- Source Code

DFS code

```
From collections import defaultdict
# This collection represents a directed graph using adjace
-ncy list representation

class graph:
    def __init(self):
        self.graph = defaultdict(list)
    def addedge(self, u, v):
        self.graph[u].append[v]
    def DFSutil(self, v, visited):
        visited.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.DFSutil(neighbour, visited)
```

```
# The function to do DFS traversal using recursive DFS
    def DFS(self,v):
        visited = set()
        self.DFSUtil(v, visited)
# Driver code
# Create a graph in given above
    g= graph()
    g.addedge(0,1)
    g.addedge(0,2)
    g.addedge(1,2)
    g.addedge(2,0)
    g.addedge(2,3)
    g.addedge(3,3)
print("The Following is in DFS for From vertex 2")
g.DFS(2)
```

- **BFS code**

```
From collections import defaultdict
class graph:
  # constructor
    def __int__(self):
        self.graph = defaultdict(list)
    def addEdge(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 BFS
        queue = []
```

```
# Mark source node as visited & enqueue
    queue.append(s)
    visited[s] = True

while queue:
    s = queue.pop(0)
    print(s, end=' ')

for i in self.graph[s]:
    if visited[i] == false:

g= graph()
g.addedge(0,1)
g.addedge(0,2)
g.addedge(1,2)
g.addedge(2,0)
g.addedge(2,3)
print("following is Breadth First starting from vertex
  2")
g.BFS(2)
```

- DFID code

```
from collections import defaultdict

class lgraph:
    def __init__(self, vertices):
        self.v = vertices
        self.graph=defaultdict(list)
    def addedge(self, u, v):
        self.graph[u].append[v]
```

```
def DLS(self, src, target, maxDepth):
    if src == target
        return true

    if maxDepth <= 0
        return false

    for i in self.graph[src]:
        if (self.DLS(i, target, maxdepth-1)):
            return true
        return false

    def IDDFS(self, src, target, maxdepth)
        for i in range(max depth):
            if (self.BFS(src.target, i)):
                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.DFSutil(neighbour, visited)
    def BFS(self, s):
        visited = [false] * max(self)
    queue = []
    queue.append(s)
    visited[s] = true
    print()
```

DATE : / /

```
while queue:
    s = queue.pop(0)
    print(s, end = ' '):
    for in self.graph[s]:
        if visited[i] == False:
            queue.append[i]
            visited[i] = True
g = graph[7]:
g.addedge(0,1)
g.addedge(0,2)
g.addedge(0,3)
g.addedge(1,4)
g.addedge(2,5)

target = 6
max_depth = 3
src = 0
if
print("Target is reachable from source" +
            "within max_Depth")

else:
    Print("Target is not reachable from source" +
            "within max_Depth")

g.DFS(0)
g.BFS(0)
```