Segundo Parcial

Nombre: Alejandro Paredes La Torre
Registro: 22000219 2
Pregunta 6.

1. Se debe implementar una clase con nodos o aristas que mapee las relaciones mediante una función principal el detalle de relación

Ej. implementación

```
class Vertices:
    def __init__(self, clave):
        self.id = clave
        self.conectadoA = {}
    def agregarVecino(self, vecino, ponderacion=0):
        self.conectadoA[vecino] = ponderacion
    def __str__(self):
        return str(self.i) + 'conectadoA: ' + str([x.id for x in self.conectadoA])
    def obtenerConexiones(self):
        return self.conectadoA.keys()
    def obtenerId(self):
        return self.id

class Grafo:
    def __init__(self):
        self.listaVertices = {}
        self.numVertices = 0
    def agregarVertice(self, clave):
        self.numVertices = self.numVertices + 1
        nuevoVertice = Vertices(clave)
        self.listaVertices[clave] = nuevoVertice
        return nuevoVertice
    def agregarArista(self, de, a, costo=0):
        if de not in self.listaVertices:
            nv = self.agregarVertice(de)
        if a not in self.listaVertices:
            nv = self.agregarVertice(a)
        self.listaVertices[de].agregarVecino(self.listaVertices[a], costo)
```

Segundo parcial
Estructuras de datos II – MV

Nombre: Alejandro Paredes La Torre
Registro: 22000219-2

Pregunta 7
Implementar un algoritmo que permita calcular el coste mínimo y el camino mas corto entre un punto origen a uno destino

```
class Graph:
    def __init__(self,V:int):
        self.V=V
        self.adj=[[]for _ in range(V)]
    def addEdge(self, u: int, v:int, w: int): #Añadir aristas
        self.adj[u].append((v, w))
        self.adj[v].append((u,w))
    def shortestPath(self, src:int, W:int):  # Encontrar camino mas corto
        dist=[[INF,None] for _ in range(self.V)]
        dist[src][0]=0
        B=[[] for _ in range(W*self.V+1)]
        B[0].append(src)
        idx=0
        while True:
            while len(B[idx])==0 and idx<W*self.V:
                idx+=1
            if idx== W*self.V
                break
            u=B[idx][0]
            B[idx].pop(0)
            for v, weight in self.adj[u]:
                du=dist[u][0]
                dv=dist[v][0]
                if dv>du+weight:
                    if dv!=INF:
                        B[dv].remove(v)
                    dist[v][0]=du+weight
                    dv=dist[v][0]
```

→ Continua atras

```
            S[dv].append(v)
            dist[v][1] = len(S[dv]) - 1
    for i in range(self.V):
        print(f"{i}, {dist[1][0]}")
```