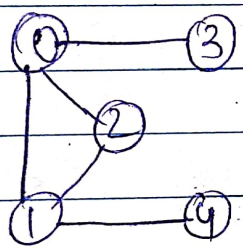
18. To implement Depth First Search, Breadth First search traversals on a graph

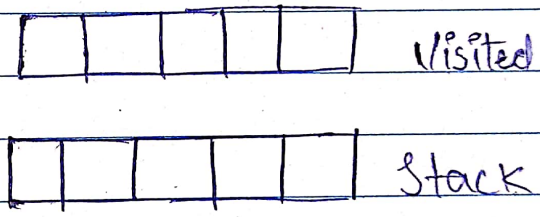
A standard Depth First search implementation puts each vertex of the graph into one of two categories
→ Visited
→ Not visited
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

Breadth First search is a recursive algorithm for searching all the vertices of a graph or tree data structure. A standard Breadth First Search implementation puts each vertex of the graph into one of two categories.
→ Visited
→ Not Visited

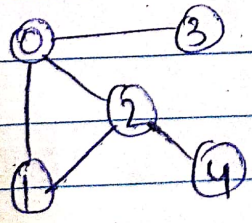
Example :-

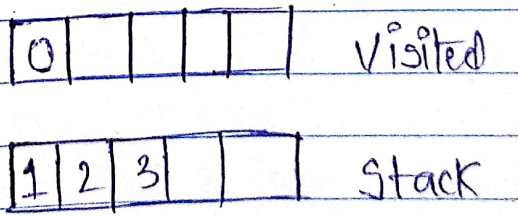
We use an undirected graph with 5 vertices





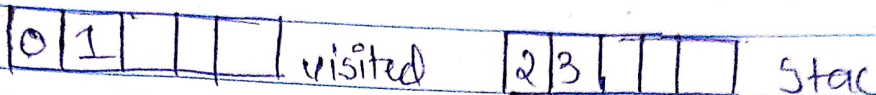
We start from vertex 0, the DFS algorithm starts by putting it in the visited list and putting all its adjacent vertices in stack



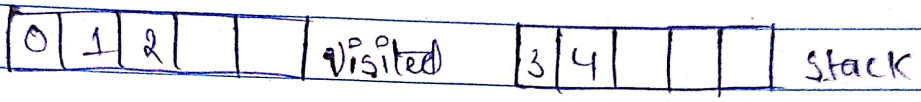


Next we visit the element at the top of the stack ie

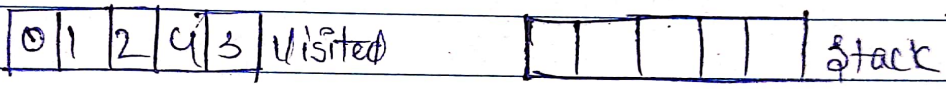
1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead



Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.

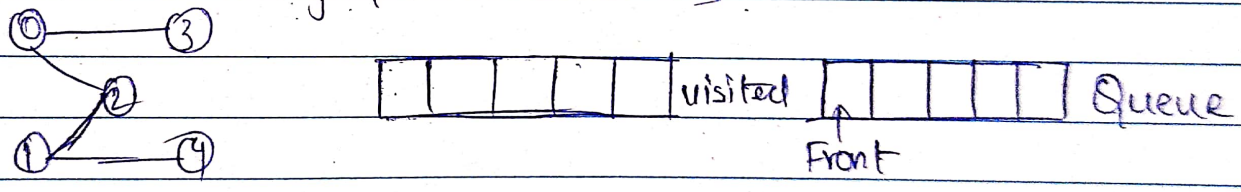


After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First traversal.

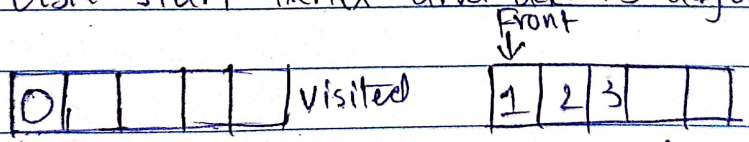


Breadth First search :-

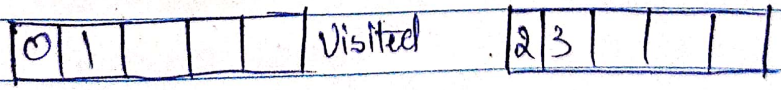
undirected graph with vertices



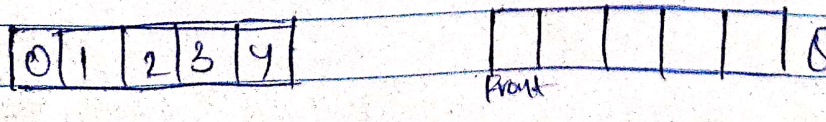
Visit start vertex and add its adjacent vertices to Queue



Visit the first node (neighbour) of start node 0, which is 1



Visit 2 which was added to queue earlier to add, its neighbour



Code: (DFS)

```
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[v]=True
        print v,
        for i in self.graph[v]
            if visited[i]==False:
                self.DFSutil(i,visited)
    def DFS(self):
        V=len(self.graph)
        visited = [False]*(V)
        for i in range(V):
            if visited[i]==False:
                self.DFSutil(i,visited)
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", g.DFS(1)
```

(BFS)

```
graph = {
    '5': ['3', '7'],
    '3': ['2', '4'],
    '7': ['8']
    '2': []
    '4': ['8']
    '8': []
}
visited = []
queue = []
def bfs(visited, graph, node):
    visited.append(node)
    queue.append(node)
  while queue:
    m = queue.pop(0)
    print(m, end=" ")
    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)

print("Following is BFS")
bfs(visited, graph, '5')
```

Output:-

following is Depth First Traversal
2 0 3 1
Following Is BFS
5 3 7 2 4 8