Reporte

Andrea Estefanía Galván Irigoyen

m.1735583 gpo.001

Matemáticas Computacionales

En este reporte se mostrará de lo que se trata el código de kruskal y cómo implementarlo en Python, el problema del agente viajero, algoritmo de aproximación y expansión mínima, de esta forma utilizar el código de kruskal para sacar algunos datos que se presentarán a continuación.

El algoritmo de kruskal se trata casi lo mismo que dijkstra solamente que en éste lo que cambia es que ahora las aristas de cada nodo tendrán un peso definido y así con el código de kruskal te dará un listado de aristas con el peso mínimo del grafo implementado. Para implementarlo se necesita un grafo ya con los nodos y aristas se les pondrá un peso para cada uno, después se comienza por las aristas de menor peso y así se ira con cada uno siempre y cuando se seleccione la mínima.

Problema de agente viajero

Este se trata de tener varias rutas en el cual el agente necesita visitar cada uno de los puntos, pero este visitante necesita encontrar una ruta con la menor distancia posible recorrida.

Sobre el algoritmo de algoritmo de aproximación nos da una optimización a las soluciones.

Código de kruskal

```
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 árbol
```

a esto solo se le agrega el código de grafos ya visto en reporte anterior.

Código de vecino mas cercano en Python

```
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
```

Código de solución exacta

```
import time
def permutation(lst):
if len(lst) == 0:
return []
if len(lst) == 1:
return [lst]
l = [] # empty list that will store current permutation
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
```

siguiente de este se incorpora el códigode pila, grafo y DFS ya vistos en reportes anteriores

```
def PAV(self):
perm=permutation(list(self.V))
mejor=-1
camino=[]
for w in perm:
peso=0
for i in range(len(w)-1):
peso+=self.E[(w[i],w[i+1])]
peso+=self.E[(w[-1],w[0])]
if peso<mejor or mejor==-1:
mejor=peso
camino=w
return camino
```

asi termina.

Implementando el código kruskal construyendo rutas a 5 puntos para visitar, los siguientes

```
>>> g.conecta('fundi', 'arena mty', .6)
>>> g.conecta('funid', 'museo', 1.5)
>>> g.conecta('fundi', 'morelos', 3.5)
>>> g.conecta('fundi', 'fcfm', 5.4)
>>> g.conecta('arena mty', 'museo', 2.1)
>>> g.conecta('arena mty', 'morelos', 5.1)
>>> g.conecta('arena mty', 'fcfm', 6.5)
>>> g.conecta ('museo', 'morelos', 1.4)
>>> g.conecta('museo', 'fcfm', 6.5)
>>> g.conecta('morelos', 'fcfm', 6.1)
```

Ya probando el código con los datos anteriores dio los siguientes resultados

Primero del parque fundidora a la arena mty, de la arena mty al museo de historia nacional, del museo a morelos y por último de Morelos a la Fcfm

Por el de solución exacta con el siguiente recorrido:

Morelos, fcfm, fundidora, arena mty y museo.