In [2]:

```
class ArrayStack:
    def __init__(self,size):
        self.size=size
        self.data=[0 for i in range(size)]
        self.top=0
    def isEmpty(self):
        if self.top==0:
            return True
        else:
            return False
    def Push(self,value):
        if self.top==self.size:
            print("Stack Overflow !")
        else:
            self.data[self.top]=value
            self.top+=1
    def Pop(self):
        if self.isEmpty():
            print("Stack Underflow !")
        else:
            self.top-=1
            x=self.data[self.top]
            self.data[self.top]=0
            return x
class Node:
    def __init__(self,value):
        self.value=value
        self.next=None
class ListQueue:
    def __init__(self,size):
        self.size=size
        self.head=None
        self.tail=None
    def enQueue(self,value):
        if self.Count()==self.size:
            pass
        else:
            newnode=Node(value)
            x=self.tail
            if self.head == None:
                self.head=newnode
                self.tail=newnode
            else:
                self.tail.next=newnode
                self.tail=newnode
    def deQueue(self):
        if self.head==None:
            pass
        else:
            x=self.head
            a=x.value
            x=x.next
            self.head=x
            return a
    def isEmpty(self):
        if self.Count==0:
            return True
        else:
            return False
    def Count(self):
        x=self.head
        count=0
        while x:
            count+=1
            x=x.next
        return count
class Graph:
    def __init__(self,vertex):
        self.vertex=vertex
        self.adj=[[0 for i in range(vertex)]for j in range(vertex)]
        self.visited=[]
    def AddEdge(self,src,dest):
        if src==dest:
            print("Source and destination are same.")
        else:
            self.adj[src][dest]=1
            self.adj[dest][src]=1
    def PrintMatrix(self):
        for i in range(self.vertex):
            for j in range(self.vertex):
                print(self.adj[i][j],end=" ")
            print("\r")
    def GetNeighbours(self,vertex):
        lst=[]
        for i in range(self.vertex):
```

```
            if self.adj[vertex][i] == 1:
                lst.append(i)
                #print("Position :",i,"to",vertex,"and",vertex,"to",i)
        return lst
    def DFS(self,source):
        s=ArrayStack(self.vertex)
        s.Push(source)
        self.visited.append(source)
        while not s.isEmpty():
            x=s.Pop()
            print("Visited {}".format(x))
            neighbours=self.GetNeighbours(x)
            for i in neighbours:
                if i not in self.visited:
                    s.Push(i)
                    self.visited.append(i)
    def BFS(self,source):
        self.visited=[]
        s=ListQueue(self.vertex)
        s.enQueue(source)
        self.visited.append(source)
        while not s.isEmpty():
            x=s.deQueue()
            if x!=None:
                print("Visited {}".format(x))
                neighbours=self.GetNeighbours(x)
                for i in neighbours:
                    if i not in self.visited:
                        s.enQueue(i)
                        self.visited.append(i)
            else:
                break
a=Graph(4)
a.AddEdge(1,2)
a.AddEdge(1,0)
a.AddEdge(0,2)
a.AddEdge(2,3)
a.PrintMatrix()
vertex=2
for i in a.GetNeighbours(vertex):
    print("Position :",i,"to",vertex,"and",vertex,"to",i)
print("----------DFS---------")
a.DFS(2)
print("----------BFS---------")
a.BFS(2)
```

```
0 1 1 0
1 0 1 0
1 1 0 1
0 0 1 0
Position : 0 to 2 and 2 to 0
Position : 1 to 2 and 2 to 1
Position : 3 to 2 and 2 to 3
----------DFS---------
Visited 2
Visited 3
Visited 1
Visited 0
----------BFS---------
Visited 2
Visited 0
Visited 1
Visited 3
```