**Program-1**

**Aim :-** Write a Program to Implement Breadth First Search in Python.

**BFS :-** The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

**How does BFS work?**

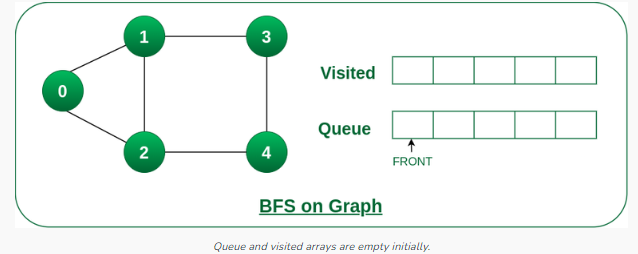
Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.

To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

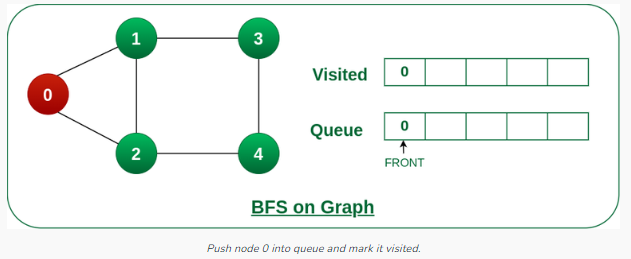
**Illustration:**

Let us understand the working of the algorithm with the help of the following example.

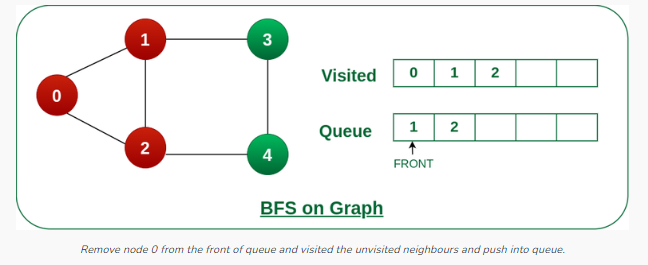
**Step1:**Initially queue and visited arrays are empty.



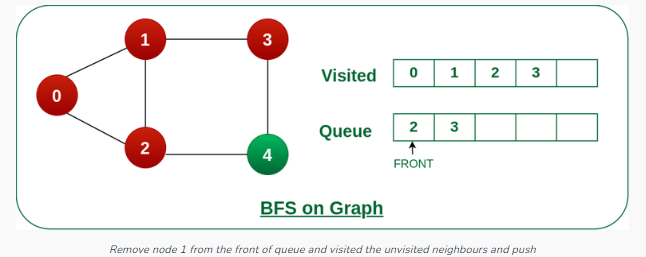
**Step2:**Push node 0 into queue and mark it visited.



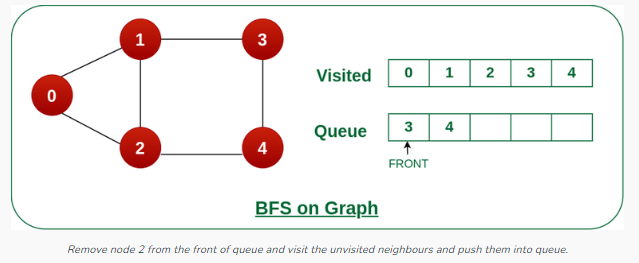
**Step 3:** Remove node 0 from the front of queue and visit the unvisited neighbours and push them into queue.



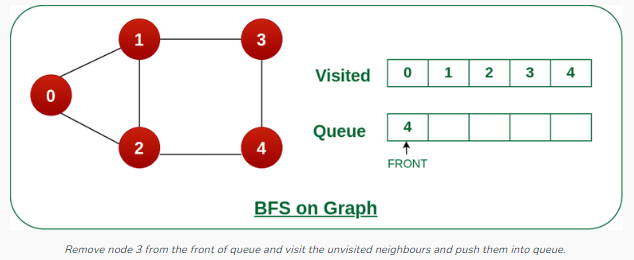
**Step 4:** Remove node 1 from the front of queue and visit the unvisited neighbours and push them into queue.



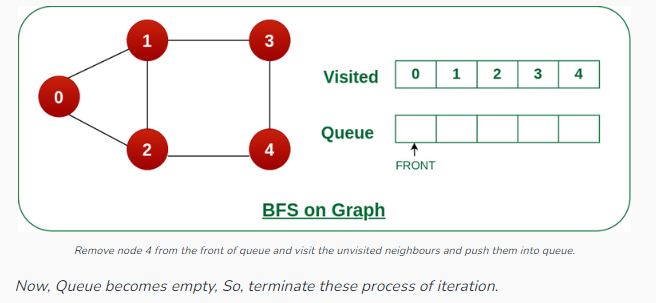
**Step 5:** Remove node 2 from the front of queue and visit the unvisited neighbours and push them into queue.



**Step 6:**Remove node 3 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 3 is visited, so move to the next node that are in the front of the queue.



**Steps 7:**Remove node 4 from the front of queue and visit the unvisited neighbours and push them into queue.   
As we can see that every neighbours of node 4 are visited, so move to the next node that is in the front of the queue.



**Code :-**

**from** collections **import** defaultdict

**class** Graph:

**def** \_\_init\_\_(self):

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

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

        self.graph[u].append(v)

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

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

       queue **=** []

       queue.append(s)

        visited[s] **=** True

**while** queue:

           s **=** queue.pop(0)

            print(s, end**=**" ")

**for** i **in** self.graph[s]:

**if** visited[i] **==** False:

                    queue.append(i)

                    visited[i] **=** True

**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)

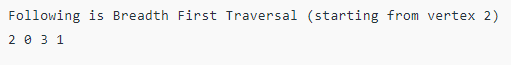
    g.addEdge(3, 3)

**print**("Following is Breadth First Traversal"

          " (starting from vertex 2)")

    g.BFS(2)

**Output:-**

****

**Program-2**

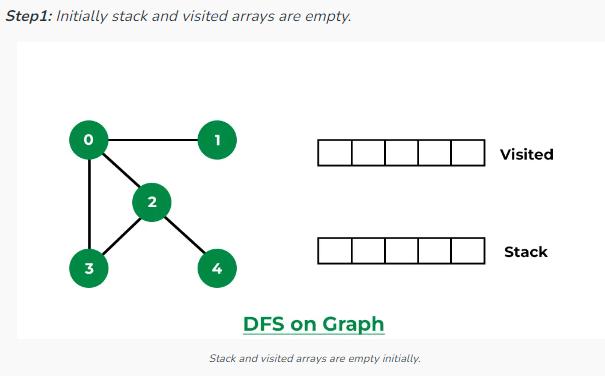
**Aim :-** Write a Program to Implement Depth First Search in Python.

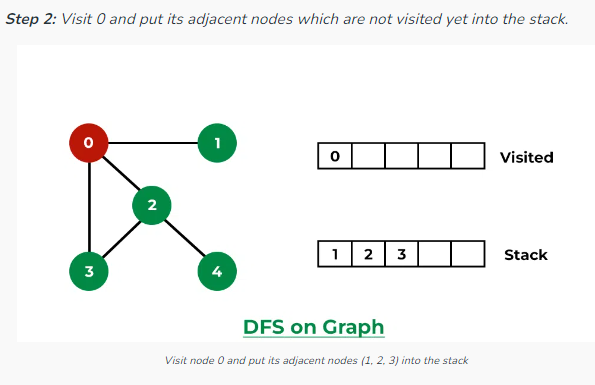
**BFS :- Depth First Traversal (or DFS)** for a graph is similar to [Depth First Traversal of a tree.](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/) The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

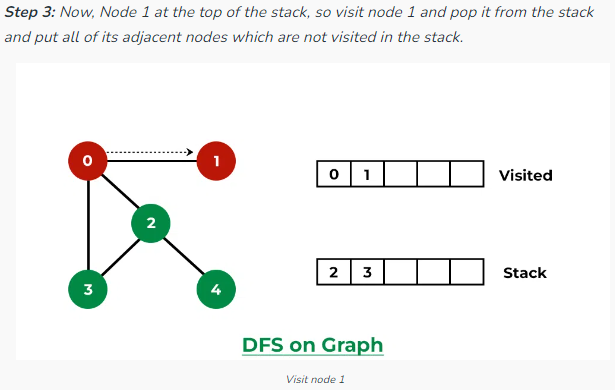
**How does DFS work?**

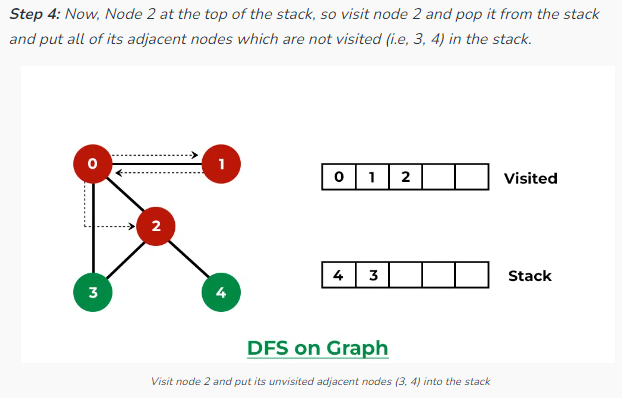
Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

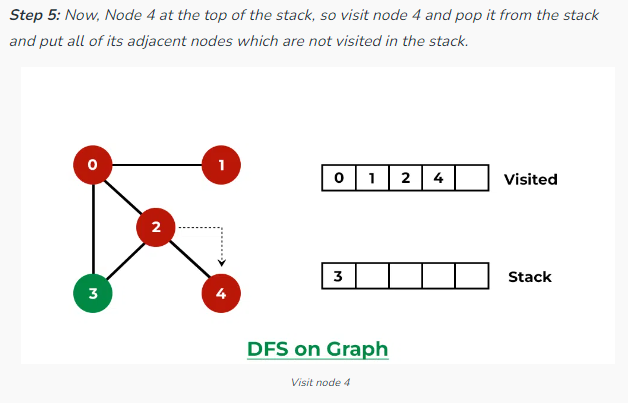
Let us understand the working of **Depth First Search** with the help of the following illustration:

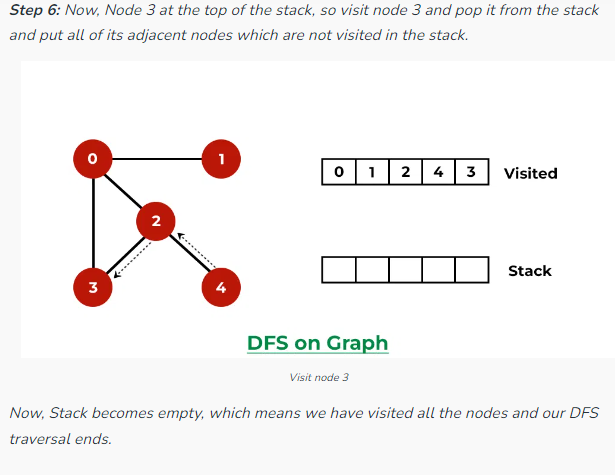












**Code:-**

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.graph = defaultdict(list)

def addEdge(self, u, v):

self.graph[u].append(v)

def DFSUtil(self, v, visited):

visited.add(v)

print(v, end=' ')

for neighbour in self.graph[v]:

if neighbour not in visited:

self.DFSUtil(neighbour, visited)

def DFS(self, v):

visited = set()

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)

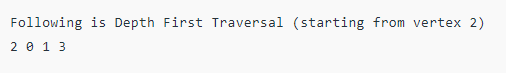
g.addEdge(2, 3)

g.addEdge(3, 3)

print("Following is Depth First Traversal (starting from vertex 2)")

g.DFS(2)

**Output:-**

****