

Day-23

Problem 1 : Given an undirected graph, return a vector of all nodes by traversing the graph using depth-first search (DFS).

```
def dfs(graph, start_node, visited, traversal_order):
```

```
    visited[start_node] = True
```

```
    traversal_order.append(start_node)
```

```
    for neighbor in graph[start_node]:
```

```
        if not visited[neighbor]:
```

```
            dfs(graph, neighbor, visited, traversal_order)
```

```
def dfs_traversal(graph):
```

```
    num_nodes = len(graph)
```

```
    visited = [False] * num_nodes
```

```
    traversal_order = []
```

```
    for node in range(num_nodes):
```

```
        if not visited[node]:
```

```
            dfs(graph, node, visited, traversal_order)
```

```
    return traversal_order
```

```
example_graph = {
```

```
    0: [1, 2],
```

```
    1: [0, 2, 3],
```

```
    2: [0, 1, 3],
```

```
    3: [1, 2, 4],
```

```
    4: [3, 5],
```

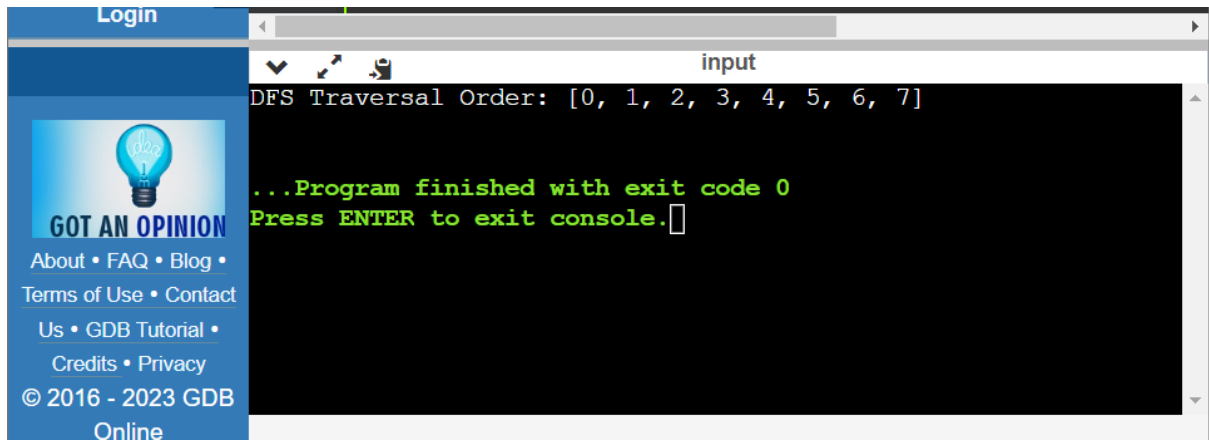
```
    5: [4],
```

```
    6: [7],
```

```
    7: [6]
```

```
}
```

```
traversal_result = dfs_traversal(example_graph)
print("DFS Traversal Order:", traversal_result)
```



The screenshot shows a web browser window with a blue sidebar on the left and a terminal window on the right. The sidebar contains a 'Login' button at the top, followed by a lightbulb icon and the text 'GOT AN OPINION'. Below this are several links: 'About • FAQ • Blog •', 'Terms of Use • Contact', 'Us • GDB Tutorial •', and 'Credits • Privacy'. At the bottom of the sidebar is the copyright notice '© 2016 - 2023 GDB Online'. The terminal window, titled 'input', displays the output of the Python code: 'DFS Traversal Order: [0, 1, 2, 3, 4, 5, 6, 7]'. Below this, it shows '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a cursor.

```
DFS Traversal Order: [0, 1, 2, 3, 4, 5, 6, 7]
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem – 2 : Given an undirected graph, return a vector of all nodes by traversing the graph using breadth-first search (BFS).

```
from collections import deque
```

```
def bfs(graph, start_node):
```

```
    visited = set()
```

```
    queue = deque([start_node])
```

```
    visited.add(start_node)
```

```
    result = []
```

```
    while queue:
```

```
        current_node = queue.popleft()
```

```
        result.append(current_node)
```

```
        for neighbor in graph[current_node]:
```

```
            if neighbor not in visited:
```

```
                visited.add(neighbor)
```

```
                queue.append(neighbor)
```

```
    return result
```

```
graph = {
```

```
    'A': ['B', 'C'],
```

```
    'B': ['A', 'D', 'E'],
```

```
    'C': ['A', 'F', 'G'],
```

```
    'D': ['B'],
```

```
    'E': ['B'],
```

```
    'F': ['C'],
```

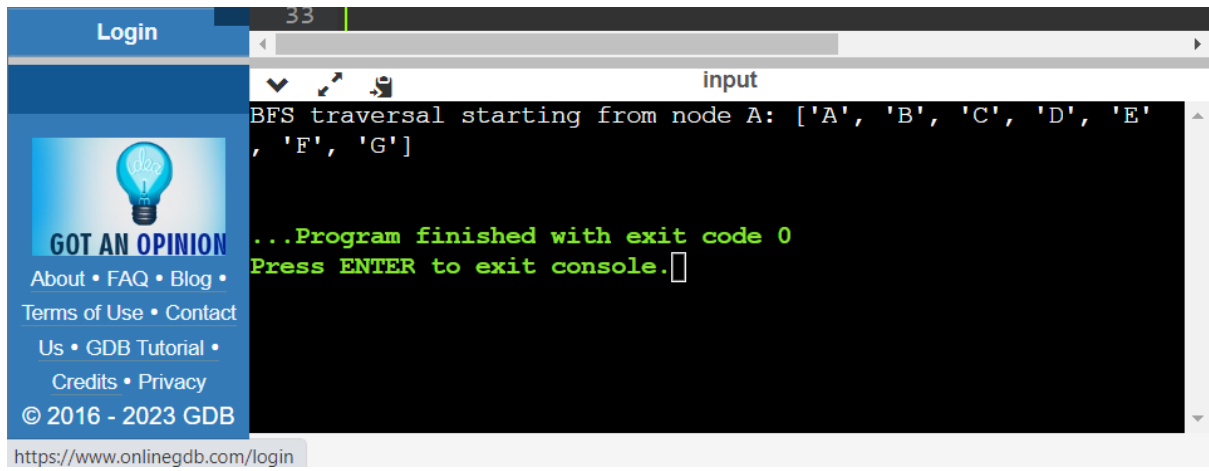
```
    'G': ['C']
```

```
}
```

```
start_node = 'A'
```

```
bfs_result = bfs(graph, start_node)
```

```
print("BFS traversal starting from node {}: {}".format(start_node, bfs_result))
```



The screenshot shows a web browser window. On the left is a sidebar with a blue header 'Login' and a section titled 'GOT AN OPINION' featuring a lightbulb icon. Below this are links: 'About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy' and a copyright notice '© 2016 - 2023 GDB'. The main content area displays a terminal window titled 'input'. The terminal output shows the BFS traversal result: 'BFS traversal starting from node A: ['A', 'B', 'C', 'D', 'E', 'F', 'G']'. It then displays '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a cursor. The browser's address bar at the bottom shows 'https://www.onlinegdb.com/login'.

```
33
```

input

```
BFS traversal starting from node A: ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

```
...Program finished with exit code 0
```

```
Press ENTER to exit console.
```

<https://www.onlinegdb.com/login>

Problem – 3 Given an undirected graph with V vertices and E edges, check whether it contains any cycle or not. (using DFS)

```
from collections import defaultdict
```

```
class Graph:
```

```
    def __init__(self, vertices):
```

```
        self.V = vertices
```

```
        self.graph = defaultdict(list)
```

```
    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)
```

```
        self.graph[v].append(u)
```

```
    def is_cyclic_util(self, v, visited, parent):
```

```
        visited[v] = True
```

```
        for neighbor in self.graph[v]:
```

```
            if not visited[neighbor]:
```

```
                if self.is_cyclic_util(neighbor, visited, v):
```

```
                    return True
```

```
            elif parent != neighbor:
```

```
                return True
```

```
        return False
```

```
    def contains_cycle(self):
```

```
        visited = [False] * self.V
```

```
        for i in range(self.V):
```

```
            if not visited[i]:
```

```
                if self.is_cyclic_util(i, visited, -1):
```

```
return True
```

```
return False
```

```
V = 5
```

```
E = 4
```

```
g = Graph(V)
```

```
edges = [(0, 1), (1, 2), (2, 1), (3, 4)]
```

```
for u, v in edges:
```

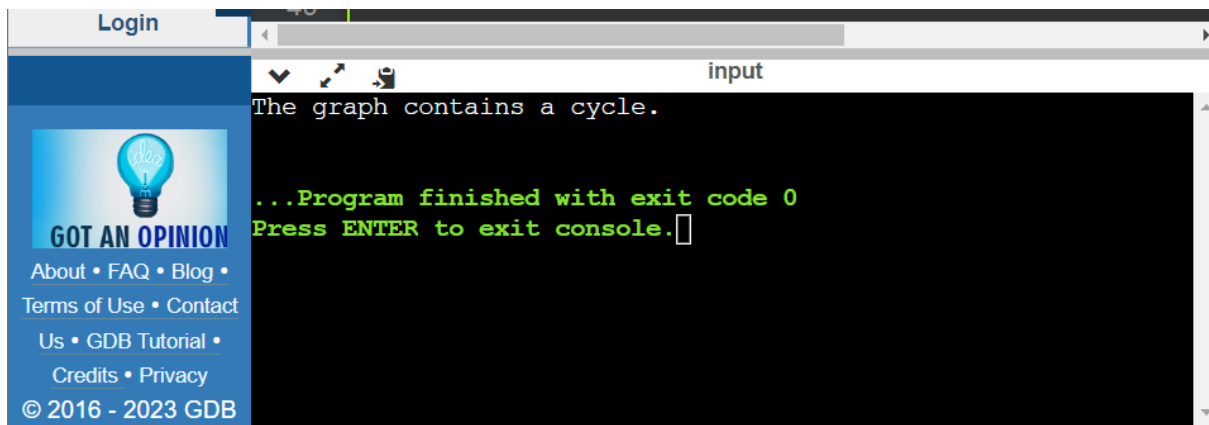
```
    g.add_edge(u, v)
```

```
if g.contains_cycle():
```

```
    print("The graph contains a cycle.")
```

```
else:
```

```
    print("The graph does not contain any cycle.")
```



The screenshot shows a web browser window. On the left is a sidebar with a blue header 'GOT AN OPINION' and a lightbulb icon. Below it are links: 'About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy' and a copyright notice '© 2016 - 2023 GDB'. The main content area shows a terminal window titled 'input' with the output 'The graph contains a cycle.' followed by '...Program finished with exit code 0' and 'Press ENTER to exit console.' with a cursor.

Problem – 4 Given an undirected graph with V vertices and E edges, check whether it contains any cycle or not. (using BFS)

```
from collections import defaultdict, deque
```

```
class Graph:
```

```
    def __init__(self, vertices):
```

```
        self.V = vertices
```

```
        self.graph = defaultdict(list)
```

```
    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)
```

```
        self.graph[v].append(u)
```

```
    def is_cyclic_util(self, v, visited, parent):
```

```
        queue = deque([(v, parent)])
```

```
        visited[v] = True
```

```
        while queue:
```

```
            current_node, parent = queue.popleft()
```

```
            for neighbor in self.graph[current_node]:
```

```
                if not visited[neighbor]:
```

```
                    queue.append((neighbor, current_node))
```

```
                    visited[neighbor] = True
```

```
                elif parent != neighbor:
```

```
                    return True
```

```
        return False
```

```
    def contains_cycle(self):
```

```
        visited = [False] * self.V
```

```

for v in range(self.V):
    if not visited[v]:
        if self.is_cyclic_util(v, visited, -1):
            return True

return False

```

V = 4

E = 4

g = Graph(V)

edges = [(0, 1), (1, 2), (2, 3), (3, 0)]

for u, v in edges:

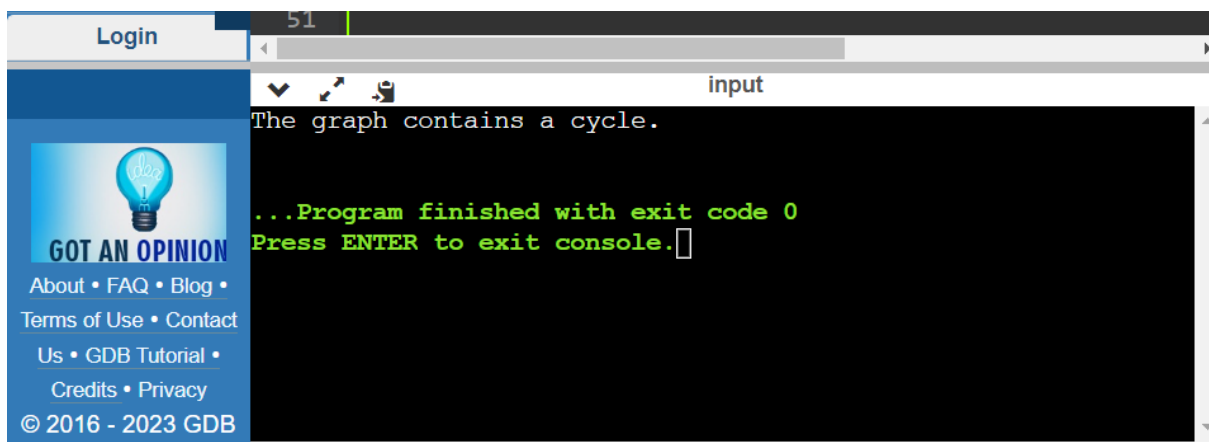
g.add_edge(u, v)

if g.contains_cycle():

print("The graph contains a cycle.")

else:

print("The graph does not contain any cycle.")



The screenshot shows a web browser window. On the left, there is a blue sidebar with a lightbulb icon and the text "GOT AN OPINION". Below this, there are links: "About • FAQ • Blog • Terms of Use • Contact Us • GDB Tutorial • Credits • Privacy" and a copyright notice "© 2016 - 2023 GDB". The main content area of the browser shows a terminal window. The terminal has a title bar with "51" and "input". The output in the terminal is: "The graph contains a cycle." followed by "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor.

Problem 5 Given an directed graph with V vertices and E edges, check whether it contains any cycle or not. (using DFS)

```
from collections import defaultdict
```

```
class Graph:
```

```
    def __init__(self, vertices):
```

```
        self.vertices = vertices
```

```
        self.graph = defaultdict(list)
```

```
    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)
```

```
    def is_cyclic_util(self, v, visited, recursion_stack):
```

```
        visited[v] = True
```

```
        recursion_stack[v] = True
```

```
        for neighbor in self.graph[v]:
```

```
            if not visited[neighbor]:
```

```
                if self.is_cyclic_util(neighbor, visited, recursion_stack):
```

```
                    return True
```

```
            elif recursion_stack[neighbor]:
```

```
                return True
```

```
        recursion_stack[v] = False
```

```
        return False
```

```
    def contains_cycle(self):
```

```
        visited = [False] * self.vertices
```

```
        recursion_stack = [False] * self.vertices
```

```
        for v in range(self.vertices):
```

```
    if not visited[v]:  
        if self.is_cyclic_util(v, visited, recursion_stack):  
            return True
```

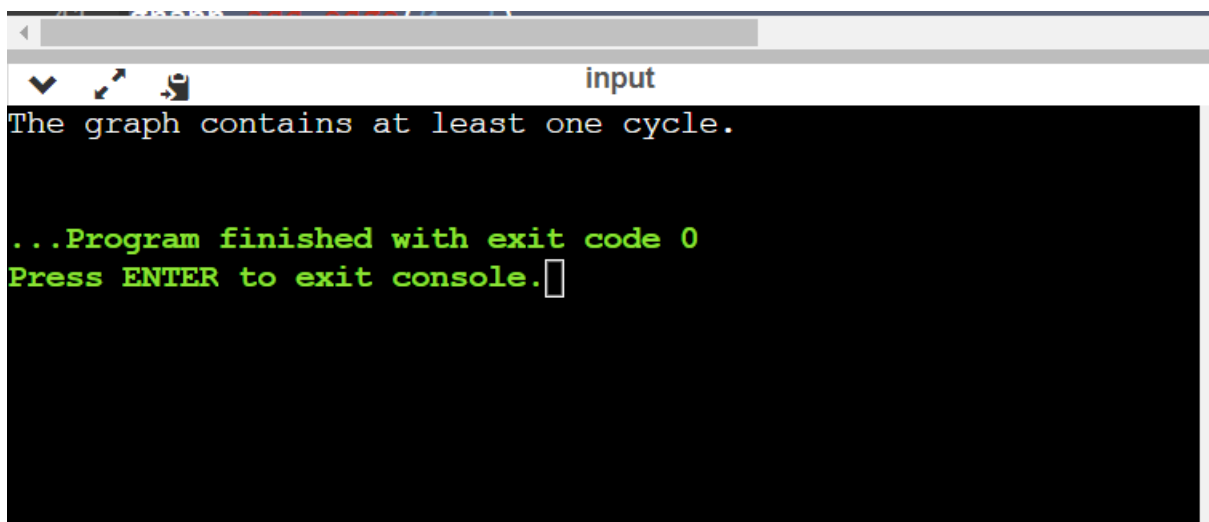
```
    return False
```

V = 4

E = 6

```
graph = Graph(V)  
graph.add_edge(0, 1)  
graph.add_edge(0, 2)  
graph.add_edge(1, 2)  
graph.add_edge(2, 0)  
graph.add_edge(2, 3)  
graph.add_edge(3, 3)
```

```
if graph.contains_cycle():  
    print("The graph contains at least one cycle.")  
else:  
    print("The graph does not contain any cycle.")
```



```
input  
The graph contains at least one cycle.  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

Problem – 6 : Topological Sort (BFS)

```
from collections import defaultdict, deque
```

```
def topological_sort_bfs(graph):
```

```
    in_degree = {node: 0 for node in graph}
```

```
    for node in graph:
```

```
        for neighbor in graph[node]:
```

```
            in_degree[neighbor] += 1
```

```
    queue = deque()
```

```
    for node in in_degree:
```

```
        if in_degree[node] == 0:
```

```
            queue.append(node)
```

```
    topological_order = []
```

```
    while queue:
```

```
        node = queue.popleft()
```

```
        topological_order.append(node)
```

```
        for neighbor in graph[node]:
```

```
            in_degree[neighbor] -= 1
```

```
            if in_degree[neighbor] == 0:
```

```
                queue.append(neighbor)
```

```
    if len(topological_order) != len(graph):
```

```
        # The graph contains cycles, so topological sort is not possible
```

```
        return None
```

```
    return topological_order
```

```
graph = {
```

```
    'A': ['B', 'C'],
```

```
'B': ['D'],  
'C': ['D'],  
'D': ['E'],  
'E': []  
}
```

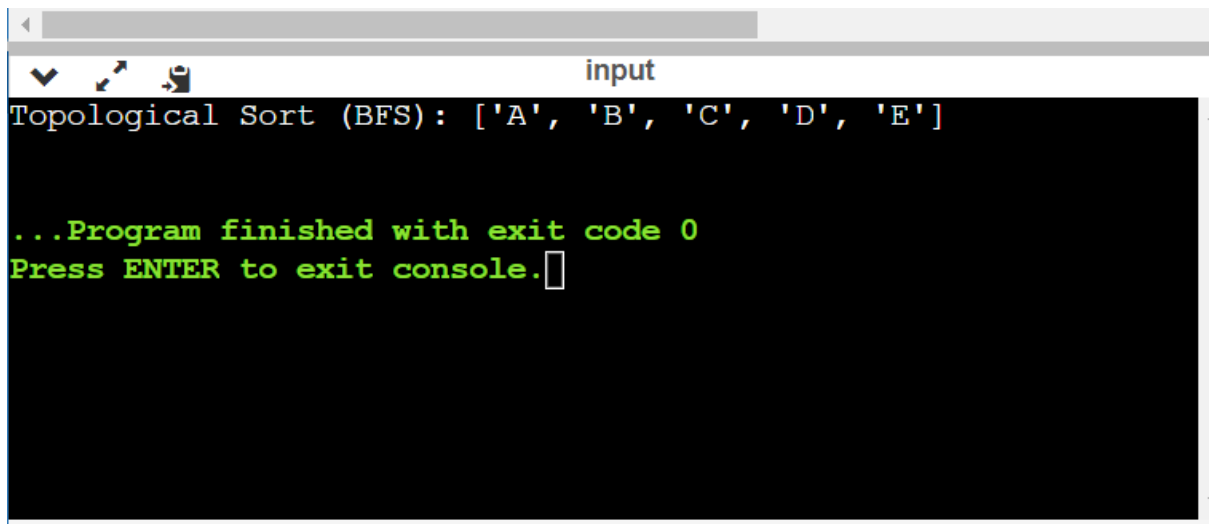
```
result = topological_sort_bfs(graph)
```

```
if result is not None:
```

```
    print("Topological Sort (BFS):", result)
```

```
else:
```

```
    print("The graph contains cycles. Topological Sort is not possible.")
```



The screenshot shows a terminal window with a title bar that includes a back arrow, a maximize button, and a close button, followed by the title "input". The terminal content displays the output of a topological sort algorithm: "Topological Sort (BFS): ['A', 'B', 'C', 'D', 'E']". Below this, a green message states "...Program finished with exit code 0". At the bottom, another green message says "Press ENTER to exit console." followed by a cursor icon.

```
input  
Topological Sort (BFS): ['A', 'B', 'C', 'D', 'E']  
  
...Program finished with exit code 0  
Press ENTER to exit console.█
```

Problem – 7 : Topological Sort (DFS)

```
from collections import defaultdict
```

```
class Graph:
```

```
    def __init__(self):
```

```
        self.graph = defaultdict(list)
```

```
    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)
```

```
    def topological_sort_util(self, v, visited, stack):
```

```
        visited[v] = True
```

```
        for neighbor in self.graph[v]:
```

```
            if not visited[neighbor]:
```

```
                self.topological_sort_util(neighbor, visited, stack)
```

```
        stack.append(v)
```

```
    def topological_sort(self):
```

```
        visited = [False] * (max(self.graph) + 1)
```

```
        stack = []
```

```
        for i in range(len(visited)):
```

```
            if not visited[i]:
```

```
                self.topological_sort_util(i, visited, stack)
```

```
        return stack[::-1]
```

```
g = Graph()
```

```
g.add_edge(5, 2)
```

```
g.add_edge(5, 0)
```

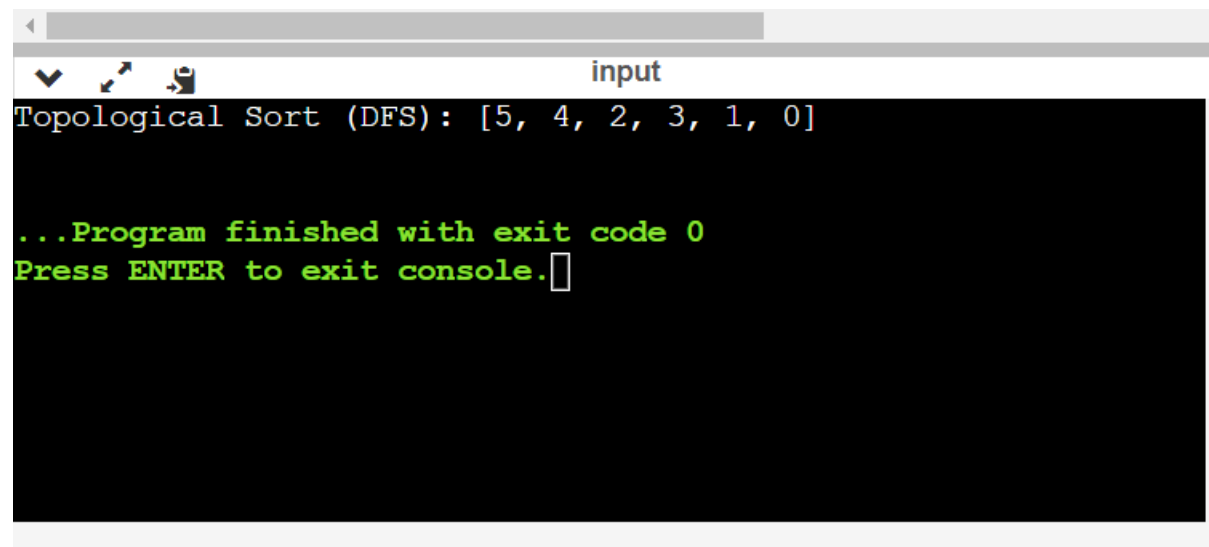
```
g.add_edge(4, 0)
```

```
g.add_edge(4, 1)
```

```
g.add_edge(2, 3)
```

```
g.add_edge(3, 1)
```

```
print("Topological Sort (DFS):", g.topological_sort())
```



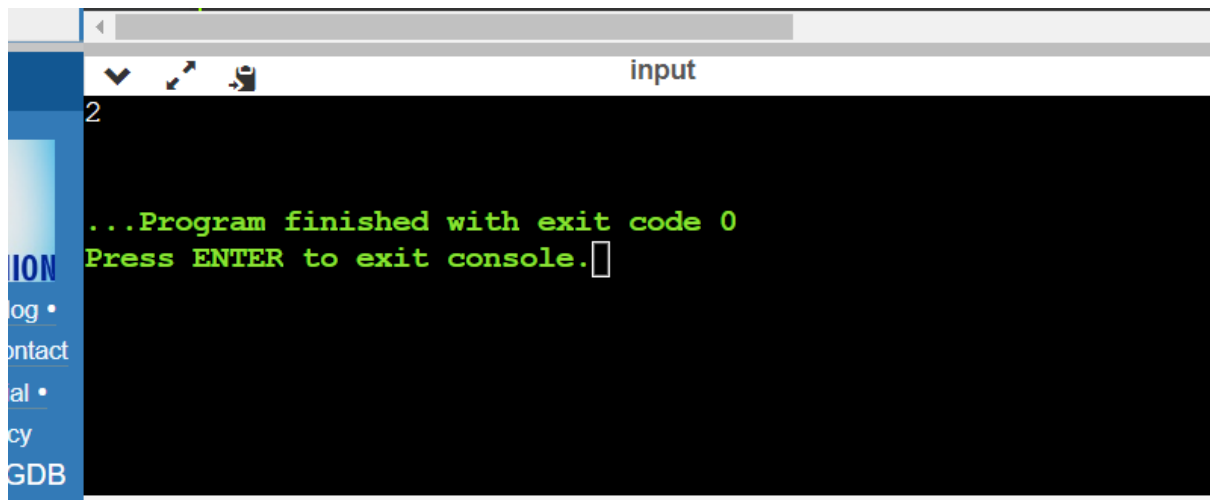
```
Topological Sort (DFS): [5, 4, 2, 3, 1, 0]  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

Problem – 8 Given a boolean 2D matrix grid of size N x M. You have to find the number of distinct islands where a group of connected 1s (horizontally or vertically) forms an island. Two islands are considered to be distinct if and only if one island is equal to another (not rotated or reflected).

```
def numDistinctIslands(grid):  
    def dfs(x, y, shape):  
        if 0 <= x < len(grid) and 0 <= y < len(grid[0]) and grid[x][y] == 1:  
            grid[x][y] = 0  
            shape.append((x, y))  
            dfs(x + 1, y, shape)  
            dfs(x - 1, y, shape)  
            dfs(x, y + 1, shape)  
            dfs(x, y - 1, shape)  
  
    unique_shapes = set()  
    for i in range(len(grid)):  
        for j in range(len(grid[0])):  
            if grid[i][j] == 1:  
                shape = []  
                dfs(i, j, shape)  
                unique_shapes.add(tuple(shape))  
  
    return len(unique_shapes)
```

```
grid = [  
    [1, 1, 0, 0, 0],  
    [1, 1, 0, 0, 0],  
    [0, 0, 0, 1, 1],  
    [0, 0, 0, 1, 1]  
]
```

```
print(numDistinctIslands(grid))
```



Problem – 9,10 : Bipartite Check using :

9. DFS

10. BFS

```
from collections import deque
```

```
def is_bipartite_bfs(graph, start_node):
```

```
    queue = deque([start_node])
```

```
    color = {start_node: 1} # Colors: 1 and -1 represent the two partitions.
```

```
    while queue:
```

```
        current_node = queue.popleft()
```

```
        for neighbor in graph[current_node]:
```

```
            if neighbor not in color:
```

```
                color[neighbor] = -color[current_node]
```

```
                queue.append(neighbor)
```

```
            elif color[neighbor] == color[current_node]:
```

```
                return False
```

```
    return True
```

```
def is_bipartite_dfs(graph, start_node):
```

```
    stack = [(start_node, 1)]
```

```
    color = {}
```

```
    while stack:
```

```
        current_node, current_color = stack.pop()
```

```
        if current_node in color:
```

```
            if color[current_node] != current_color:
```

```
                return False
```

```
            continue
```

```
        color[current_node] = current_color
```

```
next_color = -current_color
```

```
stack.extend((neighbor, next_color) for neighbor in graph[current_node])
```

```
return True
```

```
def is_bipartite(graph):
```

```
    start_node = next(iter(graph))
```

```
    return is_bipartite_bfs(graph, start_node) and is_bipartite_dfs(graph, start_node)
```

```
graph = {
```

```
    1: [2, 3],
```

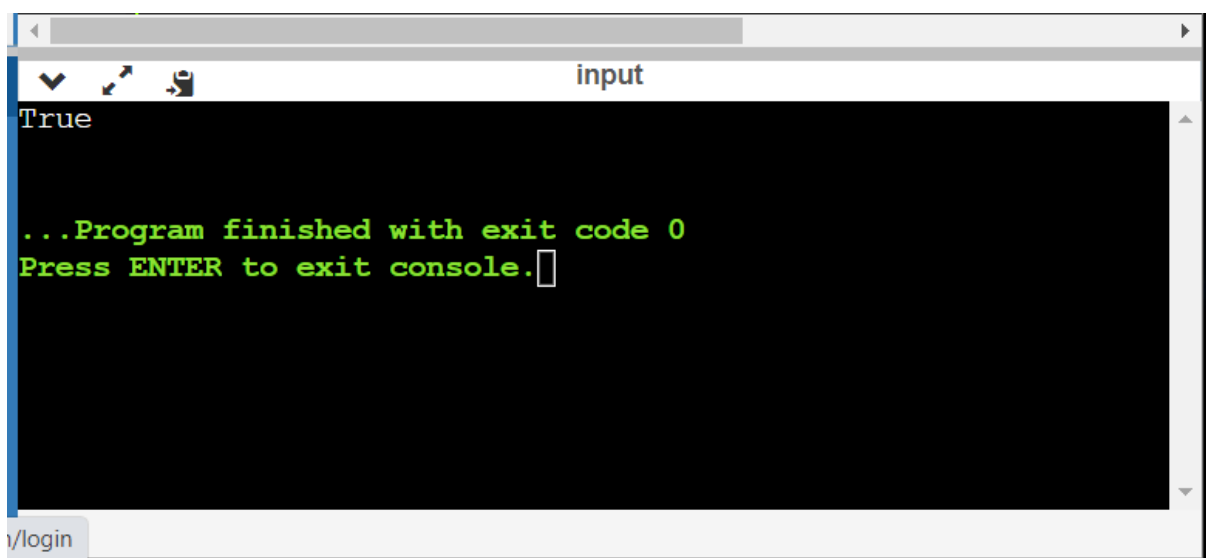
```
    2: [1, 4],
```

```
    3: [1, 4],
```

```
    4: [2, 3]
```

```
}
```

```
print(is_bipartite(graph))
```



```
True

...Program finished with exit code 0
Press ENTER to exit console.
```