```
'''
Created on 08/12/2013

@author: Flor
'''
from pydoc import deque

class Vertice(object):
    def __init__(self,dato):
        self.vecinos = dict() #clave id / peso
        self.dato = dato

    # Devuelve los vecinos del vertice
    def getVecinos(self):
        return self.vecinos

    # Agrega un vecino al diccionario de vecinos
    def agregarVecino(self,idVertice, peso):
        self.vecinos[idVertice] = peso

    # Borra un vecino del diccionario de vecinos
    def borrarVecino(self, idVertice):
        del(self.vecinos[idVertice])

    def getPesoArista(self, idVertice):
        return self.vecinos[idVertice]

    # Devuelve el dato que guarda el vertice
    def getDato(self):
        return self.dato



class Grafo(object):
    def __init__(self):
        self.vertices = dict()
        self.cantidadVertices = 0

    # Agrega un nuevo vertice al diccionario de vertices
    def agregarVertice(self,dato):
        self.cantidadVertices = self.cantidadVertices + 1
        self.vertices[dato.getId()] = Vertice(dato)

    ################################################

    # Algoritmo Dijkstra para camino minimo
    def getCaminoOptimo(self, vInicial, vFinal):
        distancias = dict()
        padres = dict()
        cola = deque()
        # marco todas las distancias en infinito y los padres en None
        for v in self.vertices.keys():
            distancias[v] = 999999999
            padres[v] = None
        distancias[vInicial] = 0 # la distancia a si mismo es cero
        cola.append((vInicial, distancias[vInicial]))
        while cola:
            u = min(cola, key = lambda x: x[1])
            cola.remove(u)
            for x in self.vertices[str(u[0])].getVecinos(): # para x en los vecinos
                if (distancias[x] > distancias[u[0]] + self.vertices[x].getPesoArista(str(u[0]))):
                    distancias[x] = distancias[u[0]] + self.vertices[x].getPesoArista(str(u[0]))
                    padres[x] = u
                    cola.append((x, distancias[x]))
        padre = padres[vFinal]
        res = list()
        res.append(vFinal)
        ###
        if padre == None:
            return distancias[vFinal], res
        ###
        while padre[0] != vInicial:
```

```
            res.append(padre[0])
            padre = padres[padre[0]]
        if not res.__contains__(vInicial):
            res.append(vInicial)
        res.reverse()
        return distancias[vFinal], res

    ##################################################

    # Borra un vertice del diccionario
    def borrarVertice(self,v):
        del(self.vertices[v.nombre])
        self.cantidadVertices = self.cantidadVertices - 1


    # Agrega una nueva arista que conecte dos vertices
    def agregarArista(self,idVertice1,idVertice2, peso):
        self.vertices[idVertice1].agregarVecino(idVertice2,peso)
        self.vertices[idVertice2].agregarVecino(idVertice1,peso)

    # Toma un vertice del grafo
    def obtenerVertice(self,v1):
        return self.vertices[v1]

    # Borra una arista del grafo
    def borrarArista(self,v1,v2):
        self.vertices[v1.nombre].borrarVecino(v2)
        self.vertices[v2.nombre].borrarVecino(v1)
```