,1)
35
```

- ***El método de Dijkstra***

```
58      #Algoritmo Dijkstra
59      def shortest1(self,v):
60          q= [(0, v, ())]
61          dist = dict()
62          visited = set()
63          while len(q) > 0:
64              (l, u, p) = heappop(q)
65              if u not in visited:
66                  visited.add(u)
67                  dist[u]= (1,u,list(flatten(p))[::-1]+[u])
68              p= (u,p)
69              for n in self.vecinos[u]:
70                  if n not in visited:
71                      el = self.E[(u,n)]
72                      heappush(q, (l+el, n, p))
73          return dist
74
```

```python
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,
       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 ditana
       return dist #regresa el diccionario de distancias
    #Algoritmo Dijkstra
    def shortest1(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]= (1,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

```

5. *Prueba del algoritmo*

Se muestra la aplicación del algoritmo con las 5 pruebas de los grafos considerándose como el nodo inicial a ‘a’ donde cada una de ellas consta de su función donde se especifica la cantidad de nodos y aristas, se observa el resultado que se obtuvo al ejecutarlo junto con la cantidad de vecinos de ‘a’ y al final la tabla interpretando los resultados.

Grafo 1:

```
75 # Primero 5 nodos con 10 aristas
76 print("Primer grafo 5 nodos con 10 aristas")
77 g=Grafo()
78 g.conecta('a','b', 2)
79 g.conecta('a','c', 5)
80 g.conecta('a','d', 13)
81 g.conecta('a','e', 6)
82 g.conecta('b','c', 23)
83 g.conecta('b','d', 9)
84 g.conecta('b','e', 6)
85 g.conecta('c','d', 4)
86 g.conecta('d','e', 1)
87 g.conecta('e','c', 8)
88 print(g.vecinos['a'])
89 print(g.shortest('a'))
```

RESULTADO:

Primer grafo 5 nodos con 10 aristas

{'b', 'e', 'd', 'c'}

{'a': (0, 'a', ['a']), 'b': (2, 'b', ['a', 'b']), 'c': (5, 'c', ['a', 'c']), 'e': (6, 'e', ['a', 'e']), 'd': (7, 'd', ['a', 'e', 'd'])}

TABLA:

NODO	CAMINO	DISTANCIA
a	'a- ['a']	**0**
b	['a- 'b']	**2**
c	['a- 'c']	**5**
e	['a'-'e']	**6**
d	['a'- 'e'-'d']	**7**

Grafo 2:

```
#segundo 10 nodos con 20 aristas
print("Segundo grafo 10 nodos con 20 aristas")
g1=Grafo()
g1.conecta('a','b', 13)
g1.conecta('a','c',5)
g1.conecta('a','d',25)
g1.conecta('a','e',23)
g1.conecta('a','f',9)
g1.conecta('a','h',18)
g1.conecta('a','i',14)
g1.conecta('a','j',2)
g1.conecta('b','c',11)
g1.conecta('b','d',29)
g1.conecta('c','d',15)
g1.conecta('c','e',7)
g1.conecta('d','e',8)
g1.conecta('e','f',16)
g1.conecta('f','g',24)
g1.conecta('g','h',1)
g1.conecta('h','e',17)
g1.conecta('i','j',12)
g1.conecta('j','h',22)
g1.conecta('j','g',30)

print(g1.vecinos['a'])
print(g1.shortest('a'))

```

RESULTADO:

Segundo grafo 10 nodos con 20 aristas

{'h', 'd', 'e', 'f', 'b', 'c', 'i', 'j'}

{'a': (0, 'a', ['a']), 'j': (2, 'j', ['a', 'j']),

'c': (5, 'c', ['a', 'c']), 'f': (9, 'f', ['a', 'f']),

'e': (12, 'e', ['a', 'c', 'e']), 'b': (13, 'b', ['a', 'b']),

'i': (14, 'i', ['a', 'i']), 'h': (18, 'h', ['a', 'h']),

'g': (19, 'g', ['a', 'h', 'g']), 'd': (20, 'd', ['a', 'c', 'd'])}

TABLA:

Nodo	Camino	Distancia
a	**'a', ['a']**	**0**
j	['a'-'j']	**2**
c	['a'-'c']	**5**
f	['a'- 'f']	**9**
e	['a'-'c'- 'e']	**12**
b	['a'- 'b']	**13**
i	['a'- 'i']	**14**
h	['a'- 'h']	**18**
g	['a'-'h'- 'g']	**19**
d	['a'-'c'- 'd']	**20**

Grafo 3:

```
#tercero 15 nodos con 30 aristas
print("Tercero grafo 15 nodos con 30 aristas")
g3=Grafo()
g3.conecta('a','b',2)
g3.conecta('a','c',9)
g3.conecta('a','d',4)
g3.conecta('a','e',3)
g3.conecta('a','f',7)
g3.conecta('a','j',1)
g3.conecta('a','h',11)
g3.conecta('a','i',15)
g3.conecta('a','j',18)
g3.conecta('a','k',19)
g3.conecta('a','l',6)
g3.conecta('a','m',8)
g3.conecta('a','n',9)
g3.conecta('a','o',5)
g3.conecta('a','p',16)
g3.conecta('b','c',1)
g3.conecta('b','d',12)
g3.conecta('b','e',14)
g3.conecta('b','f',21)
g3.conecta('b','h',11)
g3.conecta('b','i',4)
g3.conecta('b','j',15)
g3.conecta('b','k',10)
g3.conecta('b','l',3)
g3.conecta('b','m',20)
g3.conecta('b','n',10)
g3.conecta('b','o',7)
g3.conecta('b','p',5)
g3.conecta('c','d',6)
g3.conecta('c','e',13)
print(g3.vecinos['a'])
print(g3.shortest('a'))
```

RESULTADO:

Tercero grafo 15 nodos con 30 aristas

{'o', 'h', 'd', 'l', 'k', 'e', 'f', 'p', 'b', 'c', 'm', 'i', 'n', 'j'}

{'a': (0, 'a', ['a']), 'b': (2, 'b', ['a', 'b']), 'c': (3, 'c', ['a', 'b', 'c']), 'e': (3, 'e', ['a', 'e']), 'd': (4, 'd', ['a', 'd']), 'l': (5, 'l', ['a', 'b', 'l']), 'o': (5, 'o', ['a', 'o']), 'i': (6, 'i', ['a', 'b', 'i']), 'f': (7, 'f', ['a', 'f']), 'p': (7, 'p', ['a', 'b', 'p']), 'm': (8, 'm', ['a', 'm']), 'n': (9, 'n', ['a', 'n']), 'h': (11, 'h', ['a', 'h']), 'k': (12, 'k', ['a', 'b', 'k']), 'j': (17, 'j', ['a', 'b', 'j'])}

TABLA:

Nodo	Camino	Distancia
a	**'a- ['a']**	**0**
b	['a- 'b']	**2**
c	['a- 'b- 'c']	**3**
e	['a-'e']	**3**
d	['a-'d']	**4**
l	['a-'b- 'l']	**5**
o	['a- 'o']	**5**
i	['a-b- 'i']	**6**
f	['a- 'f']	**7**
p	['a- 'b- 'p']	**7**
m	['a- 'm']	**8**
n	['a- 'n']	**9**
h	['a-'h']	**11**
k	['a- 'b- 'k']	**12**
j	['a- 'b- 'j']	**17**

Grafo 4:

```
#Cuarto 20 nodos con 40 aristas
print("Cuarto grafo 20 nodos con 40 aristas")
g4=Grafo()
g4.conecta('a','b',1)
g4.conecta('a','c',5)
g4.conecta('a','d',23)
g4.conecta('a','e',25)
g4.conecta('a','f',6)
g4.conecta('a','g',8)
g4.conecta('a','h',10)
g4.conecta('a','i',11)
g4.conecta('a','j',12)
g4.conecta('a','k',5)
g4.conecta('b','c',6)
g4.conecta('b','d',27)
g4.conecta('b','e',3)
g4.conecta('b','f',4)
g4.conecta('b','g',14)
g4.conecta('b','h',16)
g4.conecta('b','i',17)
g4.conecta('b','j',7)
g4.conecta('b','k',9)
g4.conecta('b','l',14)
g4.conecta('c','d',19)
g4.conecta('c','e',22)
g4.conecta('c','f',24)
g4.conecta('c','g',28)
g4.conecta('c','h',2)
g4.conecta('c','i',1)
g4.conecta('c','j',10)
g4.conecta('c','k',13)
g4.conecta('c','l',33)
g4.conecta('c','m',37)
g4.conecta('c','n',40)
g4.conecta('c','o',26)
g4.conecta('c','p',29)
g4.conecta('c','q',12)
g4.conecta('c','r',38)
g4.conecta('c','s',13)
g4.conecta('c','t',35)
g4.conecta('d','l',34)
g4.conecta('d','m',21)
g4.conecta('d','n',28)
print(g4.vecinos['a'])
print(g4.shortest('a'))
```

RESULTADO:

Cuarto grafo 20 nodos con 40 aristas

{'h', 'd', 'k', 'e', 'f', 'g', 'b', 'c', 'i', 'j'}

{'a': (0, 'a', ['a']), 'b': (1, 'b', ['a', 'b']), 'e': (4, 'e', ['a', 'b', 'e']), 'c': (5, 'c', ['a', 'c']), 'f': (5, 'f', ['a', 'b', 'f']), 'k': (5, 'k', ['a', 'k']), 'i': (6, 'i', ['a', 'c', 'i']), 'h': (7, 'h', ['a', 'c', 'h']), 'g': (8, 'g', ['a', 'g']), 'j': (8, 'j', ['a', 'b', 'j']), 'l': (15, 'l', ['a', 'b', 'l']), 'q': (17, 'q', ['a', 'c', 'q']), 's': (18, 's', ['a', 'c', 's']), 'd': (23, 'd', ['a', 'd']), 'o': (31, 'o', ['a', 'c', 'o']), 'p': (34, 'p', ['a', 'c', 'p']), 't': (40, 't', ['a', 'c', 't']), 'm': (42, 'm', ['a', 'c', 'm']), 'r': (43, 'r', ['a', 'c', 'r']), 'n': (45, 'n', ['a', 'c', 'n'])}

TABLA:

Nodo	Camino	Distancia
a	'a'- ['a']	0
b	['a'- 'b']	1
e	['a'-'b'- 'e']	4
c	['a'- 'c']	5
f	['a'- 'b'- 'f']	5
k	['a'-'k']	5
i	['a'- 'c'- 'i']	6
h	['a'-'c'-'h']	7
g	['a'- 'g']	8
j	['a'- 'b'- 'j']	8
l	['a'- 'b'- 'l']	15
q	['a'-'c'-'q']	17
s	['a'-'c'- 's']	18
d	['a'- 'd']	23
o	['a'- 'c'- 'o']	31
p	['a'-'c'-'p']	34
t	['a'- 'c'- 't']	40
m	['a'- 'c'-'m']	42
r	['a'- 'c'- 'r']	43
n	['a'- 'c'- 'n']	45

Grafo 5:

```
#Quinto 25 nodos con 50 aristas
print("Quinto grafo 25 nodos con 50 aristas")
g5=Grafo()
g5.conecta('a','b',10)
g5.conecta('a','c',1)
g5.conecta('a','d',9)
g5.conecta('a','e',2)
g5.conecta('a','f',8)
g5.conecta('a','g',3)
g5.conecta('a','h',7)
g5.conecta('a','i',4)
g5.conecta('a','j',6)
g5.conecta('a','k',5)
g5.conecta('a','l',20)
g5.conecta('a','m',11)
g5.conecta('a','n',19)
g5.conecta('a','o',12)
g5.conecta('a','p',18)
g5.conecta('a','q',30)
g5.conecta('a','r',21)
g5.conecta('a','s',29)
g5.conecta('a','t',22)
g5.conecta('a','u',28)
g5.conecta('a','v',13)
g5.conecta('a','w',17)
g5.conecta('a','x',15)
g5.conecta('a','y',16)
g5.conecta('a','z',36)
g5.conecta('b','c',25)
g5.conecta('b','d',26)
g5.conecta('b','e',36)
g5.conecta('b','f',34)
g5.conecta('b','g',32)
g5.conecta('b','h',31)
g5.conecta('b','i',41)
g5.conecta('b','j',47)
g5.conecta('b','k',44)
g5.conecta('b','l',15)
g5.conecta('b','m',12)
g5.conecta('b','n',10)
g5.conecta('b','o',43)
g5.conecta('b','p',46)
g5.conecta('b','q',33)
g5.conecta('b','r',30)
g5.conecta('b','s',24)
g5.conecta('b','t',29)
g5.conecta('c','d',27)
g5.conecta('c','e',13)
g5.conecta('c','f',11)
g5.conecta('c','g',1)
g5.conecta('c','h',6)
g5.conecta('c','i',7)
g5.conecta('c','j',6)
print(g5.vecinos['a'])
print(g5.shortest('a'))

```

RESULTADO:

Quinto grafo 25 nodos con 50 aristas

{'o', 'e', 'n', 'j', 'z', 'v', 'h', 'r', 'd', 'l', 'f', 'y', 'g', 'c', 'm', 'i', 'q', 'w', 'x', 'k', 's', 'b', 't', 'u', 'p'}

{'a': (0, 'a', ['a']), 'c': (1, 'c', ['a', 'c']), 'e': (2, 'e', ['a', 'e']), 'g': (2, 'g', ['a', 'c', 'g']), 'i': (4, 'i', ['a', 'i']), 'k': (5, 'k', ['a', 'k']), 'j': (6, 'j', ['a', 'j']), 'h': (7, 'h', ['a', 'h']), 'f': (8, 'f', ['a', 'f']), 'd': (9, 'd', ['a', 'd']), 'b': (10, 'b', ['a', 'b']), 'm': (11, 'm', ['a', 'm']), 'o': (12, 'o', ['a', 'o']), 'v': (13, 'v', ['a', 'v']), 'x': (15, 'x', ['a', 'x']), 'y': (16, 'y', ['a', 'y']), 'w': (17, 'w', ['a', 'w']), 'p': (18, 'p', ['a', 'p']), 'n': (19, 'n', ['a', 'n']), 'l': (20, 'l', ['a', 'l']), 'r': (21, 'r', ['a', 'r']), 't': (22, 't', ['a', 't']), 'u': (28, 'u', ['a', 'u']), 's': (29, 's', ['a', 's']), 'q': (30, 'q', ['a', 'q']), 'z': (36, 'z', ['a', 'z'])}

TABLA:

Nodo	Camino	Distancia
a	'a- ['a']	0
c	['a- 'c']	1
e	['a- 'e']	2
g	['a'-'c- 'g']	2
i	['a- 'i']	4
k	['a'-'k']	5
j	['a- 'j']	6
h	['a'-'h']	7
f	['a- 'f']	8
d	['a'-'d']	9
b	['a'-'b']	10
m	['a'-'m']	11
o	['a- 'o']	12
v	['a- 'v']	13
x	['a- 'x']	15
y	['a- 'y']	16

w	['a'-'w']	17
p	['a'- 'p']	18
n	['a'- 'n']	19
l	['a'- 'l']	20
r	['a'-'r']	21
t	['a'-'t']	22
u	['a'-'u']	28
s	['a'-'s']	29
q	['a'-'q']	30

CONCLUSION:

Lo más difícil de este algoritmo a mi consideración fue tener bien estructurado que se realizara y en qué orden en lo cual tuve problemas y se mandó fuera de tiempo. Además de tener las estructuras adecuadas, tal como el Heap que me pareció curioso cómo es posible implementarlo en Python.

La principal ventaja o utilidad para nosotros como matemáticos es que es muy bueno aplicarlo en las matemáticas ya que en este caso ya contabas con algo de conocimientos sobre los grafos además que con la ayuda adecuada del maestro es sencillo utilizar este algoritmo principalmente para algún problema sobre alguna entrega o distribución de productos dentro de una red de lugares con el fin de encontrar el camino más corto como se mencionó en un principio.

Como matemáticos esto nos hace despertar el interés por implementar estos algoritmos en nuestra área en problemas para facilitar nuestro trabajo en lo personal me beneficio este algoritmo de Dijkstra ya que llevo las materias de modelado matemático y matemáticas discretas en las cuales veo este modelo de la ruta más corta y gracias a este reporte esta actividad poder utilizar Dijkstra para algunas actividades que llevare acabo durante el semestre en curso a pesar que se me dificulto mucho al recurrir al apoyo del maestro se pudo concluir esta actividad de una manera satisfactoria ya que se cumplió el objetivo, los resultados realmente comprendí el algoritmo y en que consiste para que lo aplico lo principal para llegar a lo esperado es analizar la clave es analizar con lo que trabajamos.