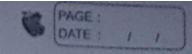
_	Source Code
_	Source Code
	DES code
	From collections import defaultdict
	# This collections mapped detaultand
_	# This collection represents a directed graph using adjace
	- ncy list representation
	class amah:
	class graph: def_init (self):
	det_Init (sur)
	self. graph = defaultdict (list)
	def addedge (self, u, v): self graph (u) append [v]
	Self graph (U) append [V]
	def DESUtil (self, v. Visited):
	vieited. add (v)
	print (v, end = '')
	# Rescue For all vertices adjacent to this vertex
	for neighbour in self. graph [v]:
	if neighbour not in visited:
	self. DES util (neighbour, visited)
-	ZELI DIZ OTIC (HEIGHOOD) AIRCEA)

```
# The function to do DFS transversal using recursive DFS
       def DFS (self, v):
           visited = set ()
self. Desutil (v. visited)
# Driver one
# Create a graph in given above
       9= graph()
       9. addedge Co. 1
       q. addedge (0,2)
       9. addedge (1,2)
       g. addedge (2,0)
       g. addedge (2, 3)
 print (" The Following is in DFS For From vertex 2")
  9. DFS (2)
BFS code
 From collections impost defaultdict
 class graph:
   # constructor
       def -- int -- (selt):
           self. graph = defaultdict (list)
       det addédge (self, u, v):
           self. graph [u]. append [v]
       def BFS (self, s)
    # Mark all vertices as non visited
           visited = [False] * (max (self, graph)+1)
     # create a queue for BFG
         queve = []
```

DATE: / /

-	
1	
	# Mark source node as visited 4 enqueue
	avoue arrent (e)
	queve append (s) visited (s) = True
-	Titled C2 ] - 1006
	Obile and i
_	while queue i
_	s = queve. pop(o)  print(s, end=')
	pmnt (s, cha =
_	1
	for i in self. graph [s]:  if visited [i] == false:
	It visited [i] = = talle:
_	g= graph() g. addedge (0,1)
	g. addodge (0,1)
	g. addedge (0,2)
	a. addedge (1,2)
	a. addedge (2,0)
	2 41 4 (213)
	point (" following is Breadth First starting trois vertex
	2")
	9.BFS(2)
	9.0136
	Doinl.
•	DFID code
	- " 1 1 1 D 1111L
	From collections import defaultdict
	class Igraph:
	def_init_ (self, vertices)!
	solF v = vertices
	mit amab = defaultdict (list)
	self. graph = default dict (list)  def add edge (self, u, v):
	get addrage (selt, U, V).
	self. graph [v]. append [v]



det DIS (self, src, target, max Depth):
if src = target
return true

if max Depth <= 0

for 1 in self. graph [src]:

if (self. DIS (i, target, max depth - 1)):

return true

return False

def IDDES (self, sxc, target, maxdepth)

For i in range (max depth):

if (self, DFS (sxc, target, 1)):

return true

return false

def DFSutil (Self, v, visited): visited add (v)

print (v. end='')

for neighbour in self. graph (v]:

if neighbour not in visited:

self. DESutil (neighbour, visited)

def RES (solf, s):

vieited = [False] \* max (self)

queue = []
queue append (s)
visited [s] = true
print ()

while queve:
S = queve. pop(o)
print (s, end = '):
for in self-graph [s]:
if visited [i] == False!
queve. append [i] visited [i] = True
visited Cit = True
g= 9 raph [7]:
g. addedge (O, 1)
g. addedge (0,2)
g. addedge (0,3)
g. addedge (1,4)
g. addedge (2,5)
g. waterige
11 - 6
target = 6  max_depth = 3  sxc = 0
max_depth - 3
Sxc = 0
IF I CO T I ! - bille From courses" +
print (" Target is reachable from source"+ " within max_Depth")
Print ("Target is not reachable from source" + within max_ Depth")
print ("Target is not reachable from source" +
within max_ Depth")
9. DES (0) 9. BES (0)
O. REC(O)

DATE :