Karen Eunice Oviedo Garza Matricula 1753764

Algoritmo Dijkstra

Recordando en un reporte anterior un grafo está compuesto de vértices que también son llamados nodos, la unión entre un vértice y otro se forma la arista, si existe el vértice 'a' que está conectado a 'b', se forma la arista AB.

```
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
```

Ahora el ***algoritmo 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.

El algoritmo parte de un vèrtice origen serà ingresado,a partir de ese vertice evaluaremos sus adyacentes,como dijkstra usa una tecnica greedy.La tècnica greedy utiliza el principio de que para que un camino sea optimo,todos los caminos que contiene tambien deben ser optimos.

Al proceso de actualizar las distancias tomando como punto intermedio al nuevo vertice se le conoce como *relajacion(relaxation)*.

Dijkstra es muy similiar 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 prioridad de Min-Heap me debe devolver el de menor valor,en nuestro caso dicho valor serà el peso acumulado en los nodos.

```
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
```

Karen Eunice Oviedo Garza Matricula 1753764

Vamos a realizar 5 grafos.

El primero tendrá 5 nodos y 10 aristas.

Grafo 1				
Nodos: a,b,c,d,e		Resultados:Vertice de origen ‘a’		
Arista	Peso	n	Nodo anterior	Min. Distancia
1.ab	2	a	-	0
2.ac	1	c	a	1
3.cd	3	b	a	2
4.ad	2	d	a	2
5.de	5	e	c	2
6.be	6			
7.cb	2			
8.ea	4			
9.bd	8			
10.ce	1			

```
>>> g=Grafo()
>>> g.conecta('a','b',2)
>>> g.conecta('a','c',1)
>>> g.conecta('c','d',3)
>>> g.conecta('a','d',2)
>>> g.conecta('d','e',5)
>>> g.conecta('b','e',6)
>>> g.conecta('c','b',2)
>>> g.conecta('b','d',8)
>>> g.conecta('e','a',4)
>>> g.conecta('c','e',1)
>>> print(g.shortest('a'))
{'a': (0, 'a', ['a']), 'c': (1, 'c', ['a', 'c']), 'b': (2, 'b', ['a', 'b']), 'd'
: (2, 'd', ['a', 'd']), 'e': (2, 'e', ['a', 'c', 'e'])}
```

El segundo tendrá 10 nodos y 20 aristas.

Grafo 2			
Nodos: a, b ,c, d, e, f, g, h, i, j.		Resultados: Vértice de origen `b	
Arista	Peso	n	Min. Distancia
ab	1	b	0
ac	2	a	1
ad	3	c	3
bc	4	d	4
be	5	e	5
bh	6	h	6
cg	7	g	10
ce	8	j	11

ci	9	i	12
ja	10	f	22
jf	11		
jd	12		
ie	13		
ib	14		
ig	15		
he	16		
ha	17		
hj	18		
ga	19		
gd	20		

```
>>> g=Grafo()
>>> g.conecta('a','b',1)
>>> g.conecta('a','c',2)
>>> g.conecta('a','d',3)
>>> g.conecta('b','c',4)
>>> g.conecta('b','e',5)
>>> g.conecta('b','h',6)
>>> g.conecta('c','g',7)
>>> g.conecta('c','e',8)
>>> g.conecta('c','i',9)
>>> g.conecta('j','a',10)
>>> g.conecta('j','f',11)
>>> g.conecta('j','d',12)
>>> g.conecta('i','e',13)
>>> g.conecta('i','b',14)
>>> g.conecta('i','g',15)
>>> g.conecta('h','e',16)
>>> g.conecta('h','a',17)
>>> g.conecta('h','j',18)
>>> g.conecta('g','a',19)
>>> g.conecta('g','d',20)
>>> print(g.shortest('b'))
{'b': (0, 'b', ['b']), 'a': (1, 'a', ['b', 'a']), 'c': (3, 'c', ['b', 'a', 'c'])
, 'd': (4, 'd', ['b', 'a', 'd']), 'e': (5, 'e', ['b', 'e']), 'h': (6, 'h', ['b',
 'h']), 'g': (10, 'g', ['b', 'a', 'c', 'g']), 'j': (11, 'j', ['b', 'a', 'j']), '
i': (12, 'i', ['b', 'a', 'c', 'i']), 'f': (22, 'f', ['b', 'a', 'j', 'f'])}
>>>
```

El tercero tendrá 15 nodos y 30 aristas

Grafo 3	
Resultados: ***Vértice de origen 'c'***	
n	Min. Distancia
-	0
l	1
h	2
a	6
b	7

Karen Eunice Oviedo Garza Matricula 1753764

f	7
j	7
m	8
k	9
d	11
n	11
e	13
o	14
g	22
i	25

```
>>> g.conecta('a','b',3)
>>> g.conecta('a','j',1)
>>> g.conecta('b','h',5)
>>> g.conecta('b','o',10)
>>> g.conecta('c','f',7)
>>> g.conecta('c','k',9)
>>> g.conecta('d','a',5)
>>> g.conecta('d','e',2)
>>> g.conecta('e','i',15)
>>> g.conecta('e','n',17)
>>> g.conecta('f','b',8)
>>> g.conecta('f','m',20)
>>> g.conecta('g','l',22)
>>> g.conecta('g','f',15)
>>> g.conecta('h','a',4)
>>> g.conecta('h','n',9)
>>> g.conecta('i','o',13)
>>> g.conecta('i','e',12)
>>> g.conecta('j','k',5)
>>> g.conecta('j','l',7)
>>> g.conecta('k','d',8)
>>> g.conecta('k','o',25)
>>> g.conecta('l','c',1)
>>> g.conecta('l','h',1)
>>> g.conecta('m','d',8)
>>> g.conecta('m','j',1)
>>> g.conecta('n','j',9)
>>> g.conecta('n','c',16)
>>> g.conecta('o','j',7)
>>> g.conecta('o','n',5)
>>> print(g.shortest('c'))
{'c': (0, 'c', ['c']), 'l': (1, 'l', ['c', 'l']), 'h': (2, 'h', ['c', 'l', 'h'])
, 'a': (6, 'a', ['c', 'l', 'h', 'a']), 'b': (7, 'b', ['c', 'l', 'h', 'b']), 'f':
 (7, 'f', ['c', 'f']), 'j': (7, 'j', ['c', 'l', 'h', 'a', 'j']), 'm': (8, 'm', [
'c', 'l', 'h', 'a', 'j', 'm']), 'k': (9, 'k', ['c', 'k']), 'd': (11, 'd', ['c',
'l', 'h', 'a', 'd']), 'n': (11, 'n', ['c', 'l', 'h', 'n']), 'e': (13, 'e', ['c',
 'l', 'h', 'a', 'd', 'e']), 'o': (14, 'o', ['c', 'l', 'h', 'a', 'j', 'o']), 'g':
 (22, 'g', ['c', 'f', 'g']), 'i': (25, 'i', ['c', 'l', 'h', 'a', 'd', 'e', 'i'])
}
```

Karen Eunice Oviedo Garza Matricula 1753764

El cuarto tendrá 20 nodos y 40 aristas.

Grafo 4	
Resultados: Vértice de origen ‘d’	
n	Min. Distancia
d	0
t	1
o	2
n	4
g	5
s	7
f	8
j	8
k	9
m	9
r	9
b	10
h	10
p	10
e	11
i	11
q	11
a	13
l	13
c	14

```
>>> g=Grafo()
>>> g.conecta('a','l',3)
>>> g.conecta('a','b',5)
>>> g.conecta('b','i',3)
>>> g.conecta('b','p',8)
>>> g.conecta('c','e',3)
>>> g.conecta('c','b',7)
>>> g.conecta('d','o',2)
>>> g.conecta('d','t',1)
>>> g.conecta('e','q',4)
>>> g.conecta('e','j',9)
>>> g.conecta('f','q',3)
>>> g.conecta('f','s',1)
>>> g.conecta('g','r',7)
>>> g.conecta('g','d',5)
>>> g.conecta('h','a',5)
>>> g.conecta('h','s',3)
>>> g.conecta('i','o',9)
>>> g.conecta('i','l',12)
>>> g.conecta('j','t',7)
>>> g.conecta('j','r',1)
>>> g.conecta('k','e',2)
>>> g.conecta('k','l',17)
>>> g.conecta('l','p',3)
>>> g.conecta('l','b',6)
>>> g.conecta('m','b',2)
>>> g.conecta('m','t',8)
>>> g.conecta('n','t',3)
>>> g.conecta('n','h',7)
>>> g.conecta('o','b',8)
>>> g.conecta('o','t',5)
>>> g.conecta('p','c',9)
>>> g.conecta('p','k',1)
>>> g.conecta('q','b',16)
>>> g.conecta('q','s',9)
>>> g.conecta('r','c',7)
>>> g.conecta('r','f',3)
>>> g.conecta('s','i',9)
>>> g.conecta('s','o',5)
>>> g.conecta('t','a',12)
>>> g.conecta('t','k',8)
>>> print(g.shortest('d'))
```

```
{'d': (0, 'd', ['d']), 't': (1, 't', ['d', 't']), 'o': (2, 'o', ['d', 'o']), 'n'
: (4, 'n', ['d', 't', 'n']), 'g': (5, 'g', ['d', 'g']), 's': (7, 's', ['d', 'o',
 's']), 'f': (8, 'f', ['d', 'o', 's', 'f']), 'j': (8, 'j', ['d', 't', 'j']), 'k'
: (9, 'k', ['d', 't', 'k']), 'm': (9, 'm', ['d', 't', 'm']), 'r': (9, 'r', ['d',
 't', 'j', 'r']), 'b': (10, 'b', ['d', 'o', 'b']), 'h': (10, 'h', ['d', 'o', 's'
, 'h']), 'p': (10, 'p', ['d', 't', 'k', 'p']), 'e': (11, 'e', ['d', 't', 'k', 'e
']), 'i': (11, 'i', ['d', 'o', 'i']), 'q': (11, 'q', ['d', 'o', 's', 'f', 'q']),
 'a': (13, 'a', ['d', 't', 'a']), 'l': (13, 'l', ['d', 't', 'k', 'p', 'l']), 'c'
: (14, 'c', ['d', 't', 'k', 'e', 'c'])}
>>>
```