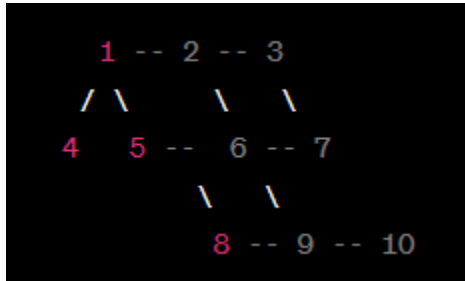


Task 1 – Breadth First Search (BFS):

(i) Create a graph with at least 10 nodes:

Let's create an example undirected graph with 10 nodes and 11 edges:



(ii) Write a program in Python to perform a Breadth First Search to find a specific node in the graph:

Python program to perform BFS on the given graph:

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

```
class Graph:
```

```
    def __init__(self):
        self.graph = defaultdict(list)
```

```
    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)
```

```
    def bfs(self, start, target):
        visited = set()
        queue = deque([start])
```

```
        while queue:
            current = queue.popleft()
            if current == target:
                return True
```

```
        visited.add(current)
        for neighbor in self.graph[current]:
            if neighbor not in visited:
                queue.append(neighbor)
```

```
        return False
```

```

# Create the graph
graph = Graph()
graph.add_edge(1, 2)
graph.add_edge(1, 4)
graph.add_edge(2, 3)
graph.add_edge(2, 5)
graph.add_edge(2, 6)
graph.add_edge(3, 6)
graph.add_edge(3, 7)
graph.add_edge(5, 6)
graph.add_edge(5, 8)
graph.add_edge(6, 9)
graph.add_edge(9, 10)

# Perform BFS and search for node 10 starting from node 1
start_node = 1
target_node = 10
result = graph.bfs(start_node, target_node)

if result:
    print(f"Node {target_node} is reachable from Node {start_node} using BFS.")
else:
    print(f"Node {target_node} is NOT reachable from Node {start_node} using BFS.")

```

(iii) Description and Output:

In the given graph, each node represents a unique vertex, and each edge represents an undirected connection between two vertices. We created a graph with 10 nodes and used adjacency lists to represent the connections between the nodes.

The BFS algorithm starts from a given start node and explores its neighbors level by level. It uses a queue data structure to keep track of the nodes to be visited.

In the Python code, we perform a BFS starting from node 1 and search for node 10. The output will indicate whether node 10 is reachable from node 1 using BFS.

Task 2 – Depth First Search (DFS):

To perform DFS on the same graph, we need to modify the `Graph` class to implement the DFS algorithm.

Python code:

```

class Graph:
    def __init__(self):

```

```
self.graph = defaultdict(list)

def add_edge(self, u, v):
    self.graph[u].append(v)
    self.graph[v].append(u)

def dfs(self, start, target, visited):
    if start == target:
        return True

    visited.add(start)
    for neighbor in self.graph[start]:
        if neighbor not in visited:
            if self.dfs(neighbor, target, visited):
                return True

    return False
```

```
# Create the graph (same as before)
```

```
# Perform DFS and search for node 10 starting from node 1
```

```
start_node = 1
```

```
target_node = 10
```

```
visited_nodes = set()
```

```
result = graph.dfs(start_node, target_node, visited_nodes)
```

```
if result:
```

```
    print(f"Node {target_node} is reachable from Node {start_node} using DFS.")
```

```
else:
```

```
    print(f"Node {target_node} is NOT reachable from Node {start_node} using DFS.")
```

```

C:\Users\Yoann\Desktop> python tp4.py > ...
1  from collections import defaultdict, deque
2
3  class Graph:
4      def __init__(self):
5          self.graph = defaultdict(list)
6
7      def add_edge(self, u, v):
8          self.graph[u].append(v)
9          self.graph[v].append(u)
10
11     def bfs(self, start, target):
12         visited = set()
13         queue = deque([start])
14
15         while queue:
16             current = queue.popleft()
17             if current == target:
18                 return True
19
20             visited.add(current)
21             for neighbor in self.graph[current]:
22                 if neighbor not in visited:
23                     queue.append(neighbor)
24
25         return False
26
27     # Create the graph
28     graph = Graph()
29     graph.add_edge(1, 2)
30     graph.add_edge(1, 4)
31     graph.add_edge(2, 3)
32     graph.add_edge(2, 5)
33     graph.add_edge(2, 6)
34     graph.add_edge(3, 6)
35     graph.add_edge(3, 7)
36     graph.add_edge(5, 6)
37     graph.add_edge(5, 8)
38     graph.add_edge(6, 9)
39     graph.add_edge(9, 10)
40
41     # Perform BFS and search for node 10 starting from node 1
42     start_node = 1
43     target_node = 10
44     result = graph.bfs(start_node, target_node)
45
46     if result:
47         print(f"Node {target_node} is reachable from Node {start_node} using BFS.")
48     else:
49         print(f"Node {target_node} is NOT reachable from Node {start_node} using BFS.")

```

```

C:\Users\Yoann\Desktop>python tp4.py
Node 10 is reachable from Node 1 using BFS.

```