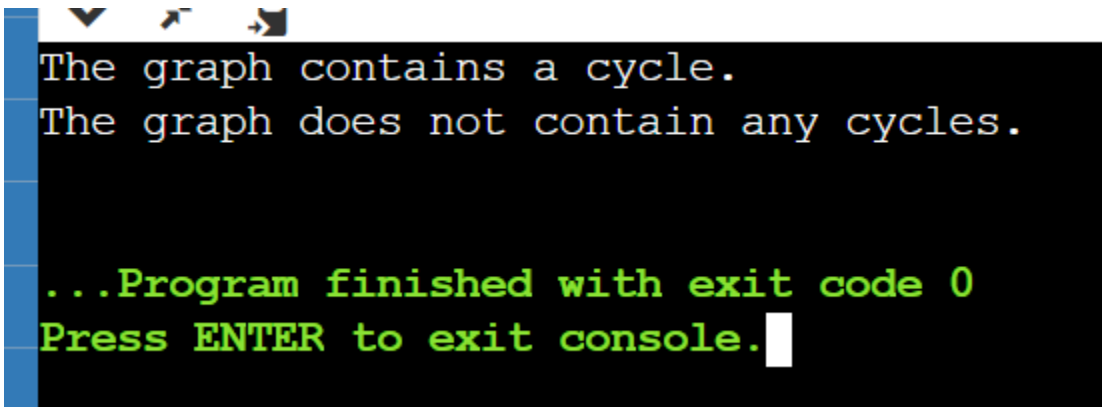
BFS IN DIRECTED GRAPH:

CODE:

```
from collections import defaultdict, deque

def contains_cycle(graph, start):
    visited = set()
    queue = deque([(start, [])])

    while queue:
        node, path = queue.popleft()

        if node in visited:
            return True

        visited.add(node)

        for neighbor in graph[node]:
            queue.append((neighbor, path + [node]))

    return False

# Example directed graph represented as an adjacency list (with cycle)
graph_with_cycle = {
    1: [2],
    2: [3, 4],
    3: [4],
    4: [1]
}

if contains_cycle(graph_with_cycle, 1):
    print("The graph contains a cycle.")
else:
    print("The graph does not contain any cycles.")


# Example directed graph represented as an adjacency list (without cycle)
graph_without_cycle = {
    1: [2],
```

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

if contains_cycle(graph_without_cycle, 1):
    print("The graph contains a cycle.")
else:
    print("The graph does not contain any cycles.")
```

Output:



BFS IN UNDIRECTED GRAPH:

CODE:

```
from collections import defaultdict, deque

def has_cycle(graph, start):
    visited = set()
    queue = deque([(start, None)])
```

```
    while queue:
        node, parent = queue.popleft()

        if node in visited:
            if parent is not None and parent != node:
                return True
            continue

        visited.add(node)

        for neighbor in graph[node]:
            queue.append((neighbor, node))

    return False

# Example undirected graph represented as an adjacency list (with cycle)
graph_with_cycle = {
    1: [2, 3],
    2: [1, 4],
    3: [1, 5],
    4: [2],
    5: [3]
}

has_cycle_result = False
for node in graph_with_cycle:
    if has_cycle(graph_with_cycle, node):
        has_cycle_result = True
        break

if has_cycle_result:
    print("The graph contains a cycle.")
else:
    print("The graph does not contain any cycles.")
```

```
The graph contains a cycle.


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