```
#Creador: Eduardo Javier Maldonado Acevedo
#Port a python 3 y adaptaciones Matías Bordone
import os
import sys

class Grafo:
def __init__(self):
self.grafo = {}
def cargarEjemplo1(self): # carga un grafo de ejemplo para probar funciones
self.grafo ={ 'A' : {'F': 0, 'G': 0},
'F': {},
'G': {'B': 0, 'U': 0, 'X': 0},
'B': {},
'U': {'Z': 0, 'D': 0},
'X': {},
'Z': {},
'D': {},
}
def __str__(self):
return (str(self.grafo))
def cargarEjemplo2(self): # carga un grafo de ejemplo para probar funciones
self.grafo = {'TIJ':{'MTY': 800},
'MZT':{'TIJ': 400, 'BJX': 300},
'MTY':{'BJX': 700},
'BJX':{'SAN': 900, 'TAM': 400, 'MEX': 350},
'GDL':{'MZT': 500, 'BJX': 250, 'MEX': 500, 'MTY': 450},
'CUN':{'GDL': 650},
'MEX':{'CUN': 650, 'MID': 450, 'CH': 50},
'TAM':{'MID': 450},
'SAN':{'MID': 1200},
'MID':{},
'CH':{'TAM': 50},
}
def agregarv(self,v): # agrega un vertice al grado
if not self.existev(v):
self.grafo[v]= {}

def existev(self,vertice): # v -> Bool
return vertice in self.grafo

def borrarv(self,v): #borra el vertice v del grafo
if v in self.grafo: # preg si la clave esta en el grafo
for w in self.grafo: # si es vecino de otro
if v in self.grafo[w]:
self.grafo[w].pop(v) # lo borra
del self.grafo[v] # y lo borra del del diccionario

def agregara(self,v,w,p): # agrega la arista vw al grafo con el peso p (deben existir v y w)
if self.existev(v) and self.existev(w):
self.grafo[v][w] = p

def borrara(self,v,w): #borra la arista vw
if v in self.grafo and w in self.grafo[v]:
del self.grafo[v][w]

def peso(self,v,w): # devuelve el peso de la arista vw
if v in self.grafo and w in self.grafo[v]:
return self.grafo[v][w]

def existea(self,v,w):
return w in self.grafo[v]

def vecinos(self,v): # v -> [v]
vecinos = []
```

```
if v in self.grafo:
vecinos.append(v)
else:
vecinos = []
return vecinos

def nvertices(self): # devuelve la cantidad de vertices de G
return len(self.grafo)

def vertices(self): # devuelve una lista con los vertices de G
return list(self.grafo.keys())

def borrarGrafo(self): # borra el grafo
self.grafo = {}

def cargardearchivo(self, archivo): # carga el grafo desde "archivo"
fd = open(archivo, 'r') # fd puntero del archivo
n, m = list(map(int, fd.readline().split()))
self.borrarGrafo()
for i in range(m):
print(i)
v1, v2, p = list(map(int, fd.readline().split()))
self.agregarv(v1)
self.agregarv(v2)
self.agregara(v1,v2,p)
fd.close()



def Menu(self):
print (
"""============== Algoritmos de Grafos ==============<
[1] Mostrar Grafo Completo
[2] Algoritmo Dijkstra
[3] Cargar grafo de un archivo
[4] DFS
[5] BFS
[6] Cargar grafo de ejemplo 1
[7] Cargar grafo de ejemplo 2
[0] Salir
>=============================================<""")

def borrarPantalla(self): #Definimos la función estableciendo el nombre que queramos
if os.name == "posix":
os.system ("clear")
elif os.name == "ce" or os.name == "nt" or os.name == "dos":
os.system ("cls")

def seleccion(self): # Muestra el menu y permite interactuar
g=self
while True:
self.borrarPantalla()
g.Menu()
opcion = input(">=> Ingresa tu opcion: ")
if opcion=="1":
g.GrafoCompleto()
elif opcion=="2":
origen = input(">=> Ingresa el origen: ")
lladatoe = self.grafo.keys()
if origen not in lladatoe:
print ("El origen no existe")
input("Presione enter para continuar")
else:
destino = input(">=> Ingresa el destino: ")
if destino not in lladatoe:
```

```
print ("El destino no existe")
input("Presione enter para continuar")
else:
g.MetodoDijkstra(self.grafo,origen,destino)
elif opcion=="6":
self.cargarEjemplo1()
elif opcion=="7":
self.cargarEjemplo2()
elif opcion=="0":
break
else:
print ("No has ingresado una opcion correcta")
input("Presione enter para continuar")

def MetodoDijkstra(self,grafo,origen,destino): # Ejecuta el algoritmo de dijkstra
arbol, distancia, padre = {}, {}, {}
cont = 0
ordenada = []
for dato in grafo:
arbol[dato] = False
distancia[dato] = float("inf")
padre[dato] = ''
distancia[origen] = 0
dato = origen
while dato != destino and arbol[dato] is False:
arbol[dato] = True
ady = grafo[dato]
'''print "Padre:",dato
print "Adyacentes:",ady
input("Presione enter para continuar")'''
for q, peso in ady.items():
if distancia[q] > distancia[dato]+peso:
distancia[q] = distancia[dato]+peso
padre[q] = dato
dato = min((a for a in grafo if arbol[a] is False), key=lambda a:distancia[a]) #recorre
ordenada = list(distancia.items()) #el grafo tomando sus claves y valores, toma los val
print ("==================================") # minimos y los guarda en una lista
print ("Paso:",cont)
print ("==================================")
print ("Ordenada:",ordenada)
print ("==================================")
cont += 1
ordenada = [(a,b) for b,a in ordenada]
ordenada.sort()
ordenada = [(a,b) for b,a in ordenada]

if padre[destino]:
camino = [destino]
dato = destino
print ("========== Resultados ==========")
while dato != origen:
camino.append(padre[dato])
print ("Camino:",camino)
dato = padre[dato]
extraido = camino[0]
for a,b in ordenada:
if extraido == a:
peso = b
print ("Costo:",peso)
print ("==================================")
input("Presione enter para continuar")
return camino[::-1]
else:
return None
```

```
def bfs(self,grafo): # v:vertice -> [vertice]
resultado = []
cola = []
while cola != []:
v = cola.pop()
for i in vecinos(v):
if i not in resultado:
cola.insert(0,i)
resultado.append(i)
return resultado

def dfs(self, grafo):
visitado = [False]
stack = grafo[inicial]
resultado = [inicial]
visitado[inicial] = True

while stack:
actual = stack[-1]
vecinos = grafo[actual]
no_visitado = sorted([v for v in vecinos if not visitado[v]])

if no_visitado:
proximo = no_visitado[0]
visitado[proximo] = True
stack.append(proximo)
resultado.append(proximo)

else:
stack.pop()

return resultado

def GrafoCompleto(self):
for key, datos in self.grafo.items():
print (key,datos)
input("Presione enter para continuar")

if __name__ == "__main__":
g = Grafo()
g.seleccion()
```