

**Title: Analysis of DFS and BFS for same application**

**Ex. No.:04**

**Reg. No.: RA2011003011334**

**Date: 09/02/23**

**Name: RISHABH SINGH SAHIL**

**Aim:**

Implementation and Analysis of DFS and BFS for same application

**DFS Program:**

class Graph:

    # Constructor

    def \_\_init\_\_(self):

        # default dictionary to store graph

        self.graph = defaultdict(list)

    # function to add an edge to graph

    def addEdge(self, u, v):

        self.graph[u].append(v)

    # A function used by DFS

    def DFSUtil(self, v, visited):

        # Mark the current node as visited

        # and print it

        visited.add(v)

        print(v, end=' ')

        # Recur for all the vertices

        # adjacent to this vertex

        for neighbour in self.graph[v]:

            if neighbour not in visited:

                self.DFSUtil(neighbour, visited)

    # The function to do DFS traversal. It uses

    # recursive DFSUtil()

    def DFS(self, v):

        # Create a set to store visited vertices

        visited = set()

        # Call the recursive helper function

        # to print DFS traversal

        self.DFSUtil(v, visited)

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

    g = Graph()

    g.addEdge(0, 1)

    g.addEdge(0, 2)

    g.addEdge(1, 2)

```

g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is DFS from (starting from vertex 2)")
# Function call
g.DFS(2)

```

### Output:

The screenshot shows a code editor with a file named 'main.py'. The code defines a 'Graph' class with methods for adding edges and performing a Depth-First Search (DFS). The DFS method is recursive and prints the vertices in the order they are visited. The output in the 'Shell' pane shows the DFS starting from vertex 2, visiting vertices 2, 0, 1, and 3 in that order.

```

main.py
1
2 from collections import defaultdict
3 class Graph:
4
5     # Constructor
6     def __init__(self):
7
8         # default dictionary to store graph
9         self.graph = defaultdict(list)
10
11     # function to add an edge to graph
12     def addEdge(self, u, v):
13         self.graph[u].append(v)
14
15     # A function used by DFS
16     def DFSUtil(self, v, visited):
17
18         # Mark the current node as visited
19         # and print it
20         visited.add(v)
21         print(v, end=' ')
22
23         # Recur for all the vertices
24         # adjacent to this vertex
25         for neighbour in self.graph[v]:

```

```

Following is DFS from (starting from vertex 2)
>
> 2 0 1 3 >
> |
>

```

### BFS Program:

```
from collections import defaultdict
```

```
class Graph:
```

```
    # Constructor
```

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

```
        # default dictionary to store graph
```

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

```
    # function to add an edge to graph
```

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

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

```
    # Function to print a BFS of graph
```

```
    def BFS(self, s):
```

```
        # Mark all the vertices as not visited
```

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

```
        # Create a queue for BFS
```

```
        queue = []
```

```
        # Mark the source node as
```

```
        # visited and enqueue it
```

```
        queue.append(s)
```

```
visited[s] = True
```

```
while queue:
```

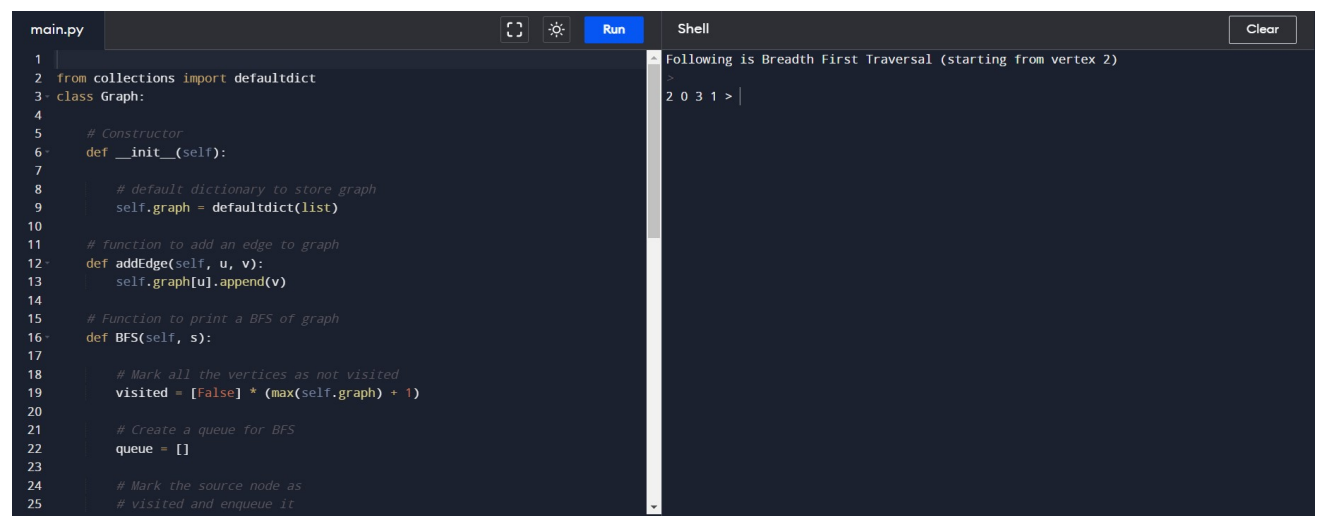
```
    # Dequeue a vertex from  
    # queue and print it  
    s = queue.pop(0)  
    print(s, end=" ")
```

```
# Create a graph given in  
# the above diagram
```

```
g = Graph()  
g.addEdge(0, 1)  
g.addEdge(0, 2)  
g.addEdge(1, 2)  
g.addEdge(2, 0)  
g.addEdge(2, 3)  
g.addEdge(3, 3)
```

```
print("Following is Breadth First Traversal"  
      " (starting from vertex 2)")  
g.BFS(2)
```

### **Output:**



The screenshot shows a code editor with a file named 'main.py'. The code defines a 'Graph' class with methods for adding edges and performing a Breadth First Search (BFS). The BFS method starts at vertex 2 and traverses the graph, printing the vertices in the order they are visited. The output in the shell window is: 'Following is Breadth First Traversal (starting from vertex 2)' followed by '2 0 3 1 >'.

```
main.py  Run  Shell  Clear
```

```
1  
2 from collections import defaultdict  
3 class Graph:  
4  
5     # Constructor  
6     def __init__(self):  
7  
8         # default dictionary to store graph  
9         self.graph = defaultdict(list)  
10  
11     # function to add an edge to graph  
12     def addEdge(self, u, v):  
13         self.graph[u].append(v)  
14  
15     # Function to print a BFS of graph  
16     def BFS(self, s):  
17  
18         # Mark all the vertices as not visited  
19         visited = [False] * (max(self.graph) + 1)  
20  
21         # Create a queue for BFS  
22         queue = []  
23  
24         # Mark the source node as  
25         # visited and enqueue it
```

```
Following is Breadth First Traversal (starting from vertex 2)  
>  
2 0 3 1 >
```

### **Result:**

Successfully DFS and BFS for same application .

