Reporte de Kruskal

Durante este tema estuvimos viendo algo relacionado a la pasada etapa, que es un algoritmo de aproximación mediante el algoritmo de Kruskal el cual indica el camino más fácil hacia el problema del agente viajero, bueno eso es lo que se verá a continuación

Primero como que mostrare un poco la introducción de problema del agente viajero, este más que nada su objetivo se basa en encontrar un recorrido en el que conecte a todos los nodos de un grafo, aquí he utilizado 10 ciudades visitando a cada una de estas buscando lo más mínimo posible del viaje. (Las ciudades que utilice fueron Puebla, Toluca, Cancun, Tampico, Morelos, Monterrey, México. Acapulco, Monclova y Chiapas)

La dificultad del Problemas del Agente Viajero (PAV) es escoger la ruta que pueda ser más cómoda, más fácil para llegar porque puede haber tantas opciones.

Un algoritmo de aproximación puede hacer muchas cosas minimiza la posibilidad de obtener un buen periodo en un tiempo determinado entre otras funciones pero como que esta es la que más se utiliza el algoritmo de aproximación, además también puede ser un algoritmo usado para encontrar soluciones aproximadas a problemas de optimización.

Ya viendo un poco más o menos sobre un algoritmo de aproximación ahora tocara ver sobre algoritmo de Kruskal y se preguntaran que es este? Pues 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 esta definición la encontré de internet ya que no había entendido muy bien sobre el algoritmo de Kruskal, que es lo busca hacer o así espero poner más atención en la siguientes clases o poder verlo ya que es tema que para mi nuevo y que le podría sacar algo provecho.

Ya explicado un poco sobre el Algoritmo de Kruskal tocara ver lo que es un árbol de expansión mínima este creo que era compuesto por todos los vértices y de las aristas de un grafo, además de esta definición que es lo que pienso yo pude investigar y una definición que había era que puede ser también definido como el mayor conjunto de aristas de grafos que no contiene ciclos, o como el mínimo conjunto de aristas que conecta todos los vértices.

A continuación se mostrara el código en cual se muestra que se realizó el problema del agente viajero con las ciudades que mencione arriba y no sé si me haya salido correctamente pero arrojo lo siguiente.

Codigo

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

def permutation(lst):
    if len(lst) == 0:
        return []
    if len(lst) == 1:
        return [lst]
    l = []
    for i in range(len(lst)):
       m = lst[i]
       remLst = lst[:i] + lst[i+1:]
       for p in permutation(remLst):
           l.append([m] + p)
    return l

class Fila:
    def __init__(self):
        self.fila= []
    def obtener(self):
        return self.fila.pop()
```

```
    def meter(self,e)
        self.fila.insert(0,e)
        return len(self.fila)
    @property
    def longitud(self):
        return len(self.fila)

class Pila:
    def __init__(self):
        self.pila= []
    def obtener(self):
        return self.pila.pop()
    def meter(self,e):
        self.pila.append(e)
        return len(self.pila)
    @property
    def longitud(self):
        return len(self.pila)

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 BFS(self,ni):
    visitados =[]
    f=Fila()
    f.meter(ni)
    while(f.longitud>0):
        na = f.obtener()
        visitados.append(na)
        ln = self.vecinos[na]
        for nodo in ln:
            if nodo not in visitados:
                f.meter(nodo)
    return visitados



def DFS(self,ni):
    visitados =[]
    f=Pila()
    f.meter(ni)
    while(f.longitud>0):
        na = f.obtener()
        visitados.append(na)
```

```
            ln = self.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)
        return visitados
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

    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

def vecinoMasCercano(self):
```

```
        ni = random.choice(list(self.V))
        result=[ni]
        while len(result) < len(self.V):
            ln = set(self.vecinos[ni])
            le = dict()
            res =(ln-set(result))
            for nv in res:
                le[nv]=self.E[(ni,nv)]
            menor = min(le, key=le.get)
            result.append(menor)
            ni=menor
        return result



g= Grafo()
g.conecta('a','b', 114)
g.conecta('a','c', 1448)
g.conecta('a','d', 501)
g.conecta('a','e', 209)
g.conecta('f','g', 913)
g.conecta('b','g', 65)
g.conecta('c','h', 1950)
g.conecta('c','i', 1795)
g.conecta('c','e', 1740)
g.conecta('k','i', 1820)
```

```
print(g.kruskal())
#print(g.shortest('c'))



print(g)
k = g.kruskal()
print([print(x, k.E[x]) for x in k.E])
for r in range(10):
    ni = random.choice(list(k.V))
    dfs =  k.DFS(ni)
    c = 0
    #print(dfs)
    #print(len(dfs))
    for f in range(len(dfs) -1):
            c += g.E[(dfs[f],dfs[f+1])]
            print(dfs[f], dfs[f+1], g.E[(dfs[f],dfs[f+1])] )

    c += g.E[(dfs[-1],dfs[0])]
    print(dfs[-1], dfs[0], g.E[(dfs[-1],dfs[0])])
    print('costo',c)

vmc = g.vecinoMasCercano()
print(vmc)
c=0
```

```
for f in range(len(dfs) -1):
    c += g.E[(vmc[f],vmc[f+1])]
    print(vmc[f], vmc[f+1], g.E[(vmc[f],vmc[f+1])] )

c += g.E[(dfs[-1],dfs[0])]
print(dfs[-1], dfs[0], g.E[(dfs[-1],dfs[0])])
print('costo',c)



data = list('abcde')
#data = ['mty','saltillo', 'chi']
tim=time.clock()
per = permutation(data)
print(time.clock()-tim)
```

De Puebla (a) a Toluca(b) 114
Puebla a Cancun(c) 1448

Monterrey(f) a Ciudad de Mexico (g) 913

Puebla a Tampico(d) 501

Puebla a Morelos(e) 209

Toluca a Ciudad de Mexico 65

Cancun a Acapulco(h) 1950

Cancun Tampico(i) 1795

Cancun a Morelos 1740

Monclova (k) a Chiapas (j)1820

Soluciones

{(“b””a”):65, (“a”,”b”):114, (“a””e”):209, (“a””d”):501, (“f””g”):913, (“a””c”):1448, (“c””e”):1740, (“c””i”):1795, (“j””k”):1820 (“c””h”): 1950

Conclusiones

Este reporte se me hizo complicado por varias razones, una fue que como dije debo de poner atención en clases y aparte como dejo todo al final es más difícil así, mas sin embargo a pesar de todo si logre entender un poco esto de los algoritmos Kruskal, como esta tan relacionado a la vida cotidiana y que es algo muy típico, personalmente me gustaría aprender a bien esta clase ya que uno no sabe y se nos puede llegar a ofrecer.