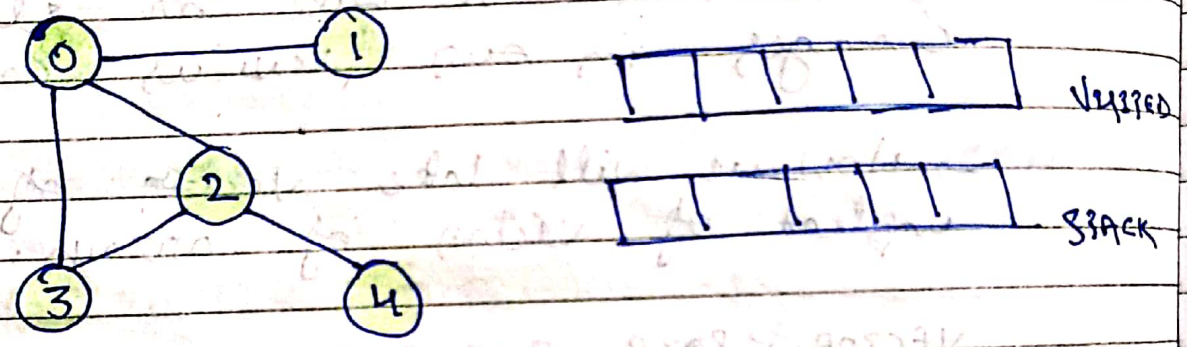
DEPTH FIRST SEARCH (DFS)

Depth-First Search is an algorithm for traversing or searching tree or graph data structure. The algorithm starts at the root node. (selecting some arbitrary node as the root node in case of a graph) and explores as

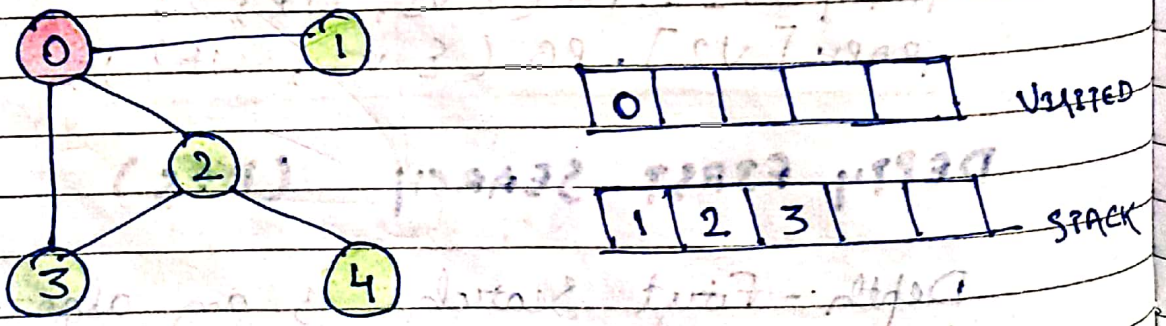
for as possible along each branch before backtracking.

Lets understand the working of Depth First Search with the help of the following illustration.

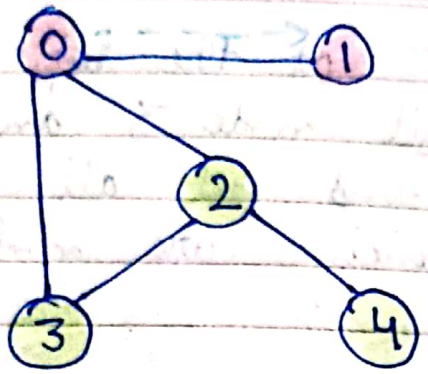
STEP→1 Initially stack and visited arrays are empty.



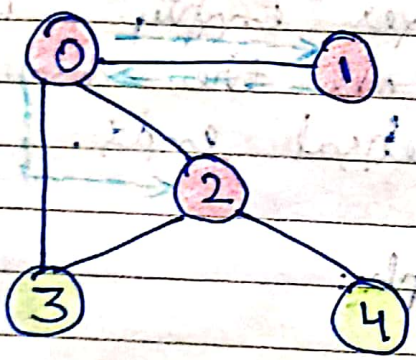
STEP→2 Visit 0 and put its adjacent nodes which are not visited yet into the stack.

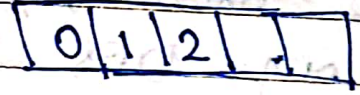


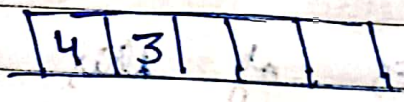
STEP→3 Now, Node 1 at the top of the stack, so visit node 1 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.



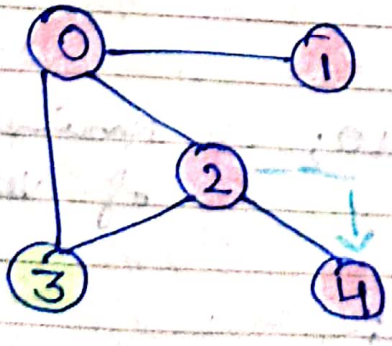
STEP→4 Now, Node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes which are not visited (i.e, 3,4) in the stack.

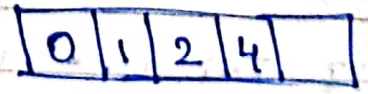




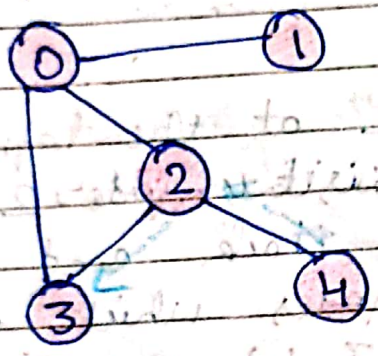


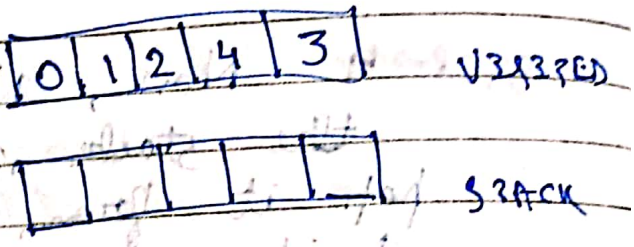
STEP→5 Now, Node 4 at the top of the stack, so visit node 4 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.





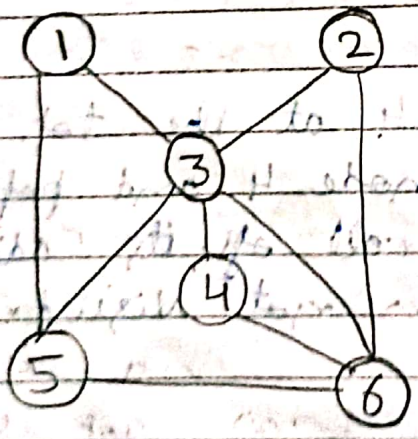
STEP→6 Now, Node 3 at the top of the stack, so visit node 3 and pop it from the stack and put all of its adjacent nodes which are not visited in the stack.





Now, stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

DFS of this graph:



CODE

```
CONST INT N = 1e5 + 10;  // Maximum capacity of vertices
// -> Adjacency list
VECTOR <INT> G[N];
BOOL VIS[N];  // Visited array
```

```
VOID DFS (INT VERTEX) // vo vertex jha se
{                          dfs start krni hai
  -> Take action on vertex after entering
  the vertex
  COUT << VERTEX << ENDL; // just to
  visualize what's happening.
  VIS [VERTEX] = true;
  FOR (INT CHILD : G [VERTEX])
  {  -> Take action on child before entering
                                    child node.
     COUT << "PARENT: " << VERTEX <<
     ", CHILD: " << CHILD << "\n";
     // just to visualize what's happening.
     IF (VIS [CHILD]) CONTINUE; // skipping
                                 visited node
     DFS (CHILD);
     -> Take action on child after
     exiting child node.
  }
  -> Take action on vertex after exiting
  the vertex.

  -> Time Complexity : O (V+E) as
  Recursive function will be called
  V (vertex) times, and in each
  call there is a for loop E (edges)
  times.
}

INT MAIN ()
{
```

```
int n, m;
cin >> n >> m;

→ creating Adjacency list of graph
for (int i=0; i<=9; i++)
{
    int v1, v2;
    cin >> v1 >> v2;
    g[v1].pb(v2);
    g[v2].pb(v1);
}

dfs(1); // Calling DFS function.
return 0;
}
```

Input:

```
6 9
1 3
1 5
3 5
3 4
3 6
3 2
2 6
4 6
5 6
```

Output:

1

PARENT: 1 , CHILD: 3

3

PARENT: 3 , CHILD: 1

PARENT : 3 , CHILD: 5

5

PARENT: 5 , CHILD: 1

PARENT: 5 , CHILD: 3

PARENT: 5 , CHILD: 6

6

PARENT: 6 , CHILD: 3

PARENT: 6 , CHILD: 2

2

PARENT: 2 , CHILD: 3

PARENT: 2 , CHILD: 6

PARENT: 6 , CHILD: 4

4

PARENT: 4 , CHILD: 3

PARENT : 4 , CHILD: 6

PARENT: 6 , CHILD: 5

PARENT : 3 , CHILD : 4

PARENT: 3 , CHILD : 6

PARENT: 3 , CHILD : 2

PARENT: 1 , CHILD : 5