

Name:-

Roll No.:-

Div:-

Title:- Implement DFS and BFS Algorithm. Use an Undirected Graph and develop a Recursive Algorithm for searching all the vertices of the graph or tree data structure.

Program :-

Breadth First Search(BFS):-

```
graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = [] # List to keep track of visited nodes.
queue = []    #Initialize a queue

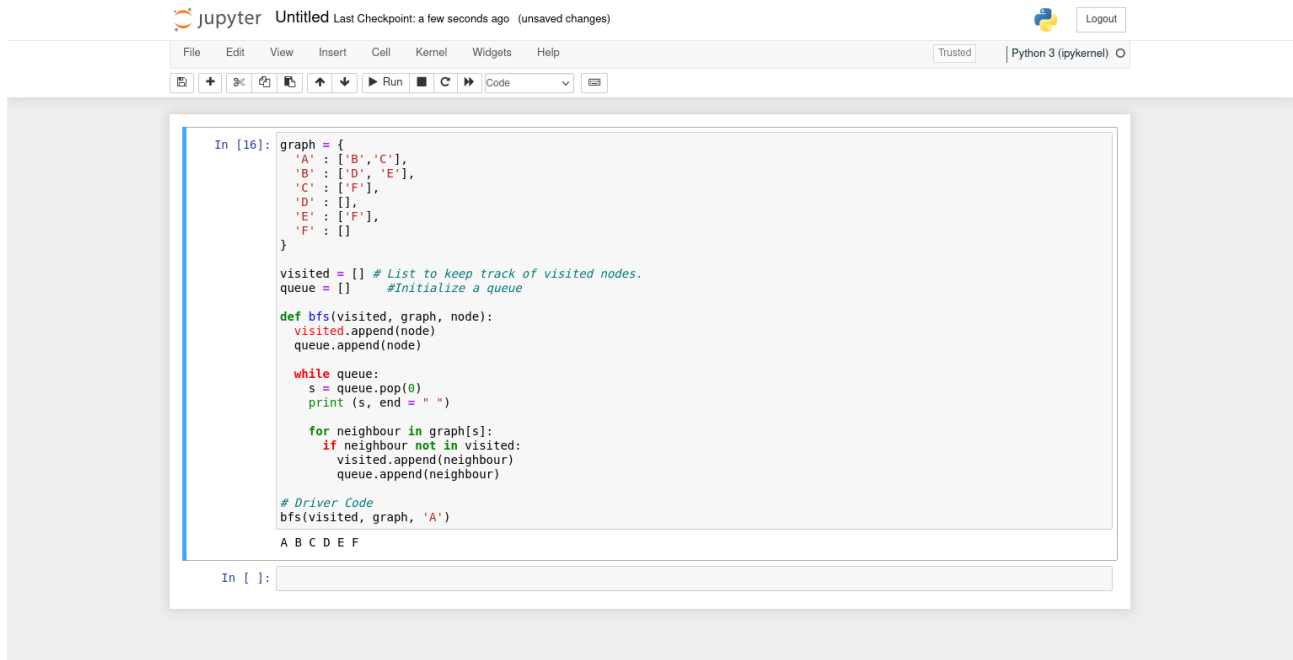
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
```

bfs(visited, graph, 'A')

A screenshot of a Jupyter Notebook interface. The top bar shows 'Jupyter' and 'Untitled' with a timestamp 'Last Checkpoint: a few seconds ago (unsaved changes)'. The right side has a 'Logout' button. Below the top bar is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. A toolbar with icons for file operations and execution is below the menu. The main area contains a code cell with the following Python code:

```
In [16]: graph = {
    'A' : ['B','C'],
    'B' : ['D', 'E'],
    'C' : ['F'],
    'D' : [],
    'E' : ['F'],
    'F' : []
}

visited = [] # List to keep track of visited nodes.
queue = []   # Initialize a queue

def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)

    while queue:
        s = queue.pop(0)
        print (s, end = " ")

        for neighbour in graph[s]:
            if neighbour not in visited:
                visited.append(neighbour)
                queue.append(neighbour)

# Driver Code
bfs(visited, graph, 'A')

A B C D E F
```

The output of the code is 'A B C D E F'.

2. Depth-first Search:

Using a Python dictionary to act as an adjacency list

```
graph = {
'A' : ['B','C'],
'B' : ['D', 'E'],
'C' : ['F'],
'D' : [],
'E' : ['F'],
'F' : []
}
```

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs

if node not in visited:

print (node)

visited.add(node)

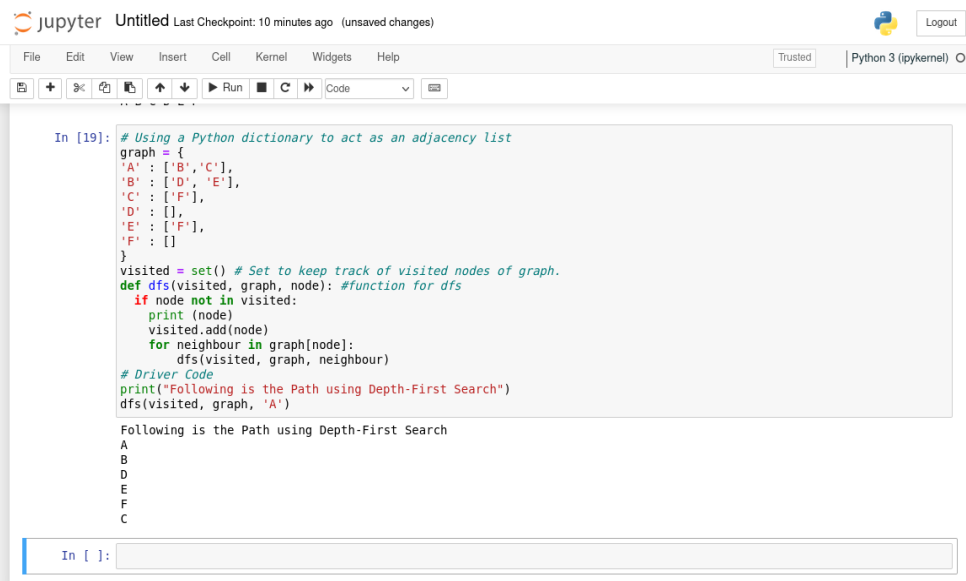
for neighbour in graph[node]:

dfs(visited, graph, neighbour)

```
# Driver Code
```

```
print("Following is the Path using Depth-First Search")
```

```
dfs(visited, graph, 'A')
```



The image shows a Jupyter Notebook interface with a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar indicating 'Python 3 (ipykernel)'. The notebook contains a code cell with the following Python code:

```
In [19]: # Using a Python dictionary to act as an adjacency list
graph = {
    'A': ['B','C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
visited = set() # Set to keep track of visited nodes of graph.
def dfs(visited, graph, node): #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
# Driver Code
print("Following is the Path using Depth-First Search")
dfs(visited, graph, 'A')
```

The output of the code cell is:

```
Following is the Path using Depth-First Search
A
B
D
E
F
C
```

Below the code cell, there is an input prompt 'In []:' followed by an empty text box.