# BFS Algorithm

# Steps

# Step 1 : All nodes - color - White, distance - Infinite(Zero), Parent - Null

# Step 2 : Starting point - color - Gray, distance - Zero

# Step 3 : Initialize Queue and enqueue the starting point

# Step 4 : While Queue is not empty loop

# Step 5 : Inside loop dequeue(u)

# Step 6 : Loop through all adjacent nodes(i) which are unvisited

# Step 7 : Set parent - u, distance dist(u)+1 color - Gray

# Step 8 : Enqueue i

# Step 9 : Set color(u) = Black

class Queue:

    def \_\_init\_\_(self) -> None:

        self.lst = []

    def enqueue(self,element) :

        self.lst.append(element)

    def dequeue(self) :

        return self.lst.pop(0)

    def isempty(self) :

        return self.lst == []

def BFS(graph,starting\_point) :

    n = len(graph)

    color\_array = ['White']\*n

    dist\_array = [0]\*n

    parent\_array = [None]\*n

    color\_array[starting\_point] = 'Gray'

    Q = Queue()

    Q.enqueue(starting\_point)

    while Q.isempty() == False :

        u = Q.dequeue()

        print(u)

        for i in range(n) :

            if graph[u][i] == 1 :

                if color\_array[i] == 'White' :

                    parent\_array[i] = u

                    dist\_array[i] = dist\_array[u]+1

                    color\_array[i] = 'Gray'

                    Q.enqueue(i)

        color\_array[u] = 'Black'

    print('-'\*40)

    print(parent\_array)

    print(dist\_array)

graph = [

    [0,1,1,0,0],

    [1,0,1,1,0],

    [1,1,0,0,0],

    [0,1,0,0,1],

    [0,0,1,1,0]

]

BFS(graph,0)

## Bipartite graph

class Queue :

    def \_\_init\_\_(self) -> None:

        self.lst = []

    def enqueue(self,element) :

        self.lst.append(element)

    def dequeue(self) :

        return self.lst.pop(0)

    def isempty(self) :

        return self.lst == []

def bipartite(graph) :

    n = len(graph)

    color\_array = ['White']\*n

    color\_array\_2 = [None]\*n

    color\_array\_2[0] = 'Blue'

    color\_array[0] = 'Gray'

    Q = Queue()

    Q.enqueue(0)

    while Q.isempty() == False :

        u = Q.dequeue()

        for i in range(n) :

            if graph[u][i] == 1 :

                if color\_array[i] == 'White' :

                    if color\_array\_2[i] == None :

                        if color\_array\_2[u] == 'Blue' :

                            color\_array\_2[i] = 'Red'

                        else :

                            color\_array\_2[i] = 'Blue'

                    else :

                        if color\_array\_2[i] == color\_array\_2[u] :

                            return False

                    color\_array[i] = 'Gray'

                    Q.enqueue(i)

                else :

                    if color\_array\_2[i] == None :

                        if color\_array\_2[u] == 'Blue' :

                            color\_array\_2[i] = 'Red'

                        else :

                            color\_array\_2[i] = 'Blue'

                    else :

                        if color\_array\_2[i] == color\_array\_2[u] :

                            return False

        color\_array[u] = 'Black'

    return True

graph = [

    [0,1,1],

    [1,0,1],

    [1,1,0]

]

graph2 = [

    [0,1,0,1],

    [1,0,1,0],

    [0,1,0,1],

    [1,0,1,0]

]

print(bipartite(graph2))

# DFS

# Steps

# DFS : Parent - set to null Color - White, time-0

# For all unvisted node call DFS\_visit

# DFS\_visit : Increment time set starting time to time

# Set color of node to gray

# For all adjacent unvisited nodes call DFS\_visit and before that set parent - u

# Increment time

# Set finishing time - time and change color to black

def DFS(graph) :

    parent\_array = [None]\*len(graph)

    color\_array = ['White']\*len(graph)

    starting\_time = [None]\*len(graph)

    finishing\_time = [None]\*len(graph)

    time = [0]

    for i in range(len(graph)) :

        if color\_array[i] == 'White' :

            DFS\_visit(graph,i,time,color\_array,parent\_array,starting\_time,finishing\_time)

    print('-'\*40)

    print(parent\_array)

    print(starting\_time)

    print(finishing\_time)

def DFS\_visit(graph,u,time,color\_array,parent\_array,starting\_time,finishing\_time) :

    print(u)

    time[0] += 1

    starting\_time[u] = time[0]

    color\_array[u] = 'Gray'

    for i in range(len(graph)) :

        if graph[u][i] == 1 and color\_array[i] == 'White' :

            parent\_array[i] = u

            DFS\_visit(graph,i,time,color\_array,parent\_array,starting\_time,finishing\_time)

    time[0] += 1

    finishing\_time[u] = time[0]

    color\_array[u] = 'Black'

graph = [

    [0,1,1,1,0],

    [1,0,0,0,0],

    [1,0,0,1,1],

    [1,0,1,0,0],

    [0,0,1,0,0]

]

DFS(graph)

## Strongly connected components

def DFS\_1(graph) :

    color\_array = ['White']\*len(graph)

    time = [0]

    initial\_time = [None]\*len(graph)

    final\_time = [None]\*len(graph)

    for i in range(len(graph)) :

        if (color\_array[i] == 'White') :

            DFS\_visit\_1(graph,i,time,color\_array,initial\_time,final\_time)

    return final\_time

def DFS\_visit\_1(graph,u,time,color\_array,initial\_time,final\_time) :

    time[0] += 1

    initial\_time[u] = time[0]

    color\_array[u] = 'Gray'

    for i in range(len(graph)) :

        if (color\_array[i] == 'White') and (graph[u][i] == 1) :

            DFS\_visit\_1(graph,i,time,color\_array,initial\_time,final\_time)

    time[0] += 1

    final\_time[u] = time[0]

    color\_array[u] = 'Black'

graph = [

    [0,0,1,1,0],

    [1,0,0,0,0],

    [0,1,0,0,0],

    [0,0,0,0,1],

    [0,0,0,0,0]

]

final\_times = []

final\_times\_unformatted = DFS\_1(graph)

for i in range(len(final\_times\_unformatted)) :

    final\_times.append([i,final\_times\_unformatted[i]])

final\_times.sort(key=lambda x: x[1],reverse=True)

print(final\_times)

already\_done = []

for i in range(len(graph)) :

    for j in range(len(graph)) :

        if ((i,j) not in already\_done) and ((j,i) not in already\_done) :

            if graph[i][j] == 1 :

                if graph[j][i] == 0 :

                    graph[i][j] = 0

                    graph[j][i] = 1

            elif graph[j][i] == 1 :

                graph[i][j] = 1

                graph[j][i] = 0

            already\_done.append((i,j))

print(graph)

def DFS\_2(graph) :

    color\_array = ['White']\*len(graph)

    res = []

    for i in final\_times :

        if color\_array[i[0]] == 'White' :

            temp = []

            DFS\_visit\_2(graph,i[0],color\_array,temp)

            res.append(temp.copy())

    print(res)

def DFS\_visit\_2(graph,u,color\_array,temp) :

    color\_array[u] = 'Gray'

    # print(color\_array)

    temp.append(u)

    for i in range(len(graph)) :

        if (graph[u][i] == 1) and (color\_array[i] == 'White') :

            DFS\_visit\_2(graph,i,color\_array,temp)

    color\_array[u] = 'Black'

    # input()

DFS\_2(graph)

## Cycle detection undirected

def DFS(graph) :

    color\_array = ['White']\*len(graph)

    parent\_array = [None]\*len(graph)

    for i in range(len(graph)) :

        if color\_array[i] == 'White' :

            if (DFS\_visit(graph,i,color\_array,parent\_array) == True) :

                return True

    # print(parent\_array)

    return False

def DFS\_visit(graph,u,color\_array,parent\_array) :

    # print(u)

    color\_array[u] = 'Gray'

    for i in range(len(graph)) :

        if graph[u][i] == 1 and color\_array[i] == 'White' :

            parent\_array[i] = u

            if (DFS\_visit(graph,i,color\_array,parent\_array) == True) :

                return True

        elif graph[u][i] == 1 and parent\_array[u] != i :

            return True

    color\_array[u] = 'Black'

graph = [

    [0,1,0,0],

    [1,0,1,1],

    [0,1,0,1],

    [0,1,1,0]

]

graph2 = [

    [0,1,0],

    [1,0,1],

    [0,1,0]

]

print(DFS(graph))

## Cycle detection directed

def DFS(graph) :

    parent\_array = [None]\*len(graph)

    color\_array = ['White']\*len(graph)

    for i in range(len(graph)) :

        if color\_array[i] == 'White' :

            path\_visited = [0]\*len(graph)

            if DFS\_visit(graph,i,path\_visited,color\_array,parent\_array) == True :

                return True

    return False

def DFS\_visit(graph,u,path\_visited,color\_array,parent\_array) :

    color\_array[u] = 'Gray'

    path\_visited[u] = 1

    for i in range(len(graph)) :

        if graph[u][i] == 1 and color\_array[i] == 'White' :

            parent\_array[i] = u

            if (DFS\_visit(graph,i,path\_visited.copy(),color\_array,parent\_array) == True) :

                return True

        elif graph[u][i] == 1 and path\_visited[i] == 1 and parent\_array[u] != i :

            return True

    color\_array[u] = 'Black'

graph = [

    [0,1,0,1],

    [0,0,1,0],

    [0,0,0,0],

    [0,0,1,0]

]

graph2 = [

    [0,1,0,0],

    [0,0,1,0],

    [0,0,0,1],

    [1,0,0,0]

]

print(DFS(graph2))