

```

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)

```

Estructura Pila: esta estructura está formada por un conjunto de elementos acomodados de manera vertical, de manera que cada elemento que agreguemos se pondrá en la parte superior de los elementos

¿Cómo funciona? Esta funciona de tal manera que, si ingresamos 3, 5, 7, 9 como elementos de la pila al momento de pedirle que imprima un valor de la pila nos dará como resultado el número 9 que fue el último elemento ingresado

```

class Fila:
    def __init__(self):
        self.fila = []
    def obtener(self):
        return self.fila.pop(0)
    def meter(self, e):
        self.fila.append(e)
        return len(self.fila)
    @property
    def longitud(self):
        return len(self.fila)

```

Estructura fila: en este tipo de estructura el primer elemento que ingresamos será el primero en salir, ya que este tipo de estructura es del tipo FIFO (first in, first out)

¿Cómo funciona? Si ingresamos los elementos 1, 4, 7, 18, 20 al momento de pedirle que imprima el primer valor, nos dará como resultado el 1, por su tipo de estructura

```

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 agregar(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.agregar(v)
        self.agregar(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

```

Grafo: esta estructura está formada por objetos llamados nodos o vértices que están unidos mediante enlaces llamados aristas

```

def DFS(g, ni):
    visitados = []
    f = Pila()
    f.meter(ni)
    while f.longitud > 0:
        na = f.obtener()
        if na not in visitados:
            visitados.append(na)
            ln = g.vecinos[na]
            for nodo in ln:
                if nodo not in visitados:
                    f.meter(nodo)
    return visitados

```

DFS: dado un grafo DFS recorre cada nodo, lo va revisando uno por uno y si este a su vez está unido con otro igual lo revisa

BFS: este es parecido al DFS, lo diferente es que este primero revisa a los nodos principales, después a los secundarios y así se va

```

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

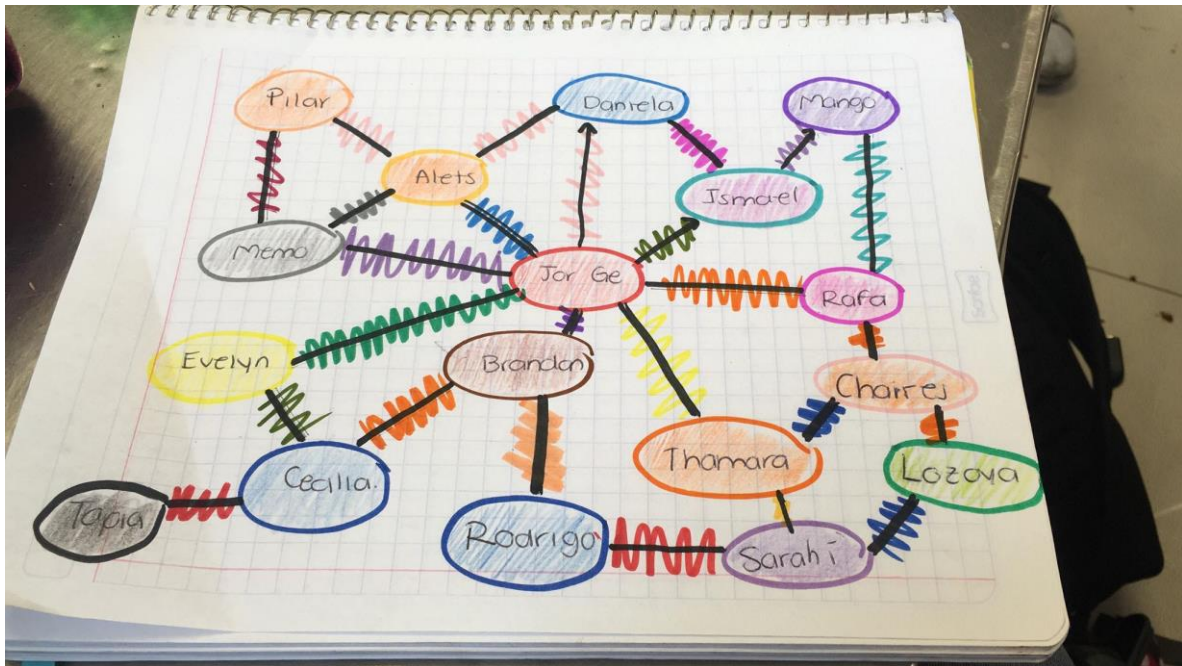
```

Mi grafo:

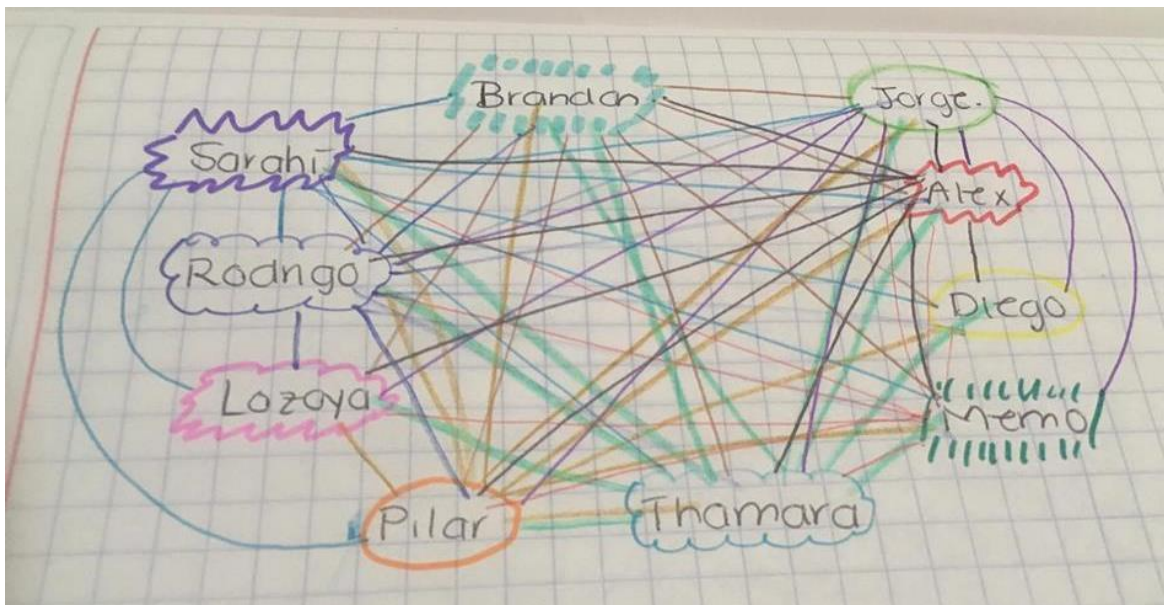
```

g = Grafo()
g.conecta('JorGe', 'Alets', 1)
g.conecta('JorGe', 'Daniela', 4)
g.conecta('JorGe', 'Ismael', 3)
g.conecta('JorGe', 'Rafa', 4)
g.conecta('JorGe', 'Thamara', 2)
g.conecta('JorGe', 'Brandon', 1)
g.conecta('JorGe', 'Evelyn', 1)
g.conecta('JorGe', 'Memo', 3)
g.conecta('Memo', 'Pilar', 2)
g.conecta('Memo', 'Alets', 3)
g.conecta('Alets', 'Pilar', 2)
g.conecta('Alets', 'Daniela', 2)
g.conecta('Ismael', 'Mango', 3)
g.conecta('Rafa', 'Chaires', 2)
g.conecta('Rafa', 'Mango', 1)
g.conecta('Chaires', 'Lozoya', 3)
g.conecta('Chaires', 'Thamara', 4)
g.conecta('Thamara', 'Sarahi', 5)
g.conecta('Brandon', 'Cecilia', 2)
g.conecta('Brandon', 'Rodrigo', 3)
g.conecta('Rodrigo', 'Sarahi', 2)
g.conecta('Cecilia', 'Evelyn', 3)
g.conecta('Cecilia', 'Tapia', 2)

```



**Grafo completo:**





El número total de un grafo de  $n$  aristas se da por  $\frac{(n)(n-1)}{2}$  entonces lo único que se tiene que hacer es dividir las aristas que hemos encontrado entre el número total de aristas

```
def densidad(self):
    cantidadAristas = len(self.E)
    cantidadVertices = len(self.V)
    cantidadMaximaAristas = cantidadAristas * (cantidadVertices - 1)
    if cantidadMaximaAristas < 1:
        cantidadMaximaAristas = 1
    dens = cantidadAristas / cantidadMaximaAristas
    return dens
```

### Centralidad de un grafo:

Se refiere a una medida posible de un nodo en el grafo y determina la importancia de este, este se mide ya sea por su importancia, influencia, relevancia o prominencia, por centralidad se mide por cual números de enlaces que posee un nodo con los demás nodos que conforman al grafo

### Problema del agente viajero:

Este problema es considerado como un conjunto de grafos cuyas aristas son vistos como las rutas que hay que recorrer para visitar a todos los nodos

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

### Algoritmo de Dijkstra

También llamado algoritmo de caminos mínimos, este algoritmo busca el camino más corto dado un vértice origen al resto de los vértices.

Este algoritmo recorre los nodos, partiendo de un nodo inicial, donde recorre todos los posibles caminos que conecta a un nodo con otro, después de recorrer y calcular la distancia más corta de todas las rutas, este algoritmo termina

### Algoritmo de kruskal

¿Cómo funciona? Primero ordena los aristas por su peso de menor a mayor, este intentara unir cada arista siempre y cuando no se forme un ciclo

Sarahi Sanchez Cruz, 17431312

