Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

PROYECTO FINAL

Brandon Rodrigo Ceballos Salazar

Matrícula: 1724045

Grafo

Un grafo en el ámbito de las ciencias de la computación es un tipo abstracto de datos, que consiste en un conjunto de nodos y un conjunto de aristas que establecen relaciones entre los nodos.

```
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
```

Algoritmo de Dijkstra

También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

```
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, v el nodo y () el "camino" hacia el
        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
            h = (u, p)  # Tupla del nodo y el camino
            for n in self.vecinos[u]:  # Para cada hijo del nodo actual
```

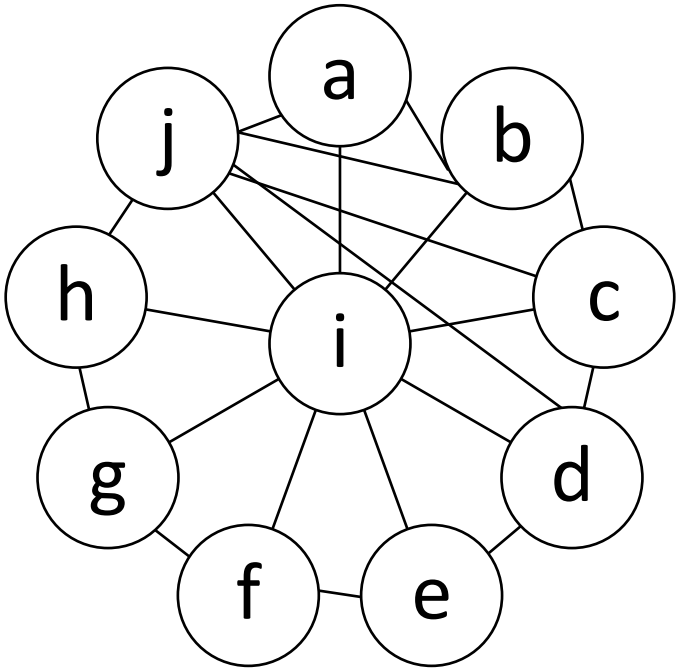
```
            if n not in visited:  # si no lo hemos visitado
                el = self.E[(u, n)]  # se toma la distancia del nodo acutal hacia el nodo hijo
                heappush(q, (l + el, n, p))  # Se agrega al arreglo "q" la distancia actual mas la ditanacia hacia el nodo hijo, el nodo hijo n hacia donde se va, y el camino
        return dist  # regresa el diccionario de distancias
```

Algoritmo de Kruskal

Es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa).

```
def kruskal(self):
        e = deepcopy(self.E)
        arbol = grafo()
        peso = 0
        comp = dict()
        t = sorted(e.keys(), key=lambda k: e[k], reverse=True)
        nuevo = set()
        while len(t) > 0 and len(nuevo) < len(self.V): # print(len(t))
            arista = t.pop()
            w = e[arista]
            del e[arista]
            (u, v) = arista
            c = comp.get(v, {v})
            if u not in c: # print('u', u, 'v', v ,'c', c)
                arbol.conecta(u, v, w)
                peso += w
                nuevo = c.union(comp.get(u, {u}))
                for i in nuevo:
                    comp[i] = nuevo
        print('MST con peso', peso, ':', nuevo, '\n', arbol.E)
        return arbol
```

Grafo utilizado para el proyecto




```
g=grafo()
g.conecta('a','b', 3)
g.conecta('a','j', 4)
g.conecta('a','h', 4)
g.conecta('b','c', 5)
g.conecta('b','j', 4)
g.conecta('c','d', 7)
g.conecta('c','j', 2)
g.conecta('d','e', 6)
g.conecta('d','i', 2)
g.conecta('d','j', 3)
g.conecta('e','f', 5)
g.conecta('e','i', 3)
g.conecta('f','g', 6)
g.conecta('f','i', 4)
g.conecta('g','h', 5)
g.conecta('g','i', 3)
g.conecta('h','i', 2)
g.conecta('h','j', 3)
g.conecta('i','j', 1)
```

el grafo no fue bien elaborado porque no había buenos grafos como machote

Prueba con cada arista

```
print(g.shortest('a'))
{'a': (0, 'a', ['a']), 'b': (3, 'b', ['b']), 'h': (4, 'h', ['h']), 'j': (4, 'j', ['j']), 'i': (5, 'i', ['i']), 'c': (6, 'c', ['c']),
'd': (7, 'd', ['d']), 'e': (8, 'e', ['e']), 'g': (8, 'g', ['g']), 'f': (9, 'f', ['f'])}
print(g.shortest('b'))
{'b': (0, 'b', ['b']), 'a': (3, 'a', ['a']), 'j': (4, 'j', ['j']), 'c': (5, 'c', ['c']), 'i': (5, 'i', ['i']), 'd': (7, 'd', ['d']),
'h': (7, 'h', ['h']), 'e': (8, 'e', ['e']), 'g': (8, 'g', ['g']), 'f': (9, 'f', ['f'])}
print(g.shortest('c'))
{'c': (0, 'c', ['c']), 'j': (2, 'j', ['j']), 'i': (3, 'i', ['i']), 'b': (5, 'b', ['b']), 'd': (5, 'd', ['d']), 'h': (5, 'h', ['h']),
'a': (6, 'a', ['a']), 'e': (6, 'e', ['e']), 'g': (6, 'g', ['g']), 'f': (7, 'f', ['f'])}
print(g.shortest('d'))
{'d': (0, 'd', ['d']), 'i': (2, 'i', ['i']), 'j': (3, 'j', ['j']), 'h': (4, 'h', ['h']), 'c': (5, 'c', ['c']), 'e': (5, 'e', ['e']),
'g': (5, 'g', ['g']), 'f': (6, 'f', ['f']), 'a': (7, 'a', ['a']), 'b': (7, 'b', ['b'])}
print(g.shortest('e'))
{'e': (0, 'e', ['e']), 'i': (3, 'i', ['i']), 'j': (4, 'j', ['j']), 'd': (5, 'd', ['d']), 'f': (5, 'f', ['f']), 'h': (5, 'h', ['h']),
'c': (6, 'c', ['c']), 'g': (6, 'g', ['g']), 'a': (8, 'a', ['a']), 'b': (8, 'b', ['b'])}
print(g.shortest('f'))
{'f': (0, 'f', ['f']), 'i': (4, 'i', ['i']), 'e': (5, 'e', ['e']), 'j': (5, 'j', ['j']), 'd': (6, 'd', ['d']), 'g': (6, 'g', ['g']),
'h': (6, 'h', ['h']), 'c': (7, 'c', ['c']), 'a': (9, 'a', ['a']), 'b': (9, 'b', ['b'])}
print(g.shortest('g'))
{'g': (0, 'g', ['g']), 'i': (3, 'i', ['i']), 'j': (4, 'j', ['j']), 'd': (5, 'd', ['d']), 'h': (5, 'h', ['h']), 'c': (6, 'c', ['c']),
'e': (6, 'e', ['e']), 'f': (6, 'f', ['f']), 'a': (8, 'a', ['a']), 'b': (8, 'b', ['b'])}
print(g.shortest('h'))
{'h': (0, 'h', ['h']), 'i': (2, 'i', ['i']), 'j': (3, 'j', ['j']), 'a': (4, 'a', ['a']), 'd': (4, 'd', ['d']), 'c': (5, 'c', ['c']),
'e': (5, 'e', ['e']), 'g': (5, 'g', ['g']), 'f': (6, 'f', ['f']), 'b': (7, 'b', ['b'])}
print(g.shortest('i'))
{'i': (0, 'i', ['i']), 'j': (1, 'j', ['j']), 'd': (2, 'd', ['d']), 'h': (2, 'h', ['h']), 'c': (3, 'c', ['c']), 'e': (3, 'e', ['e']),
'g': (3, 'g', ['g']), 'f': (4, 'f', ['f']), 'a': (5, 'a', ['a']), 'b': (5, 'b', ['b'])}
print(g.shortest('j'))
{'j': (0, 'j', ['j']), 'i': (1, 'i', ['i']), 'c': (2, 'c', ['c']), 'd': (3, 'd', ['d']), 'h': (3, 'h', ['h']), 'a': (4, 'a', ['a']),
'b': (4, 'b', ['b']), 'e': (4, 'e', ['e']), 'g': (4, 'g', ['g']), 'f': (5, 'f', ['f'])}
```

Prueba de Kruskal

```
MST con peso 24 : {'e', 'f', 'd', 'g', 'h', 'c', 'b', 'a', 'j', 'i'}
 {('j', 'i'): 1, ('i', 'j'): 1, ('i', 'h'): 2, ('h', 'i'): 2, ('i', 'd'): 2, ('d', 'i'): 2, ('j', 'c'): 2, ('c', 'j'): 2, ('i', 'g'): 3, ('g', 'i'): 3, ('i', 'e'): 3, ('e', 'i'): 3, ('b', 'a'): 3, ('a', 'b'): 3, ('i', 'f'): 4, ('f', 'i'): 4, ('j', 'b'): 4, ('b', 'j'): 4}
```