Program

def build\_matrix(self):  
 *"""  
 Builds the graph in the form of a matrix, and also builds it's path matrix* ***:return****:  
 """* self.\_matrix = {}  
 self.\_path = {}  
 for v in self.parseX():  
 self.\_matrix[v] = {}  
 self.\_path[v] = {}  
 for w in self.parseX():  
 if v == w:  
 self.\_matrix[v][w] = 0  
 self.\_path[v][w] = 0  
 elif (v, w) not in self.\_edge\_dict:  
 self.\_matrix[v][w] = -1  
 self.\_path[v][w] = 0  
 else:  
 self.\_matrix[v][w] = self.\_edge\_dict[(v, w)]  
 self.\_path[v][w] = v  
  
def print\_matrices(self):  
 *"""  
 Prints the walk matrix and the path matrix in their current states* ***:return****:   
 """* line = ' '  
 for v in self.\_matrix:  
 line += str(v).rjust(3) + ' '  
 line += ' ' + line  
 print(line)  
 print(' ' + ''.join(['-' for i in range(len(line)-5)]))  
 for v in self.\_matrix:  
 line = str(v).rjust(3) + '| '  
 for w in self.\_matrix:  
 line += str(self.\_matrix[v][w]).rjust(3) + ' '  
 line += ' | '  
 for w in self.\_matrix:  
 line += str(self.\_path[v][w]).rjust(3) + ' '  
 print(line + '|' + str(v))  
 print()  
  
def Floyd\_Warshall(self, x, y):  
 *"""  
 Computes the walk matrix and the path matrix using the Floyd-Warshall algorithm and returns the minimum cost  
 walk between the two given vertices* ***:param*** *x: (int) the source vertex* ***:param*** *y: (int) the destination vertex* ***:return****: (list of int) the minimum cost walk between the two vertices if there is a walk between them or  
 None if there is no walk between them  
 """* self.build\_matrix()  
  
 self.print\_matrices()  
  
 for k in self.\_matrix:  
 for i in self.\_matrix:  
 for j in self.\_matrix[i]:  
 if j == k or i == k or self.\_matrix[i][k] == -1 or self.\_matrix[k][j] == -1:  
 continue  
 if self.\_matrix[i][j] > self.\_matrix[i][k] + self.\_matrix[k][j] or self.\_matrix[i][j] == -1:  
 self.\_matrix[i][j] = self.\_matrix[i][k] + self.\_matrix[k][j]  
 self.\_path[i][j] = self.\_path[k][j]  
 self.print\_matrices()  
   
 if self.\_path[x][y] == 0:  
 return None  
 ver = self.parseX()  
 path = [0 for i in range(len(ver))]  
  
 k = len(ver) - 1  
 path[k] = y  
  
 while path[k] != x and k > 0:  
 path[k - 1] = self.\_path[x][path[k]]  
 k -= 1  
  
 return path[k:], self.\_matrix[x][y]