3. ITERATIVE DEEPENING SEARCH

from collections import defaultdict

class Graph:

def \_\_init\_\_(self,vertices):

# No. of vertices

self.V = vertices

# default dictionary to store graph

self.graph = defaultdict(list)

# function to add an edge to graph

def addEdge(self,u,v):

self.graph[u].append(v)

# A function to perform a Depth-Limited search

# from given source 'src'

def DLS(self,src,target,maxDepth):

if src == target : return True

# If reached the maximum depth, stop recursing.

if maxDepth <= 0 : return False

# Recur for all the vertices adjacent to this vertex

for i in self.graph[src]:

if(self.DLS(i,target,maxDepth-1)):

return True

return False

# IDDFS to search if target is reachable from v.

# It uses recursive DLS()

def IDDFS(self,src, target, maxDepth):

# Repeatedly depth-limit search till the

# maximum depth

for i in range(maxDepth):

if (self.DLS(src, target, i)):

return True

return False

# Create a graph given in the above diagram

g = Graph (7);

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 3)

g.addEdge(1, 4)

g.addEdge(2, 5)

g.addEdge(2, 6)

target = 6; maxDepth = 3; src = 0

if g.IDDFS(src, target, maxDepth) == True:

print ("Target is reachable from source " +

"within max depth")

else :

print ("Target is NOT reachable from source " +

"within max depth")



