Warshall Algorithm Implementation:

def warshall(a):

n = len(a)

dis\_matrix = a

for k in range(n):

ndis\_matrix = [list(row) for row in dis\_matrix]

for i in range(n):

for j in range(n):

ndis\_matrix[i][j] = min(dis\_matrix[i][j], dis\_matrix[i][k] + dis\_matrix[k][j])

dis\_matrix = ndis\_matrix

return dis\_matrix

matrix = [

[0, 0, 0, 1],

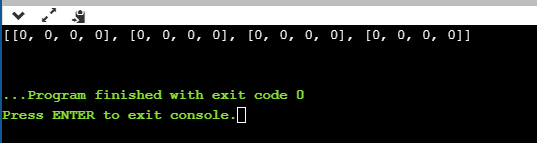
[1, 0, 1, 0],

[1, 0, 0, 1],

[0, 0, 1, 0]

]

print(warshall(matrix))



Floyds Transitive Closure:

class Floyd:

def \_\_warshall\_\_(s, n):

s.V = n

def printSolution(self, reach):

print ("Following matrix transitive closure of the given matrix ")

for i in range(s.V):

for j in range(s.V):

print ([i][j]),

def transitiveClosure(self,matrix):

a =[i[:] for i in matrix]

for k in range(self.V):

for i in range(self.V):

for j in range(self.V):

a[i][j] = a[i][j] or (a[i][k] and a[k][j])

self.printSolution(a)

m= Floyd(4)

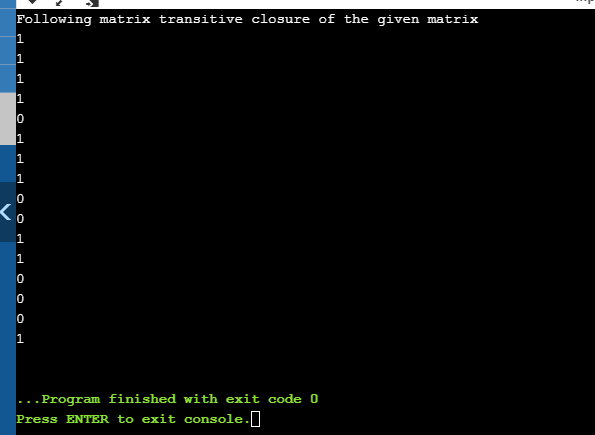
matrix = [[1, 1, 0, 1],

[0, 1, 1, 0],

[0, 0, 1, 1],

[0, 0, 0, 1]]

m.transitiveClosure(matrix)



Warshall:

def warshall(a):

assert (len(row) == len(a) for row in a)

n = len(a)

for k in range (1,n):

for i in range (1,n):

for j in range (1,n):

a[i][j] = a[i][j] or (a[i][k] and a[k][j])

return a

warshall ([[0,0,0,1],[1,0,1,0],[1,0,0,1],[0,0,1,0]])

print(warshall)

