def print\_path(self, P, d):  
 if P[d] != -1: # if parent exists  
 self.print\_path(P, P[d])  
 print(d, end=" ")  
  
def matrix\_multiplication(self, D, A, P):  
 nrV = self.get\_number\_vertices()  
 for i in range(nrV):  
 for j in range(nrV):  
 for k in range(nrV):  
 # if we found a vertex between i and j and the 2 edges summed have a lowe cost than the edge (i,j)  
 # save it. Also save the vertex between i and j in the parent matrix  
 if D[i][k] + A[k][j] < D[i][j]:  
 D[i][j] = D[i][k] + A[k][j]  
 P[i][j] = k  
  
 # return matrix of cost  
 return D  
  
def find\_cost\_matrix(self):  
 nrV = self.get\_number\_vertices()  
 A = [[math.inf for i in range(nrV)] for j in range(nrV)] # matrix  
 P = [[-1 for i in range(nrV)] for j in range(nrV)] # parent matrix  
 steps = [] # save steps  
  
 # the matrix has initially 0 diagonally, the cost where is an edge, or infinite if there is not an edge  
 for i in range(nrV):  
 for j in range(nrV):  
 if i == j:  
 A[i][j] = 0  
 elif self.is\_edge(i, j):  
 A[i][j] = self.\_dictCost[(i, j)]  
 P[i][j] = i  
 D = deepcopy(A)  
 steps.append(deepcopy(D))  
  
 # start the algorithm  
 for m in range(2, nrV):  
 D = self.matrix\_multiplication(D, A, P)  
 steps.append(deepcopy(D))  
  
 # return matrix of costs, parent matrix and the steps  
 return D, P, steps