# UNIVERSIDAD AUTONOMA DE NUEVO LEON

# FACULTAD DE CIENCIAS FISICO MATAMACIAS

## PROYECTO FINAL DE MATEMATICAS COMPUTACIONALES VERSION 2

**NOMBRE:** MELINA PLATAS SANCHEZ

**MATRICULA:** 1740967

**MATERIA:** MATEMATICAS COMPUTACIONALES

**MAESTRO:** JOSE ANASTACIO HERNANDEZ SALDAÑA

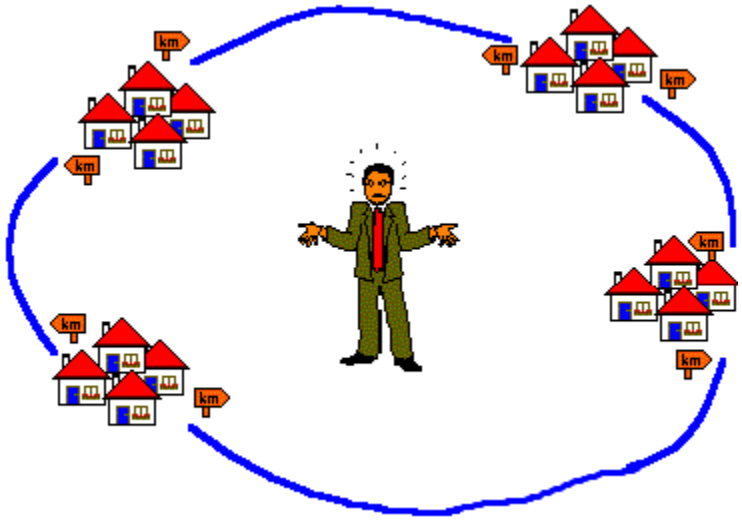
**FECHA DE ENTREGA:** 6 DE JUNIO DEL 2018

# INTRODUCCION

En este trabajo se realizara un programa en donde se explique la ruta más corta y en cuanto tiempo se realiza en 10 lugares que son países del mundo. No solamente usaremos Kruskal si no también ocuparemos de un programa llamado vecino más cercano. Y para este proyecto se necesita esencialmente conocer acerca de grafos para poder realizar las rutas más cortas.

Este proyecto está considerado como un conjunto de grafos cuyas aristas son vistas como las rutas a recorrer para visitar a todos los nodos.

El objetivo que se plantea de este problema es encontrar un recorrido que conecte todos los nodos de un conjunto de modo que los visite solo una vez, regresando a su nodo de inicio y además de esto sea mínima la distancia que se tiene que recorrer.



# DEFINICIONES QUE SON NECESARIAS.

***GRAFO:***

Es un conjunto de objetos llamados vértices o nodos unidos por enlaces llamados aristas o arcos, que permite relaciones binarias entre elementos de un conjunto.

***ALGORITMO:***

Es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite llevar a cabo una actividad mediante pasos sucesivos.

***ARBOL:***

Es una gráfica el cual no existen ciclos.

***ALGORITMO KRUSKAL:***

Es un algoritmo para encontrar un árbol de expansión mínima en un grafo ponderado.

***ALGORITMO VECINO MÁS CERCANO:***

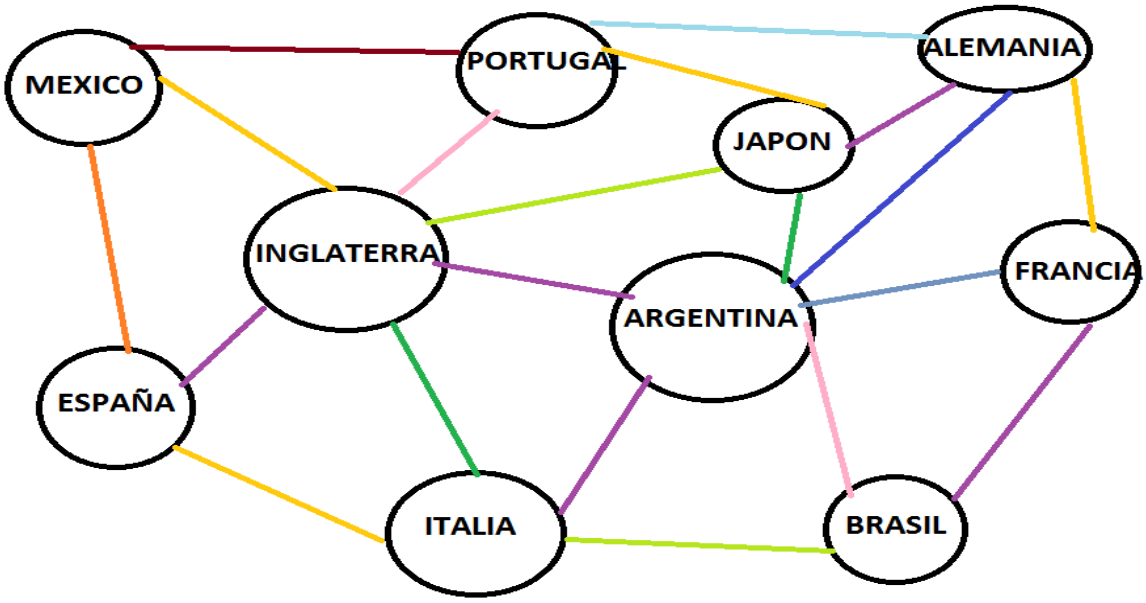
También llamado algoritmo voraz (greedy) permite al vigilante elegir la ciudad no visitada más cercana como próximo movimiento

# GRAFO

El proyecto consiste en llevar a cabo la implementación de un algoritmo de aproximación usando el algoritmo de kruskal para obtener un árbol de expansión mínima y a partir de él crear una solución aproximada al problema del agente viajero. En primera instancia se eligieron 10 países que representan los nodos y se calcularon las distancias existentes entre ellas representando las aristas de nuestro grafo. Partiendo de lo anterior se tuvo como objetivo encontrar un recorrido que conecte se todas las ciudades, visitando cada una de ellas para después volver a nuestro punto de partida, buscando minimizar la distancia de la ruta.

Los países seleccionados fueron México, España, Inglaterra Portugal, Italia, Argentina, Alemania, Japón, Francia y Brasil.

A continuación se muestra el grafo de los países seleccionados con sus respectivas aristas entre ellas.



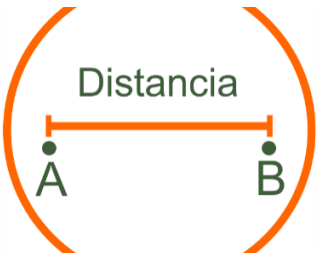
Ahora se mostrará la relación que hay de las distancias entre los países en donde su distancia es la misma como de ida a como de regreso, en la siguiente tabla se menciona la relación de todos los países entre ellos (el número que está a lado de cada relación entre dos países es el número de los miles de kilómetros que se tienen entre ellos)

TABLA DE DISTANCIAS ENTRE PAISES

| PAIS (INICIO) | PAIS (FINAL) | KILOMETROS |
|---|---|---|
| MEXICO | ESPAÑA | 8.9 |
| MEXICO | INGLATERRA | 8.6 |
| MEXICO | PORTUGAL | 8.6 |
| MEXICO | ITALIA | 10 |
| MEXICO | ARGENTINA | 7 |
| MEXICO | ALEMANIA | 9.3 |
| MEXICO | JAPON | 10.5 |
| MEXICO | FRANCIA | 9.1 |
| MEXICO | BRASIL | 6.5 |
| ESPAÑA | INGLATERRA | 1.3 |
| ESPAÑA | PORTUGAL | 0.4 |
| ESPAÑA | ITALIA | 1.4 |
| ESPAÑA | ARGENTINA | 9.5 |
| ESPAÑA | ALEMANIA | 1.6 |
| ESPAÑA | JAPON | 10.8 |
| ESPAÑA | FRANCIA | 0.8 |
| ESPAÑA | BRASIL | 7.6 |
| INGLATERRA | PORTUGAL | 1.5 |
| INGLATERRA | ITALIA | 1.4 |
| INGLATERRA | ARGENTINA | 10.5 |
| INGLATERRA | ALEMANIA | 0.7 |
| INGLATERRA | JAPON | 9.5 |
| INGLATERRA | FRANCIA | 0.7 |

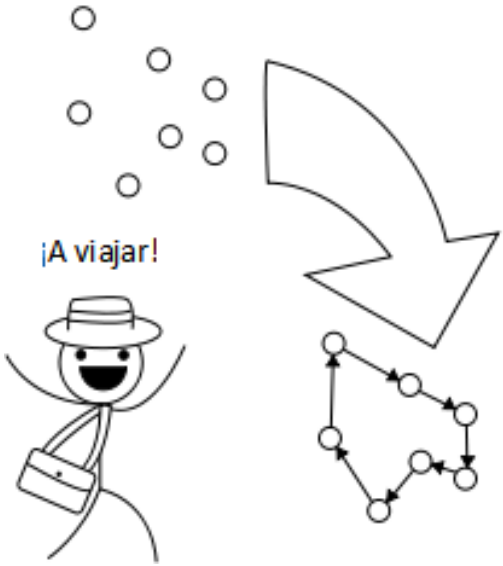
| | | |
|---|---|---|
| INGLATERRA | BRASIL | 8.6 |
| PORTUGAL | ITALIA | 1.8 |
| PORTUGAL | ARGENTINA | 9.2 |
| PORTUGAL | ALEMANIA | 1.9 |
| PORTUGAL | JAPON | 11 |
| PORTUGAL | FRANCIA | 1.1 |
| PORTUGAL | BRASIL | 7.3 |
| ITALIA | ARGENTINA | 10.9 |
| ITALIA | ALEMANIA | 0.9 |
| ITALIA | JAPON | 9.7 |
| ITALIA | FRANCIA | 0.9 |
| ITALIA | BRASIL | 9 |
| ARGENTINA | ALEMANIA | 11.1 |
| ARGENTINA | JAPON | 17.4 |
| ARGENTINA | FRANCIA | 10.3 |
| ARGENTINA | BRASIL | 1.9 |
| ALEMANIA | JAPON | 9.1 |
| ALEMANIA | FRANCIA | 0.8 |
| ALEMANIA | BRASIL | 9.2 |
| JAPON | FRANCIA | 9.9 |
| JAPON | BRASIL | 16.7 |
| FRANCIA | BRASIL | 8.4 |

Esta es cantidad total que hay de distancia de miles kilómetros entre todos los diez países.

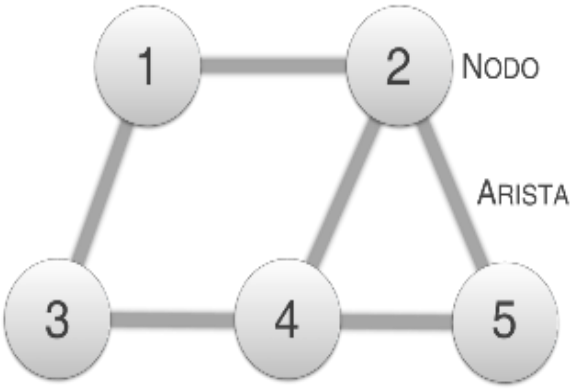


# EL PROBLEMA DEL AGENTE VIAJERO

Al tomar cierta cantidad de lugares y el costo de cada viaje de un lugar a otro lugar, se encontrará la forma más barata de poder ir visitando cada lugar exactamente sólo una vez y volver al punto de origen. En este problema, su dificultad depende de la implementación de las posibilidades que tiene al realizar los recorridos, donde se están comparando las rutas con el menor peso, para así poder deducir el mejor recorrido posible que se puede hacer con el menor gasto. Así, su complejidad aumenta al momento de crecer los lugares que recorrerá, ya que tendrá más variables y rutas de viajes.



# ALGORITMO KRUSKAL




Un algoritmo de aproximación KRUSKAL es aquel en el cual se menciona que es un algoritmo exacto en donde encuentra soluciones óptimas y requieren tiempos superpolinomiales, o sea, su tiempo es exponencial. El objetivo de este algoritmo es formar una aproximación de un resultado deseado.

A continuación se muestra el algoritmo:

Kruskal.py - C:/Users/isaac/Desktop/Mat Comp/Kruskal.py (3.6.2)

File Edit Format Run Options Window Help

```
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
```

# RESULTADOS

A continuación el programa expulso los resultados siguiendo este orden de países.

['JAPON', 'ALEMANIA', 'INGLATERRA', 'FRANCIA', 'ESPAÑA', 'PORTUGAL', 'ITALIA', 'BRASIL', 'ARGENTINA', 'MEXICO']

| PAIS (INCIO) | PAIS (FINAL) | KILOMETROS |
|---|---|---|
| JAPON | ALEMANIA | 9.1 |
| ALEMANIA | INGLATERRA | 0.7 |
| INGLATERRA | FRANCIA | 0.7 |
| FRANCIA | ESPAÑA | 0.8 |
| ESPAÑA | PORTUGAL | 0.4 |
| PORTUGAL | ITALIA | 1.8 |
| ITALIA | BRASIL | 9 |
| BRASIL | ARGENTINA | 1.9 |
| ARGENTINA | MEXICO | 7 |
| MEXICO | JAPON | 10.5 |

Esta tabla muestra los resultados obtenidos con el algoritmo de kruskal y tiene un costo de 41.9

# ALGORITMO HEURISTICA

Se denomina heurística al arte de inventar. En programación se dice que un algoritmo es heurístico cuando la solución no se determina en forma directa, sino mediante ensayos, pruebas y reensayos.

El método consiste en generar candidatos de soluciones posibles de acuerdo a un patrón dado; luego los candidatos son sometidos a pruebas de acuerdo a un criterio que caracteriza a la solución. Si un candidato no es aceptado, se genera otro; y los pasos dados con el candidato anterior no se consideran. Es decir, existe inherentemente una vuelta atrás, para comenzar a generar un nuevo candidato; por esta razón, este tipo de algoritmo también se denomina "con vuelta atrás"

Una vez ya bien definido el algoritmo heurística se mostrara el programa para resolver dicho algoritmo.

***Implementación:***

```
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
```

# Solución al algoritmo heurística

Para esto usaremos los países siguientes en este orden:

['España ', 'Portugal', 'Francia', 'Inglaterra', 'Alemania', 'Italia', 'Brasil', 'Argentina', 'México', 'Japón']

| PAIS | PAIS | KILOMETROS |
|---|---|---|
| ESPAÑA | PORTUGAL | 0.4 |
| PORTUGAL | FRANCIA | 0.7 |
| FRANCIA | INGLATERA | 0.7 |
| INGLATERA | ALEMANIA | 0.7 |
| ALEMANIA | ITALIA | 0.9 |
| ITALIA | BRASIL | 9 |
| ITALIA | BRASIL | 9 |
| BRASIL | ARGENTINA | 1.9 |
| ARGENTINA | MEXICO | 7 |
| MEXICO | JAPON | 10.5 |

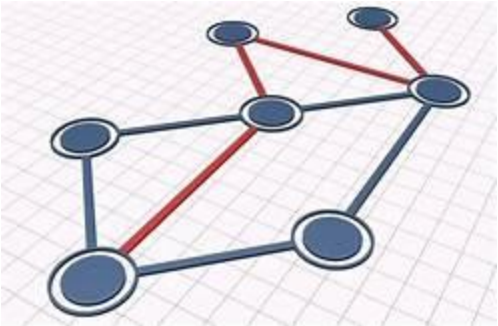
Y el resultado final de este algoritmo tuvo un costo de 43.0

Ya para concluir sobre este programa, es importante tener en claro que este algoritmo se enfoca a resolver el problema que planteamos desde el principio, pero tiene diferentes resultados, así que se escogerá el resultado que más convenga o el más esencial en ambos sentidos. Además, aunque este algoritmo sirve mucho para nuestro caso de recorrer lugares, también puede ser usado para muchas más cosas. Por último, con este programa se puede optimizar el tiempo y las distancias con el simple hecho que te ayuda a tener mejores recorridos al momento de viajar ya que ese será nuestro objetivo desde el principio.

## DIJKSTRA

En este algoritmo nos dimos cuenta de que sirve para caber cual es el camino más corto al momento de darle un nodo inicial con el resto de los nodos en un grafo el dónde en cada arista tiene un peso. Este algoritmo explora todos los caminos más cortos que parten desde el nodo inicial y llevan al resto de los nodos, cuando por fin se obtiene el camino más corto el algoritmo se detiene.

También se puede decir que es usado para encontrar la ruta más corta, ya que cada arista tiene un peso, el algoritmo encontrará el camino con menos peso. Hay que mencionar que este algoritmo sólo funciona con pesos positivos.



¿Cómo se programa?

Para empezar, hay que marcar todos los vértices como no visitados.

Empezando del nodo inicial, evaluamos las aristas para poder encontrar el que tenga un camino más corto, ósea, con menor peso, como se mencionó antes. Después se compara la arista con los demás nodos para ver si se puede llegar más rápido, una vez que ya tengamos el nodo más cercano, se repite otra vez el proceso, así estamos obteniendo la arista con el menor peso que se pueda. Se finaliza cuando llega hasta nuestro nodo distinto.


En la otra página se muestra el algoritmo de dicho programa.

File Edit Format Run Options Window Help

```
from heapq import heappop, heappush


def flatten(L):
    while len(L) > 0:
        yield L[0]
        L = L[1]


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

    def shortest(self, v): # Algoritmo Dijkstra
        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
```

POR ULTIMO….

Los resultados muestran que en las pruebas realizadas, el que obtuvo mejores soluciones fue el algoritmo de la heurística y en menos tiempo, desde nuestro punto de vista y para fines más prácticos usaríamos el algoritmo de la heurística, además de que es lo más cercano a la lógica humana, ya que por lo regular si estamos en lugar y tenemos que ir a otro lugares, generalmente nos dirigimos hacia el lugar que esté más cerca de donde estamos actualmente. Y no pensamos que el algoritmo de aproximación esté equivocado o sea ineficiente, solo que por simplicidad usaríamos el de la heurística.

Gracias por este semestre maestro. :D