Artificial Intelligence  
Lab Exercise 4  
Analysis of DFS and BFS

short line

Ashwin Prakash  
RA1911026010048

**DFS:**

Depth-first search (DFS) is an algorithm for searching a graph or tree data structure. The algorithm starts at the root (top) node of a tree and goes as far as it can down a given branch (path), then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored.

**Aim:**

To implement and analyse the DFS algorithm on graphs.

**Algorithm:**

Create a recursive function that takes the index of the node and a visited array.

1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

**Program:**

from collections import defaultdict

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=' ')

        for neighbour in self.graph[v]:

            if neighbour not in visited:

                self.DFSUtil(neighbour, visited)

    def DFS(self, v):

        visited = set()

        self.DFSUtil(v, visited)

if \_\_name\_\_ == '\_\_main\_\_':

    g = Graph()

    range1=int(input('Enter the total number of traversals: '))

    print("Enter node connections")

    for i in range(range1):

        arr=list(map(int,input().strip().split()))

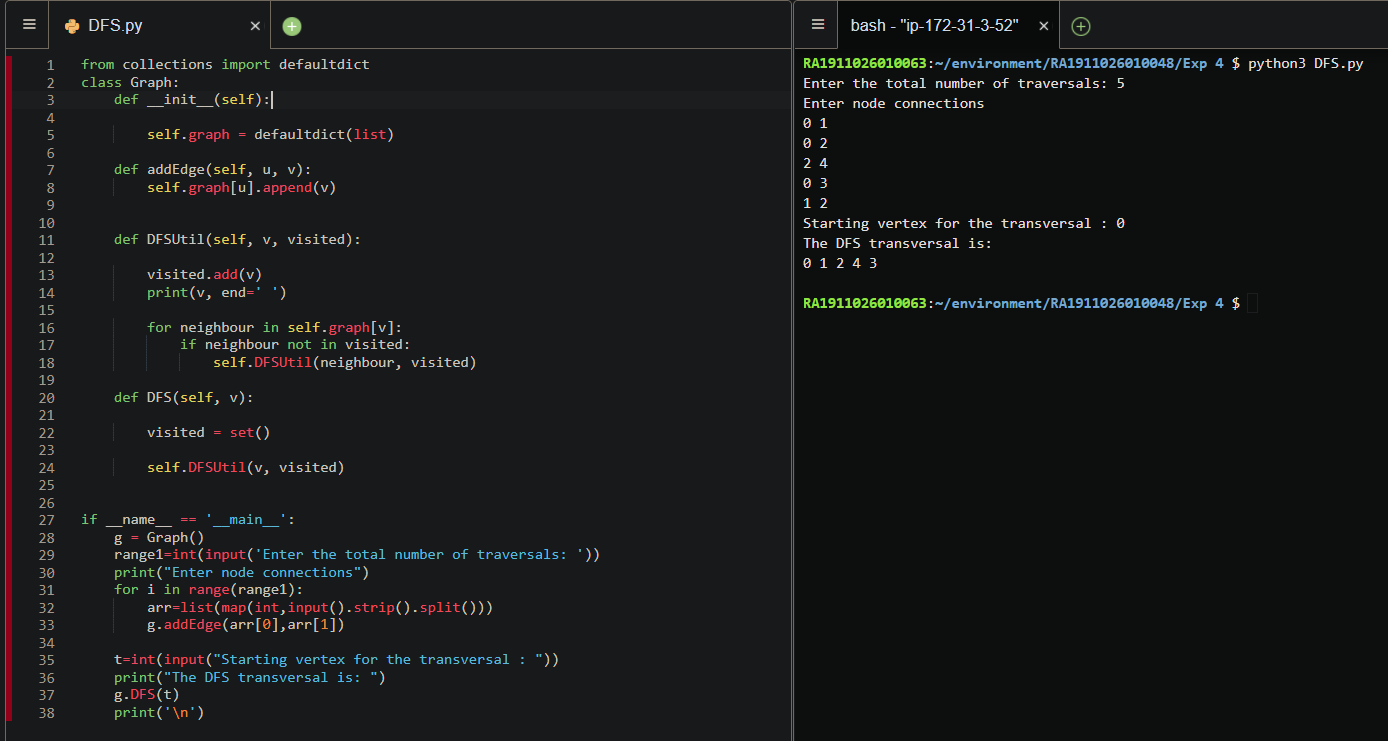
        g.addEdge(arr[0],arr[1])

    t=int(input("Starting vertex for the transversal : "))

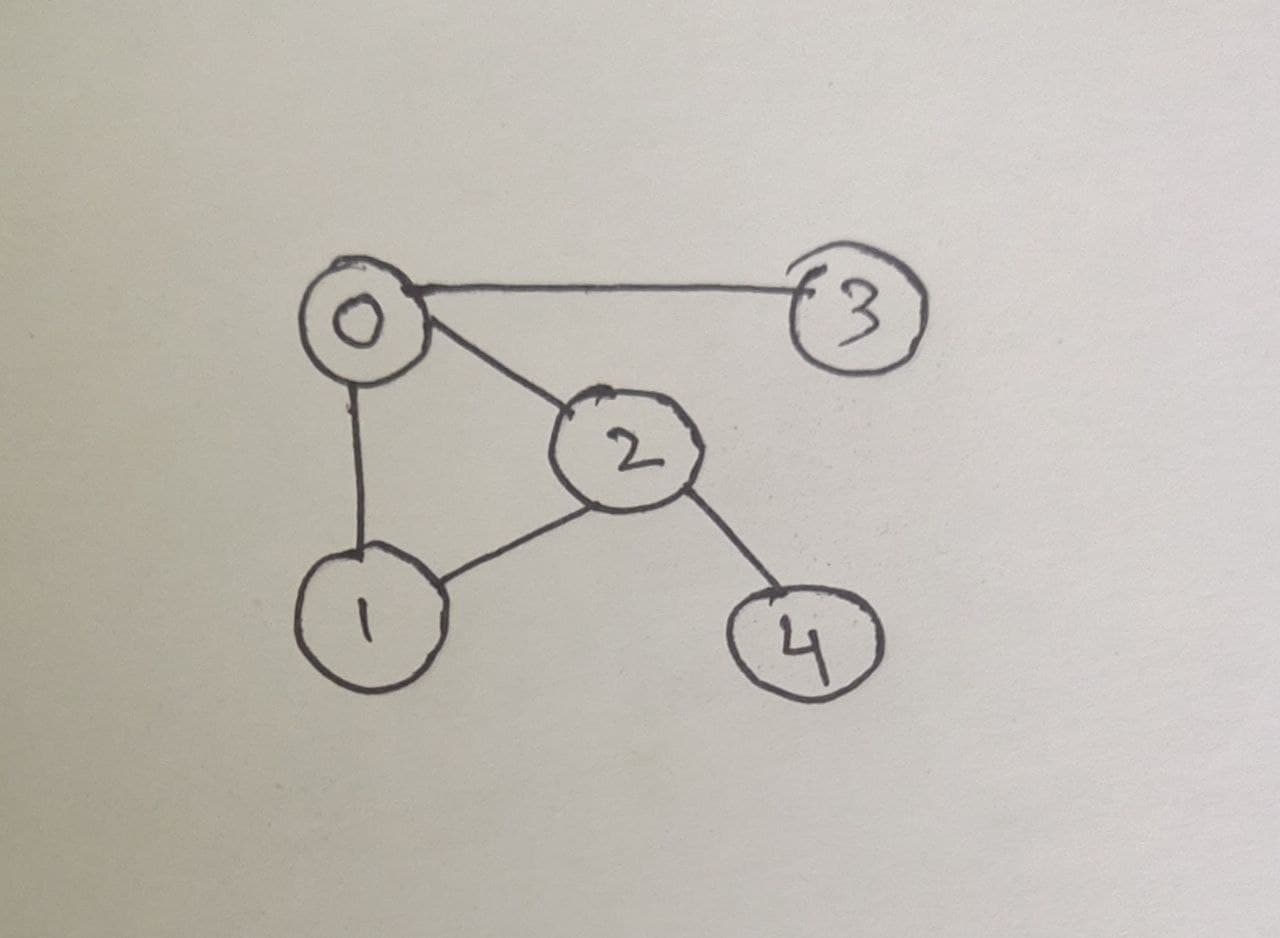
    print("The DFS transversal is: ")

    g.DFS(t)

**Output:**

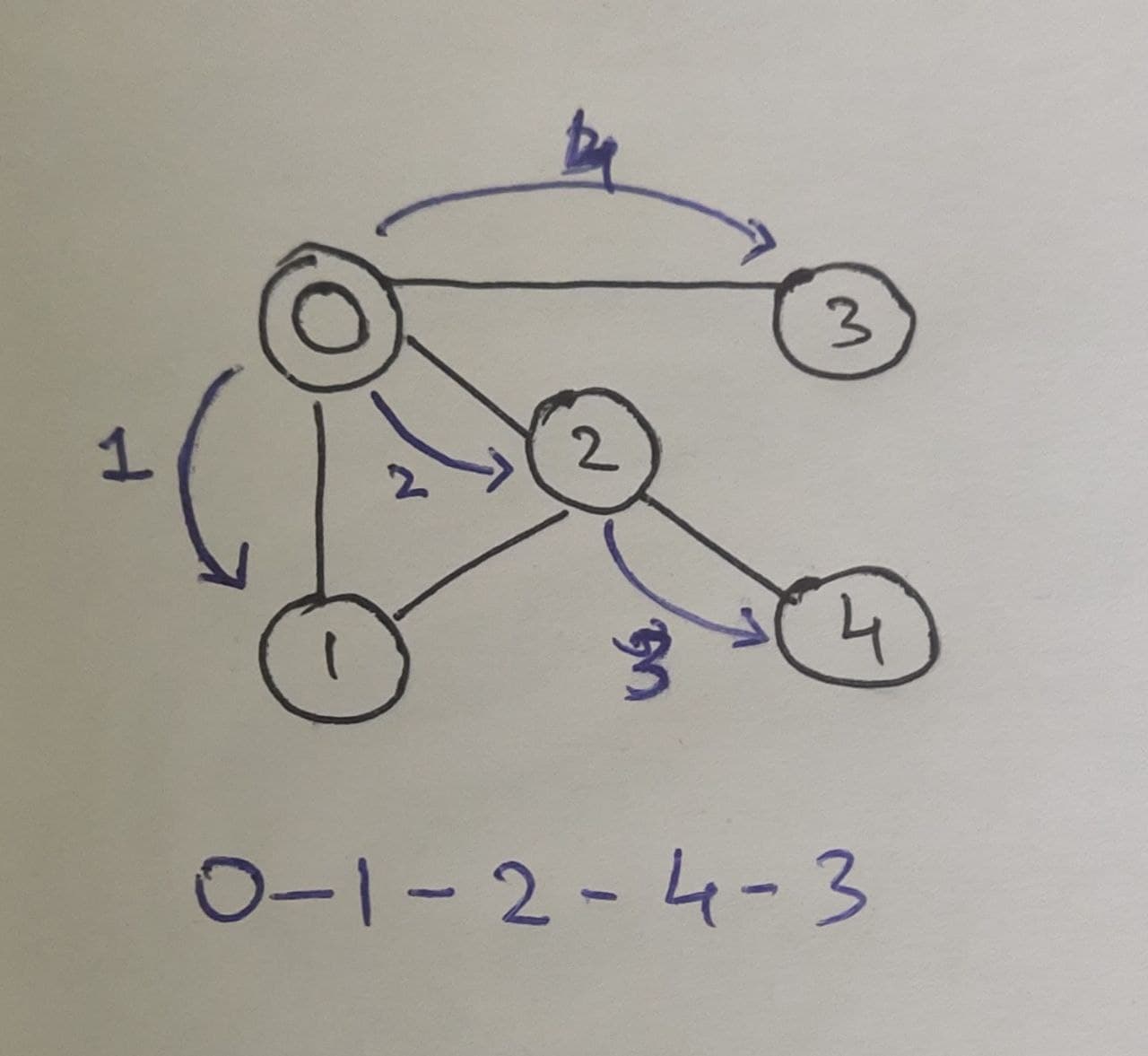


**Observations:**

****

Here, we start traversing the undirected graph from 0 and traverse all the adjacent nodes i.e. (1,2,3) and each of them is added to the stack. Now we pop the element on the top of the stack, add it to the traversal array and search for adjacent nodes. If there are no adjacent nodes to be visited, we go for the next top value of stack and repeat the process. Hence, we obtain the DFS

DFS transversal: 0 1 2 4 3



**Applications of Depth First Search:**

1. Detecting cycle in a graph
2. Path Finding
3. Topological Sorting
4. To test if a graph is bipartite
5. Finding Strongly Connected Components of a graph

**Application:**

**To check if the graph is bi-parted**

**Program:**

V = 4

def colorGraph(G, color, pos, c):

    if color[pos] != -1 and color[pos] != c:

        return False

    color[pos] = c

    ans = True

    for i in range(0, V):

        if G[pos][i]:

            if color[i] == -1:

                ans &= colorGraph(G, color, i, 1-c)

            if color[i] !=-1 and color[i] != 1-c:

                return False

        if not ans:

            return False

    return True

def isBipartite(G):

    color = [-1] \* V

    pos = 0

    return colorGraph(G, color, pos, 1)

if \_\_name\_\_ == "\_\_main\_\_":

    G = [[0, 1, 0, 1],

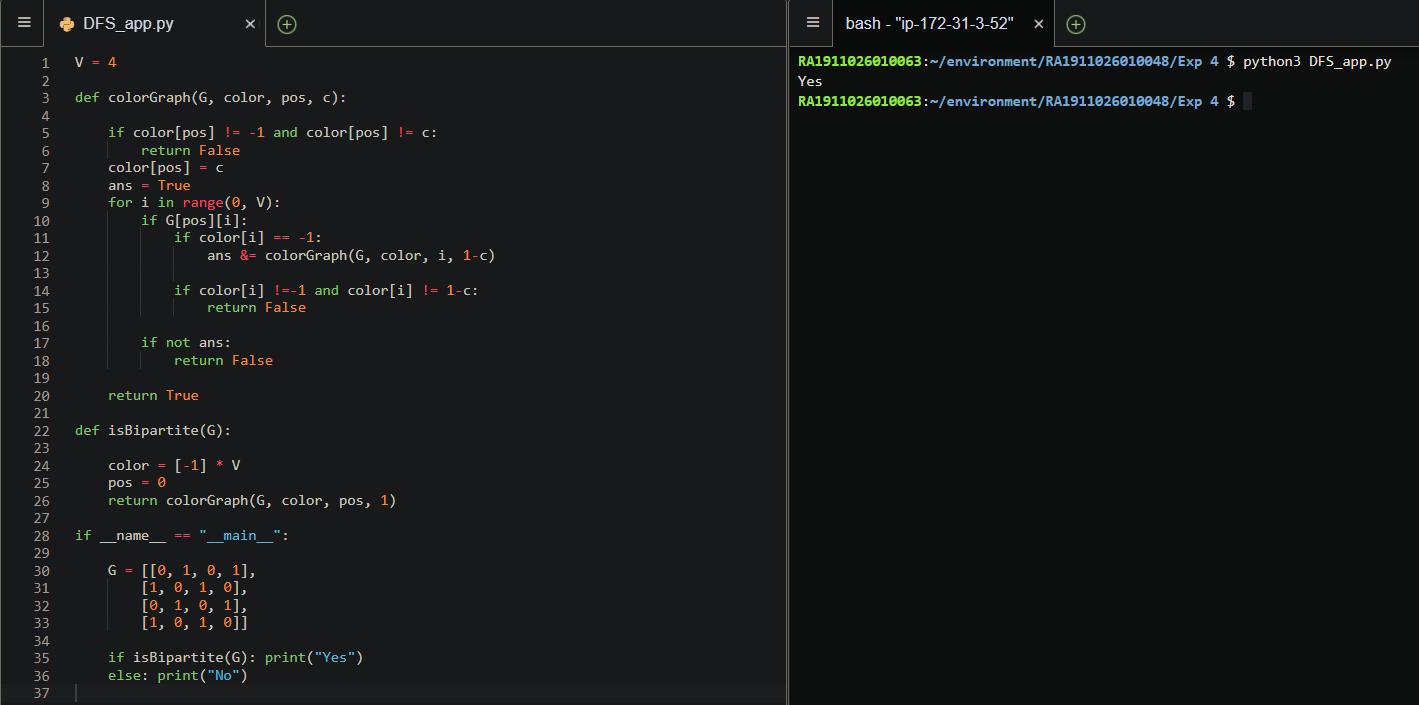
        [1, 0, 1, 0],

        [0, 1, 0, 1],

        [1, 0, 1, 0]]

    if isBipartite(G): print("Yes")

    else: print("No")

****

**Result:**

We have implemented a DFS algorithm for graphs and analysed it.

**Breadth-First Search:**

Breadth-first search (BFS) is an algorithm for searching a tree data structure for a node that satisfies a given property. It starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level

**Aim:**

To implement and analyse the BFS algorithm.

**Algorithm:**

For each node, first, the node is visited and then its child nodes are put in a FIFO queue.

1. Create an empty queue q
2. Enqueue the adjacent neighbors of starting vertex
3. Dequeue a node, print the value and enqueue its adjacent neighbors
4. Repeat Step 3 until there is no node left to be traversed in graph

**Program:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def addEdge(self,u,v):

        self.graph[u].append(v)

    def BFS(self, s):

        visited = set()

        queue = []

        queue.append(s)

        while queue:

            v = queue.pop(0)

            print (v, end = " ")

            for neighbour in self.graph[v]:

                if neighbour not in visited:

                    visited.add(neighbour)

                    queue.append(neighbour)

if \_\_name\_\_ == '\_\_main\_\_':

    g = Graph()

    range1=int(input('Enter the total number of traversals: '))

    print("Enter node connections")

    for i in range(range1):

        arr=list(map(int,input().strip().split()))

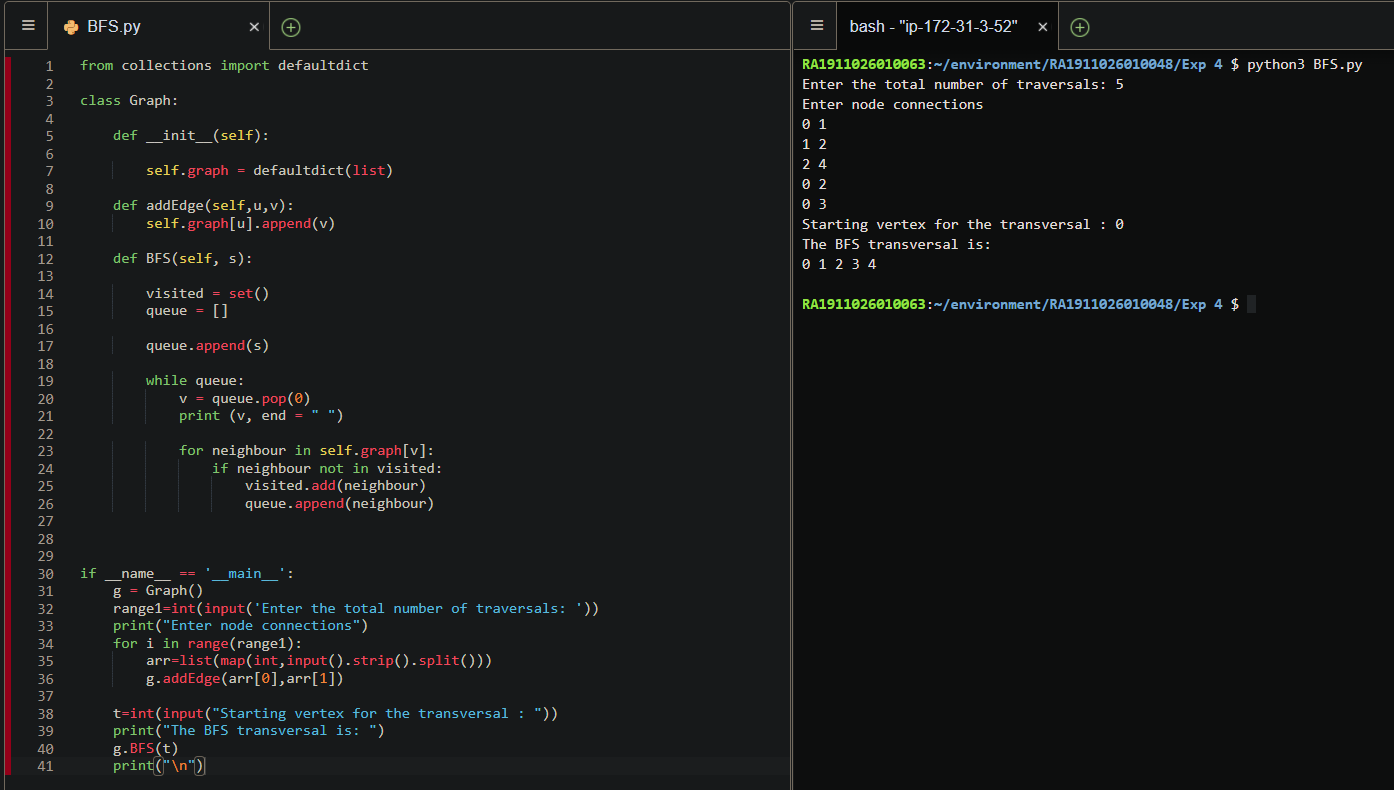
        g.addEdge(arr[0],arr[1])

    t=int(input("Starting vertex for the transversal : "))

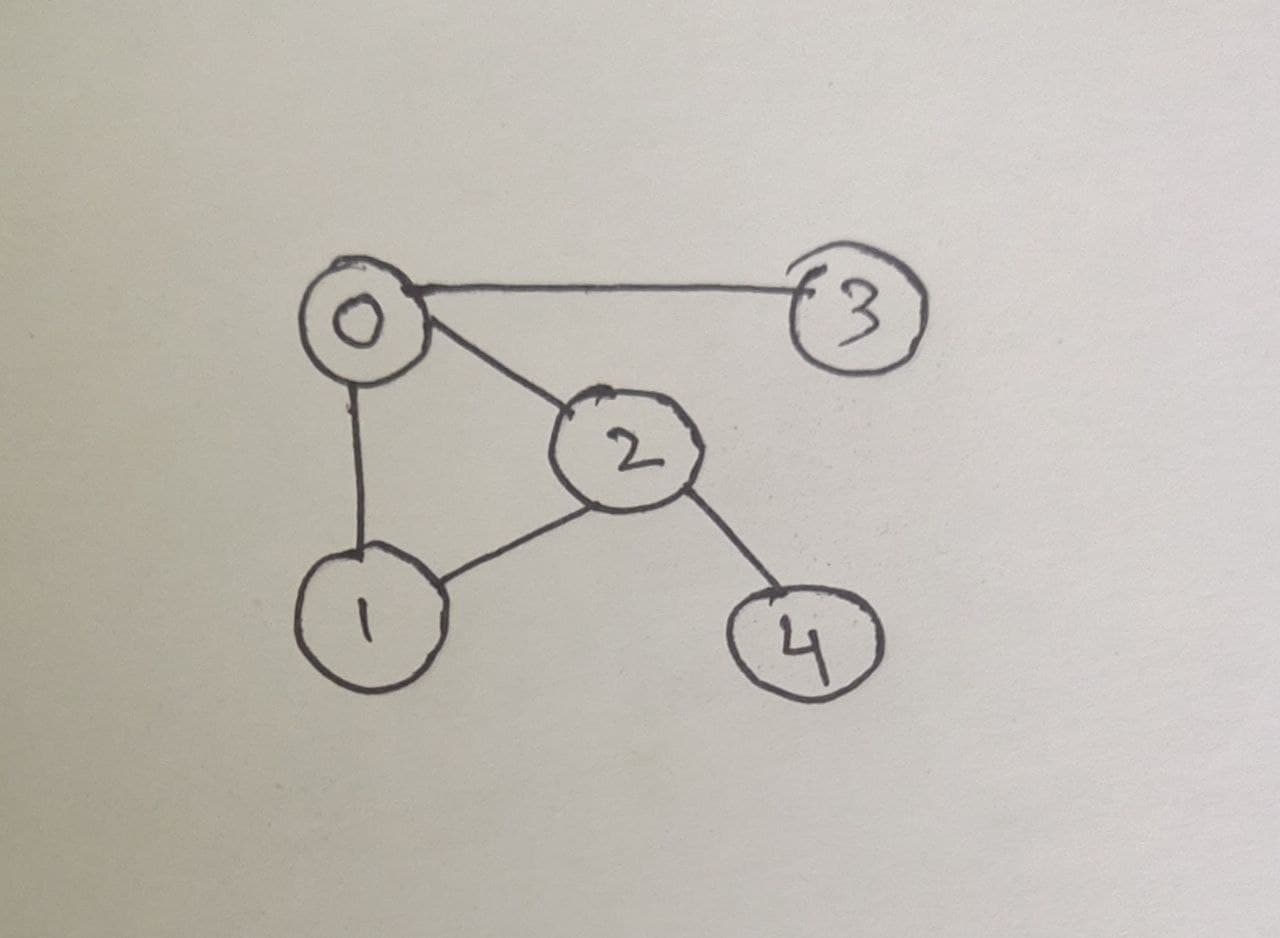
    print("The BFS transversal is: ")

    g.BFS(t)

**Output:**

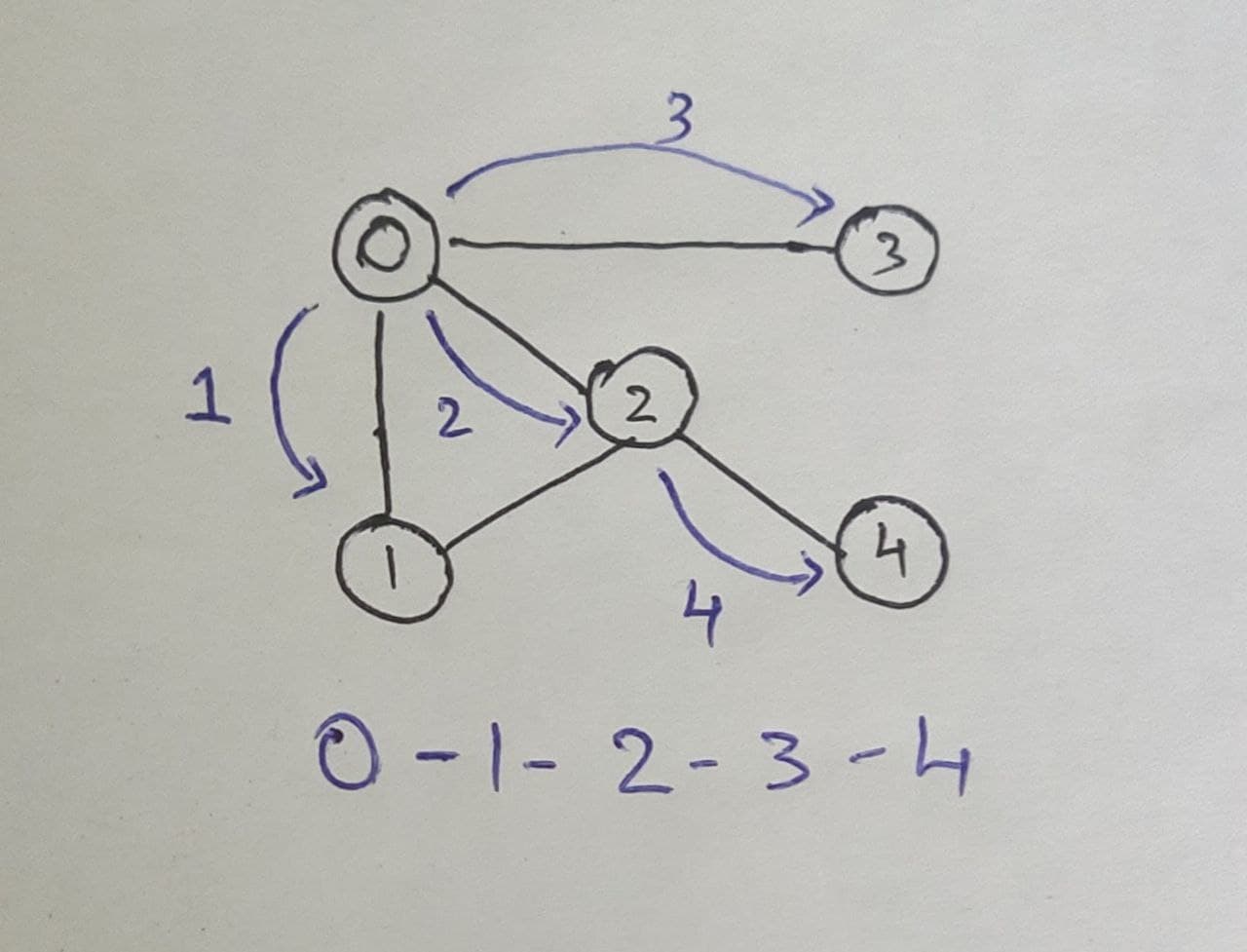
****

**Observations:**

****

Here, we start traversing the undirected graph from 0 and traverse all the adjacent nodes i.e. (1,2,3) and each of them is added to the queue. Then dequeue the first element and traverse all of its adjacent nodes. If there are no adjacent nodes to be visited, we dequeue the next element and repeat the process. Hence, we obtain the BFS

BFS Traversal: 0 1 2 3 4



**Applications of Breadth First Traversal:**

1. Shortest Path and Minimum Spanning Tree for unweighted graph
2. Peer to Peer Networks
3. Crawlers in Search Engines
4. Social Networking Websites
5. GPS Navigation systems

**Result:**

The CSP program has been developed for the graph colouring problem(map colouring problem) applied on the map of a sub-region of India.