# PROBLEMA DEL AGENTE VIAJERO

Jorge Rodríguez Fernández

Junio 2018

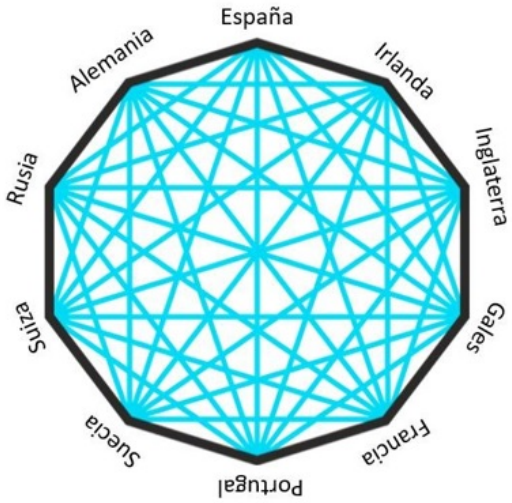
## 1 El problema del agente viajero

Este programa permite al usuario descubrir la ruta más directa por atravesar varios objetos y terminar el recorrido con la menor distancia posible, esto es recorrer todos los puntos sin repetir y a la vez lo más económico.

En el transcurso del semestre se vio diversos métodos de ordenamiento, de búsqueda por profundidad y por anchura, para la resolución de este problema emplearemos la combinación de algunos de los programas vistos en clase.

En el semestre se aprendió el funcionamiento de los programas ya mencionados, y su posible utilización para diferentes fines, en este caso se utilizarán para encontrar la mejor ruta, esto es la que menor distancia, tiempo, dinero o la medición que se utilice, en menor cantidad.

Este es mi grafo de países:



Aquí se muestran 10 países y sus conexiones entre sí, la medida utilizada es el tiempo que se tarde de llegar de un país a otro en horas en avión, para representar de una manera clara que todos los países están conectados entre sí, se escogió un decágono con sus diagonales, así o se repite ninguna arista y se asegura que estén interconectados.

Esta es la representación del grafo en un código en Python:

```
g= Grafo()
g.conecta('España','Alemania',3)
g.conecta('España','Rusia',6)
g.conecta('España','Suiza',3)
g.conecta('España','Suecia',5)
g.conecta('España','Portugal',1)
g.conecta('España','Francia',2)
g.conecta('España','Gales',3)
g.conecta('España','Inglaterra',4)
g.conecta('España','Irlanda',3)
g.conecta('Alemania','Rusia',4)
g.conecta('Alemania','Suiza',1)
g.conecta('Alemania','Suecia',2)
g.conecta('Alemania','Portugal',4)
g.conecta('Alemania','Francia',1)
g.conecta('Alemania','Gales',4)
g.conecta('Alemania','Inglaterra',3)
g.conecta('Alemania','Irlanda',4)
g.conecta('Rusia','Suiza',4)
g.conecta('Rusia','Suecia',3)
g.conecta('Rusia','Portugal',7)
g.conecta('Rusia','Francia',5)
g.conecta('Rusia','Gales',8)
g.conecta('Rusia','Inglaterra',8)
g.conecta('Rusia','Irlanda',9)
g.conecta('Suiza','Suecia',4)
g.conecta('Suiza','Portugal',4)
g.conecta('Suiza','Francia',1)
g.conecta('Suiza','Gales',3)
g.conecta('Suiza','Inglaterra',3)
g.conecta('Suiza','Irlanda',4)
g.conecta('Suecia','Portugal',6)
g.conecta('Suecia','Francia',4)
g.conecta('Suecia','Gales',4)
g.conecta('Suecia','Inglaterra',3)
g.conecta('Suecia','Irlanda',4)
g.conecta('Portugal','Francia',3)
```

```
g.conecta('Portugal','Gales',4)
g.conecta('Portugal','Inglaterra',4)
g.conecta('Portugal','Irlanda',5)
g.conecta('Francia','Gales',2)
g.conecta('Francia','Inglaterra',2)
g.conecta('Francia','Irlanda',3)
g.conecta('Gales','Inglaterra',1)
g.conecta('Gales','Irlanda',1)
g.conecta('Inglaterra','Irlanda',1)
```

El código es el siguiente:

```
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 = []  # 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


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):  # 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
        p = (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
```

```
    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

print(g.kruskal())
MST con peso 14 :
{'Portugal', 'Inglaterra', 'Francia', 'Suiza', 'Alemania', 'Irlanda', 'Gales',
'España', 'Rusia', 'Suecia'}
 {('Irlanda', 'Inglaterra'): 1, ('Inglaterra', 'Irlanda'): 1, ('Irlanda', 'Gales'): 1,
 ('Gales', 'Irlanda'): 1,
('Francia', 'Suiza'): 1, ('Suiza', 'Francia'): 1, ('Francia', 'Alemania'): 1,
```

```
('Alemania', 'Fra
ncia'): 1, ('Portugal', 'España'): 1, ('España', 'Portugal'): 1, ('Inglaterra',
'Francia'): 2, ('Francia', 'Inglaterra'): 2, ('Suecia', 'Alemania'): 2, ('Alemania',
'Suecia'): 2, ('Francia', 'España'): 2, ('
España', 'Francia'): 2, ('Suecia', 'Rusia'): 3, ('Rusia', 'Suecia'): 3}
<__main__.Grafo object at 0x02807050>
print(g)
<__main__.Grafo object at 0x02807210>
>>> k = g.kruskal()
MST con peso 14 : {'Portugal', 'Inglaterra', 'Francia', 'Suiza', 'Alemania',
'Irlanda', 'Gales', 'España', 'Rusia', 'Suecia'}
{('Irlanda', 'Inglaterra'): 1, ('Inglaterra', 'Irlanda'): 1, ('Irlanda', 'Gales'): 1,
('Gales', 'Irlanda'): 1,('Francia', 'Suiza'): 1, ('Suiza', 'Francia'): 1, ('Francia',
'Alemania'): 1, ('Alemania', 'Francia'): 1, ('Portugal', 'España'): 1, ('España',
'Portugal'): 1, ('Inglaterra', 'Francia'): 2, ('Francia', 'Inglaterra'): 2, ('Suecia',
'Alemania'): 2, ('Alemania', 'Suecia'): 2, ('Francia', 'España'): 2, ('España',
'Francia'): 2, ('Suecia', 'Rusia'): 3, ('Rusia', 'Suecia'): 3}
>>> print([print(x, k.E[x]) for x in k.E])
('Irlanda', 'Inglaterra') 1
('Inglaterra', 'Irlanda') 1
('Irlanda', 'Gales') 1
('Gales', 'Irlanda') 1
('Francia', 'Suiza') 1
('Suiza', 'Francia') 1
('Francia', 'Alemania') 1
('Alemania', 'Francia') 1
('Portugal', 'España') 1
('España', 'Portugal') 1
('Inglaterra', 'Francia') 2
('Francia', 'Inglaterra') 2
('Suecia', 'Alemania') 2
('Alemania', 'Suecia') 2
('Francia', 'España') 2
('España', 'Francia') 2
('Suecia', 'Rusia') 3
('Rusia', 'Suecia') 3
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)
```

Donde nos da los siguientes resultados:

Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Suiza 4, Suiza Inglaterra 3, Inglaterra Irlanda 1, Irlanda Gales 1, Gales España 3, España Portugal 1, Portugal Francia 3, costo 22

España Francia 2, Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Suiza 4, Suiza Inglaterra 3, Inglaterra Irlanda 1, Irlanda Gales 1, Gales Portugal 4, Portugal España 1, costo 22

Rusia Suecia 3, Suecia Alemania 2, Alemania suiza 1, Suiza Francia 1, Francia Gales 2, Gales Inglaterra 1, Inglaterra Irlanda 1, Irlanda España 3, España Portugal 1, Portugal Rusia 7, costo 22

Alemania Suecia 2, Suecia Rusia 3, Rusia Francia 5, Francia Suiza 1, Suiza Inglaterra 3, Inglaterra Irlanda 1, Irlanda Gales 1, Gales España 3, España Portugal 1, Portugal Alemania 4, costo 24

Portugal España 1, España Francia 2, Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Suiza 4, Suiza Inglaterra 3, Inglaterra Irlanda 1, Irlanda Gales 1, Gales Portugal 4, costo 22

Inglaterra Irlanda 1, Irlanda Gales 1, Gales Francia 2, Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Suiza 4, Suiza España 3, España Portugal 1, Portugal Inglaterra 4, costo 22

Suiza Francia 1, Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Inglaterra 8, Inglaterra Irlanda 1, Irlanda Gales 1, Gales España 3, España Portugal 1, Portugal Suiza 4, costo 25

Suecia Alemania 2, Alemania suiza 1, Suiza Francia 1, Francia Gales 2, Gales Inglaterra1, Inglaterra Irlanda 1, Irlanda España 3 , España Portugal 1, Portugal Rusia 7, Rusia Suecia 3, Costo 22

Gales Irlanda 1, Irlanda Inglaterra 1, Inglaterra Francia 2, Francia Alemania 1, Alemania Suiza 1, Suiza España 3, España Portugal 1, Portugal Suecia 6, Suecia Rusia 3, Rusia Gales 8, costo 27

Irlanda Inglaterra 1, Inglaterra Gales 1, Gales Francia 2, Francia Alemania 1, Alemania Suiza 1, Suiza España 3, España Portugal 1, Portugal Suecia 6, Suecia Rusia 3, Rusia Irlanda 9, Costo 28

Por lo tanto, tenemos que la mejor ruta, en este caso la ruta en la cual el viaje es el mínimo de horas es el siguiente:

Portugal España 1, España Francia 2, Francia Alemania 1, Alemania Suecia 2, Suecia Rusia 3, Rusia Suiza 4, Suiza Inglaterra 3, Inglaterra Irlanda 1, Irlanda Gales 1, Gales Portugal 4

Con un total de 22 horas de vuelo

Este programa nos arroga diferentes resultados con cifras semejantes, las cuales nos darían el mismo resultado, lo cual podría decir que hay mas de una ruta con la cual podríamos hacer el recorrido, o no se podría arrojar un solo resultado viable, dependiendo de la situación.