Algoritmo Dijkstra

Reporte de la clase Matemáticas Computacionales

Paulina Aguirre García

21.10.17

Introducción.

Anteriormente hablamos de lo que es un grafo y ciertas maneras de como recorrerlo (por “Anchura” y por “Profundidad”), por fines de la ejemplificación que dimos no consideramos algunos “componentes” de un grafo, específicamente a lo que llaman peso. Es decir, que las aristas de nuestro grafo tuvieran un valor, es a lo que llaman *grafo ponderado.*

¿Para que sirve este “valor” y por qué lo consideramos ahora?

Imaginemos que tenemos dos caminos que nos llevan a casa, uno de 10 km y otro de 3 km, lo más optimo es ir por el camino más corto (ya sea para caminar menos o gastar menos en combustible), entonces es como si ahora nuestro grafo fuera un camino que tenemos que recorrer, pero sus aristas difieren en distancias.

El algoritmo que ejemplificaremos recorre también un grafo, pero ahora es un grafo ponderado.

Dijkstra.

Este algoritmo descubierto por el físico neerlandés Edsger Dijkstra en 1959, es uno de los diferentes algoritmos para hallar un **camino de longitud mínima** entre dos vértices de un grafo ponderado.

Es decir, como mencione antes tendré diversos caminos (las aristas hacia los vecinos del vértice raíz que tomemos) que difieren en distancia, entonces este algoritmo opta por el camino más corto. Por el que “cueste” menos.

Implementación en Python (Código).

Bueno como vamos a recorrer un grafo, necesitamos un grafo. Utilizaremos la estructura que hicimos la última vez.

```
 7 from heapq import heappop, heappush
 8
 9 def flatten(L):
10     while len(L)>0:
11         yield L[0]
12         L=L[1]
13
14 #Grafo
15 class grafo:
16     def __init__(self):
17         self.vertices=set()
18         self.E=dict()
19         self.vecinos=dict()
20
21     def A(self, v): #Agregar
22         self.vertices.add(v)
23         if not v in self.vecinos: #Vecindá de v
24             self.vecinos[v]=set()
25
26     def C(self, v, u, peso=1): #Concectar
27         self.A(v)
28         self.A(u)
29         self.E[(v,u)]=self.E[(u,v)]=peso
30         self.vecinos[v].add(u)
31         self.vecinos[u].add(v)
32
33     def __str__(self):
34         return "Aristas= " + str(self.E)+"\nVertices = " +str(self.vertices)
35
```

Agregamos la función flatten para usarla en el algoritmo.

```
36 #Algoritmo Dijkstra
37     def shortest(self, v): # Dijkstra's algorithm
38         q = [(0, v, ())] # arreglo "q" de las "Tuplas" de lo que se va a almac
39         dist = dict() #diccionario de distancias
40         visited = set() #Conjunto de visitados
41         while len(q) > 0: #mientras exista un nodo pendiente
42             (l, u, p) = heappop(q) # Se toma la tupla con la distancia menor
43             if u not in visited: # si no lo hemos visitado
44                 visited.add(u) #se agrega a visitados
45                 dist[u] = (l,u,list(flatten(p))[::-1] + [u])    #agrega al dic
46             p = (u, p) #Tupla del nodo y el camino
47             for n in self.vecinos[u]: #Para cada hijo del nodo actual
48                 if n not in visited: #si no lo hemos visitado
49                     el = self.E[(u,n)] #se toma la distancia del nodo acutal h
50                     heappush(q, (l + el, n, p)) #Se agrega al arreglo "q" la d
51         return dist #regresa el diccionario de distancias
52
```

**Una tupla es una secuencia de valores agrupados.*

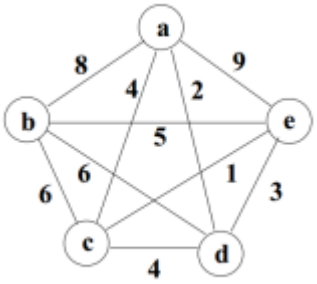
Lo que hace el algoritmo es empezar en un vértice asignado, donde la distancia a él mismo es 0, luego “ve” los hijos de este vértice y la distancia a la que están, dónde “heappop” toma la distancia menor y pasamos al vértice correspondiente el cuál pasa al conjunto de nodos visitados. Ahora, cabe a resaltar que estas distancias claramente deben ser positivas. Entonces estando en el nodo actual hacemos el mismo procedimiento, pero ahora comenzamos a considerar si alguno de sus hijos ya fue visitado o no. Con esto, nuestro grafo puede ser un *“grafo dirigido*.

Ejemplos.

1.- 5 nodos 10 vértices

```
###Grafillos para probar el Dijkstra###

g= grafo() #Grafillo con 5 nodos y 10 vertices
g.C('a','b', 8)
g.C('b','c', 6)
g.C('a','d', 2)
g.C('a','e', 9)
g.C('c','e', 1)
g.C('c','d', 4)
g.C('a','c', 4)
g.C('b','d', 6)
g.C('d','e', 3)
g.C('b','e', 5)

print(g.shortest('a'))

```

Gráficamente



Grafo Ponderado No Dirigido

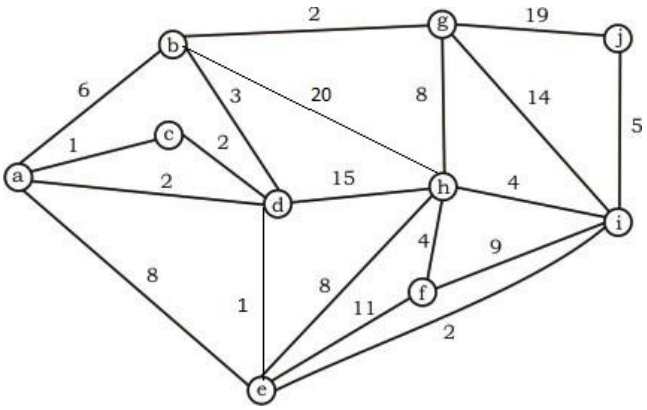
```
In [85]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/
Desktop')
{'a': (0, 'a', ['a']), 'd': (2, 'd', ['a', 'd']), 'c': (4, 'c', ['a', 'c']),
'e': (5, 'e', ['a', 'c', 'e']), 'b': (8, 'b', ['a', 'b'])}

In [86]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/
Desktop')
{'a': (0, 'a', ['a']), 'd': (2, 'd', ['a', 'd']), 'c': (4, 'c', ['a', 'c']),
'e': (5, 'e', ['a', 'c', 'e']), 'b': (8, 'b', ['a', 'b'])}
```

El algoritmo nos regresa que nodo está más cerca de ‘a’ hasta el que está más lejano.

2.- 10 nodos 20 vértices

```
69 g2=grafo() #Grafo con 10 nodos y 20 vertices
70 g2.C('a','b', 6)
71 g2.C('a','c', 1)
72 g2.C('a','d', 2)
73 g2.C('a','e', 8)
74 g2.C('b','d', 3)
75 g2.C('b','g', 2)
76 g2.C('c','d', 2)
77 g2.C('d','h',15)
78 g2.C('e','h', 8)
79 g2.C('e','f',11)
80 g2.C('e','i', 2)
81 g2.C('g','h', 8)
82 g2.C('g','j',19)
83 g2.C('g','i',14)
84 g2.C('h','i', 4)
85 g2.C('h','f', 4)
86 g2.C('j','i', 5)
87 g2.C('f','i', 9)
88 g2.C('d','e', 1)
89 g2.C('b','h',20)
90
91 print(g2.shortest('a'))
92
```

Gráficamente.




```
In [87]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/Desktop')
{'a': (0, 'a', ['a']), 'c': (1, 'c', ['a', 'c']), 'd': (2, 'd', ['a', 'd']), 'e': (3, 'e',
['a', 'd', 'e']), 'b': (5, 'b', ['a', 'd', 'b']), 'i': (5, 'i', ['a', 'd', 'e', 'i']),
'g': (7, 'g', ['a', 'd', 'b', 'g']), 'h': (9, 'h', ['a', 'd', 'e', 'i', 'h']), 'j': (10,
'j', ['a', 'd', 'e', 'i', 'j']), 'f': (13, 'f', ['a', 'd', 'e', 'i', 'h', 'f'])}

In [88]:
```

3.- 15 nodos 30 vértices

```
g3=grafo() #Grafo de 15 nodos y 30 vertices
g3.C('1','4',13)
g3.C('1','7', 2)
g3.C('2','1', 1)
g3.C('3','2', 2)
g3.C('3','1',25)
g3.C('3','5',30)
g3.C('5','2', 5)
g3.C('5','8',14)
g3.C('5','6', 4)
g3.C('6','3',11)
g3.C('6','9', 9)
g3.C('7','4',12)
g3.C('7','6',17)
g3.C('7','10',8)
g3.C('8','9', 3)
g3.C('8','11',6)
g3.C('9','5',15)
g3.C('10','9',8)
g3.C('11','10',7)
g3.C('11','12',1)
g3.C('8','14',4)
g3.C('14','11',2)
g3.C('9','13',8)
g3.C('13','11',3)
g3.C('7','15',2)
g3.C('15','6',1)
g3.C('10','15',4)
g3.C('15','8',3)
g3.C('10''13',9)
g3.C('8','13',0)
```

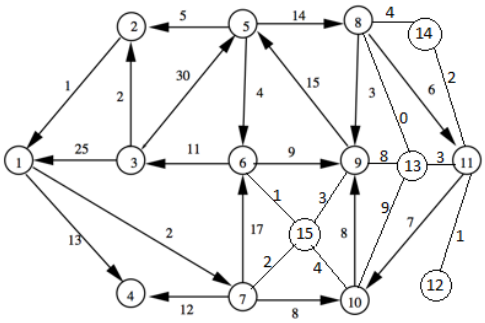


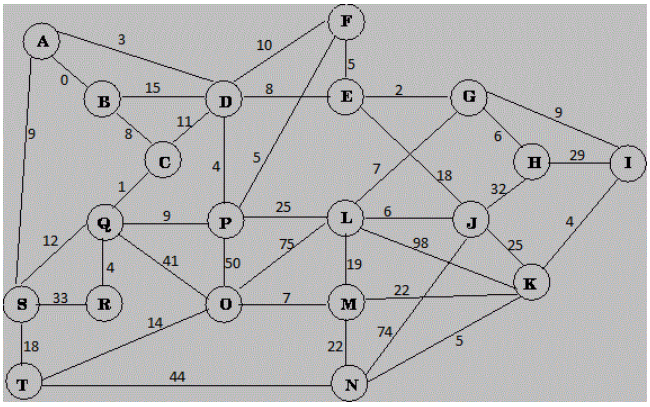
Figure 4

Gráficamente.

```
In [92]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/Desktop')
{'1': (0, '1', ['1']), '2': (1, '2', ['1', '2']), '7': (2, '7', ['1', '7']), '3': (3, '3', ['1', '2',
'3']), '15': (4, '15', ['1', '7', '15']), '6': (5, '6', ['1', '7', '15', '6']), '5': (6, '5', ['1',
'2', '5']), '8': (7, '8', ['1', '7', '15', '8']), '13': (7, '13', ['1', '7', '15', '8', '13']), '10':
(8, '10', ['1', '7', '15', '10']), '11': (10, '11', ['1', '7', '15', '8', '13', '11']), '9': (10,
'9', ['1', '7', '15', '8', '9']), '12': (11, '12', ['1', '7', '15', '8', '13', '11', '12']), '14':
(11, '14', ['1', '7', '15', '8', '14']), '4': (13, '4', ['1', '4'])}

In [93]:
```

4.- 20 nodos 40 vértices.

```
g4=grafo() #Grafillo de 20 nodos y 40 vertices
g4.C('A','B', 0)
g4.C('A','S', 9)
g4.C('A','D', 3)
g4.C('D','F',10)
g4.C('F','E', 5)
g4.C('B','D',15)
g4.C('D','E', 8)
g4.C('B','C', 8)
g4.C('E','G', 2)
g4.C('G','I', 9)
g4.C('G','H', 6)
g4.C('H','I',29)
g4.C('C','D',11)
g4.C('D','P', 4)
g4.C('C','Q', 1)
g4.C('Q','P', 9)
g4.C('F','P', 5)
g4.C('P','O',50)
g4.C('Q','S',12)
g4.C('S','R',33)
g4.C('Q','R', 4)
g4.C('S','T',18)
g4.C('T','O',14)
g4.C('Q','O',41)
g4.C('O','M', 7)
g4.C('O','L',75)
g4.C('T','N',44)
g4.C('N','M',22)
g4.C('M','L',19)
g4.C('L','G', 7)
g4.C('E','J',18)
g4.C('J','K',25)
g4.C('L','J', 6)
g4.C('J','H',32)
g4.C('K','I', 4)
g4.C('L','K',98)
g4.C('M','K',22)
g4.C('N','J',74)
g4.C('N','K', 5)
g4.C('P','L',25)
```



Gráficamente.

```
In [89]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/Desktop')
{'A': (0, 'A', ['A']), 'B': (0, 'B', ['A', 'B']), 'D': (3, 'D', ['A', 'D']), 'P': (7, 'P', ['A', 'D', 'P']),
'C': (8, 'C', ['A', 'B', 'C']), 'Q': (9, 'Q', ['A', 'B', 'C', 'Q']), 'S': (9, 'S', ['A', 'S']), 'E': (11,
'E', ['A', 'D', 'E']), 'F': (12, 'F', ['A', 'D', 'P', 'F']), 'G': (13, 'G', ['A', 'D', 'E', 'G']), 'R': (13,
'R', ['A', 'B', 'C', 'Q', 'R']), 'H': (19, 'H', ['A', 'D', 'E', 'G', 'H']), 'L': (20, 'L', ['A', 'D', 'E',
'G', 'L']), 'I': (22, 'I', ['A', 'D', 'E', 'G', 'I']), 'J': (26, 'J', ['A', 'D', 'E', 'G', 'L', 'J']), 'K':
(26, 'K', ['A', 'D', 'E', 'G', 'I', 'K']), 'T': (27, 'T', ['A', 'S', 'T']), 'N': (31, 'N', ['A', 'D', 'E',
'G', 'I', 'K', 'N']), 'M': (39, 'M', ['A', 'D', 'E', 'G', 'L', 'M']), 'O': (41, 'O', ['A', 'S', 'T', 'O'])}

In [90]:
```

Bien, nuestro nodo ahora es bastante grande

Aquí se puede observar que al recorrer varios nodos suma las distancias, como:

(7, 'P', ['A', 'D', 'P']) que dice que la distancia de A a D a P es igual a 7 ó

(41, 'O', ['A', 'S', 'T', 'O']) es decir, la distancia de A a S a T a O es igual a 41.

Lo cual podemos comprobar fácilmente en la gráfica.

Aquí los 4 ejemplos juntos.

```
In [93]: runfile('C:/Users/Aguir/Desktop/Dijkstra.py', wdir='C:/Users/Aguir/Desktop')
{'a': (0, 'a', ['a']), 'd': (2, 'd', ['a', 'd']), 'c': (4, 'c', ['a', 'c']), 'e': (5, 'e', ['a', 'c', 'e']),
'b': (8, 'b', ['a', 'b'])}
{'a': (0, 'a', ['a']), 'c': (1, 'c', ['a', 'c']), 'd': (2, 'd', ['a', 'd']), 'e': (3, 'e', ['a', 'd', 'e']),
'b': (5, 'b', ['a', 'd', 'b']), 'i': (5, 'i', ['a', 'd', 'e', 'i']), 'g': (7, 'g', ['a', 'd', 'b', 'g']),
'h': (9, 'h', ['a', 'd', 'e', 'i', 'h']), 'j': (10, 'j', ['a', 'd', 'e', 'i', 'j']), 'f': (13, 'f', ['a',
'd', 'e', 'i', 'h', 'f'])}
{'1': (0, '1', ['1']), '2': (1, '2', ['1', '2']), '7': (2, '7', ['1', '7']), '3': (3, '3', ['1', '2', '3']),
'15': (4, '15', ['1', '7', '15']), '6': (5, '6', ['1', '7', '15', '6']), '5': (6, '5', ['1', '2', '5']),
'8': (7, '8', ['1', '7', '15', '8']), '13': (7, '13', ['1', '7', '15', '8', '13']), '10': (8, '10', ['1',
'7', '15', '10']), '11': (10, '11', ['1', '7', '15', '8', '13', '11']), '9': (10, '9', ['1', '7', '15', '8',
'9']), '12': (11, '12', ['1', '7', '15', '8', '13', '11', '12']), '14': (11, '14', ['1', '7', '15', '8',
'14']), '4': (13, '4', ['1', '4'])}
{'A': (0, 'A', ['A']), 'B': (0, 'B', ['A', 'B']), 'D': (3, 'D', ['A', 'D']), 'P': (7, 'P', ['A', 'D', 'P']),
'C': (8, 'C', ['A', 'B', 'C']), 'Q': (9, 'Q', ['A', 'B', 'C', 'Q']), 'S': (9, 'S', ['A', 'S']), 'E': (11,
'E', ['A', 'D', 'E']), 'F': (12, 'F', ['A', 'D', 'P', 'F']), 'G': (13, 'G', ['A', 'D', 'E', 'G']), 'R': (13,
'R', ['A', 'B', 'C', 'Q', 'R']), 'H': (19, 'H', ['A', 'D', 'E', 'G', 'H']), 'L': (20, 'L', ['A', 'D', 'E',
'G', 'L']), 'I': (22, 'I', ['A', 'D', 'E', 'G', 'I']), 'J': (26, 'J', ['A', 'D', 'E', 'G', 'L', 'J']), 'K':
(26, 'K', ['A', 'D', 'E', 'G', 'I', 'K']), 'T': (27, 'T', ['A', 'S', 'T']), 'N': (31, 'N', ['A', 'D', 'E',
'G', 'I', 'K', 'N']), 'M': (39, 'M', ['A', 'D', 'E', 'G', 'L', 'M']), 'O': (41, 'O', ['A', 'S', 'T', 'O'])}

In [94]:
```

Conclusiones.

La función de este algoritmo me recuerda mucho al “problema del viajero”, por lo cual busque cuales han sido sus aplicaciones y efectivamente ha sido usado en eso.

Es interesante ver que entre más grande es el grafo comienza a hacer más operaciones de cuál es un buen camino, como en el ejemplo 3 y 4.

Falto de agregar un ejemplo por lo extenso y por no tener la gráfica, pero está incorporado al código que estará en GitHub.