# Depth First Search Algorithm

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

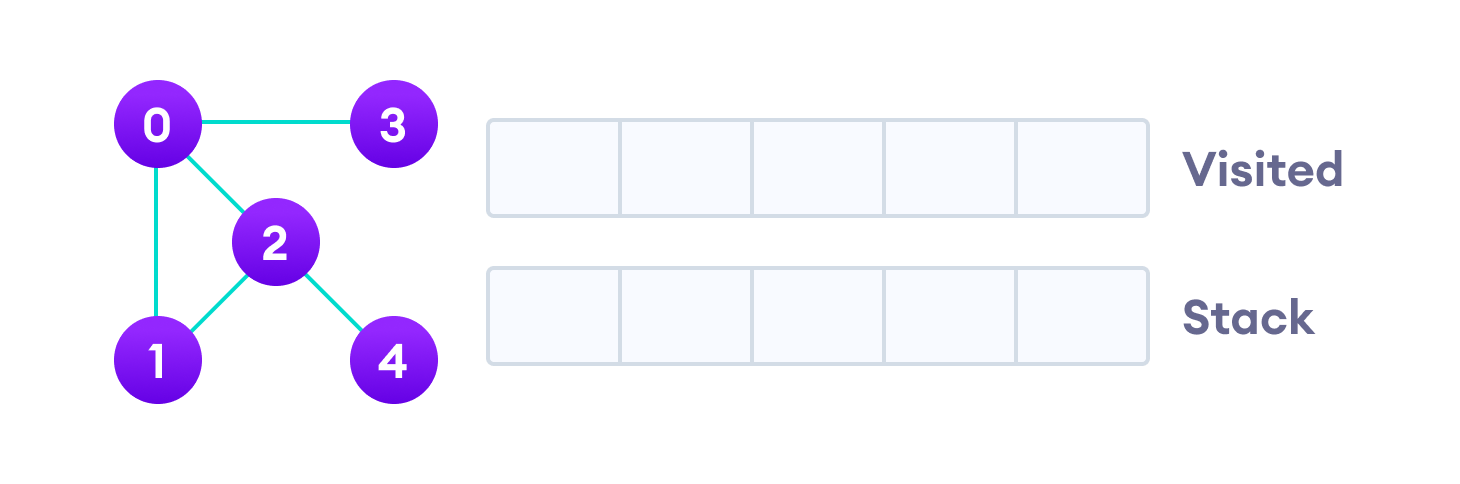
The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

# The DFS algorithm works as follows:

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

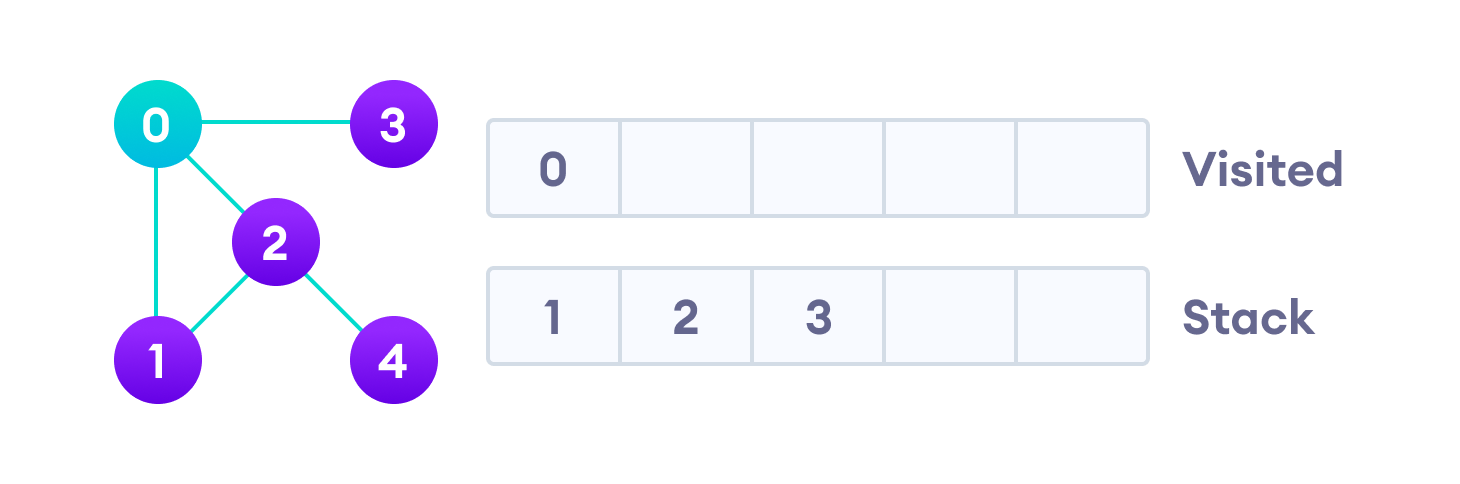
# Depth First Search Example

Let's see how the Depth First Search algorithm works with an example. We use an undirected graph with 5 vertices.



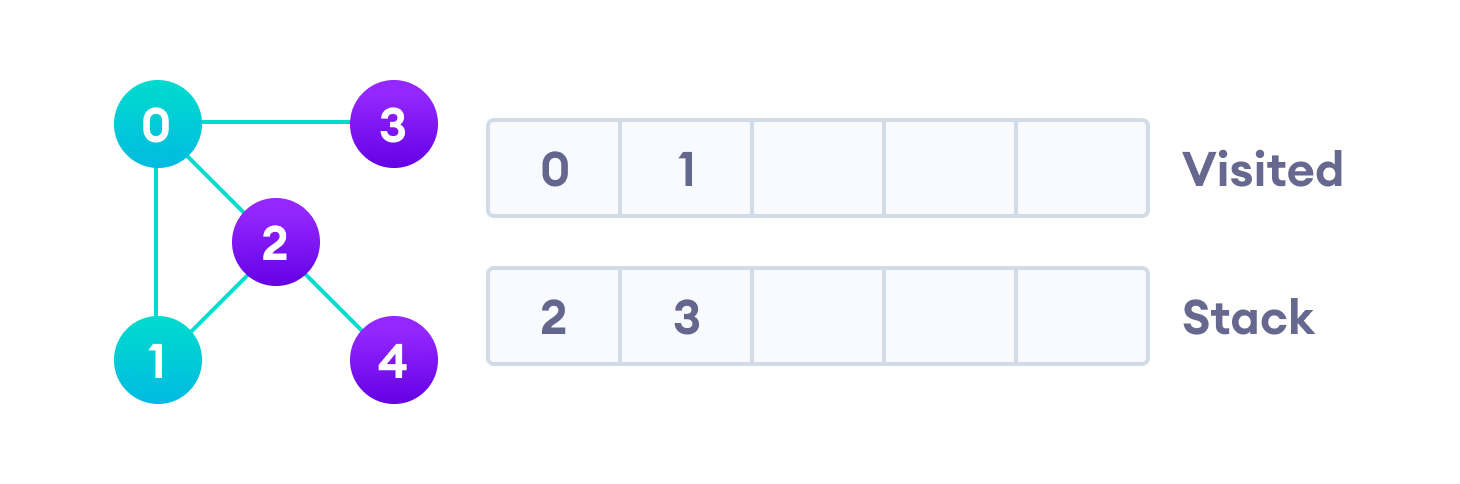
Undirected graph with 5 vertices

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



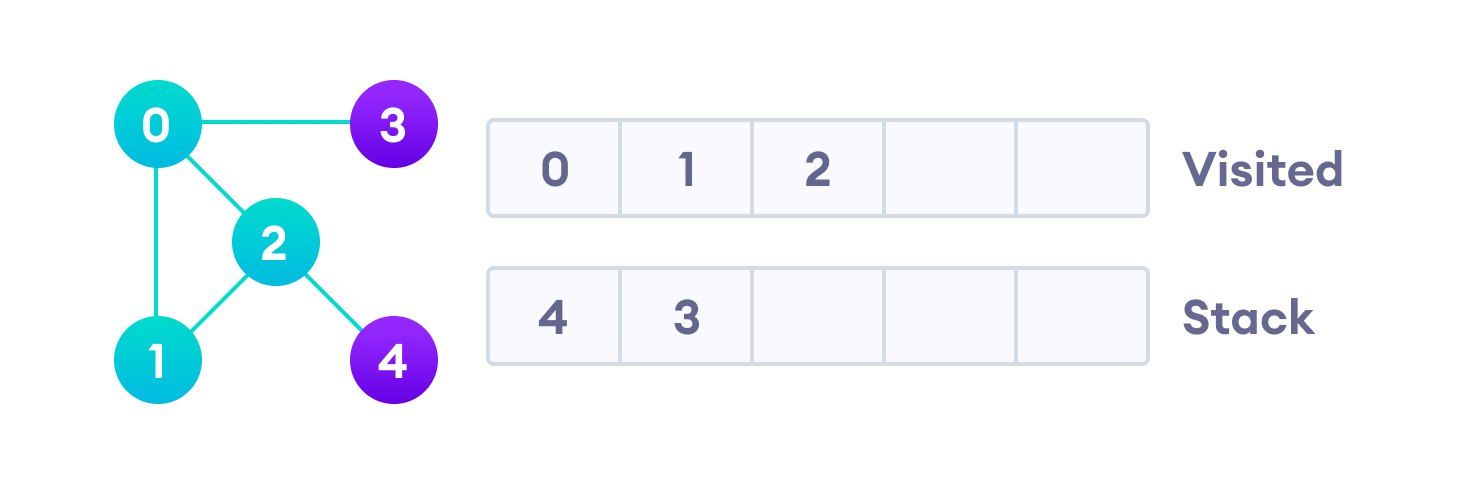
Visit the element and put it in the visited list

Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

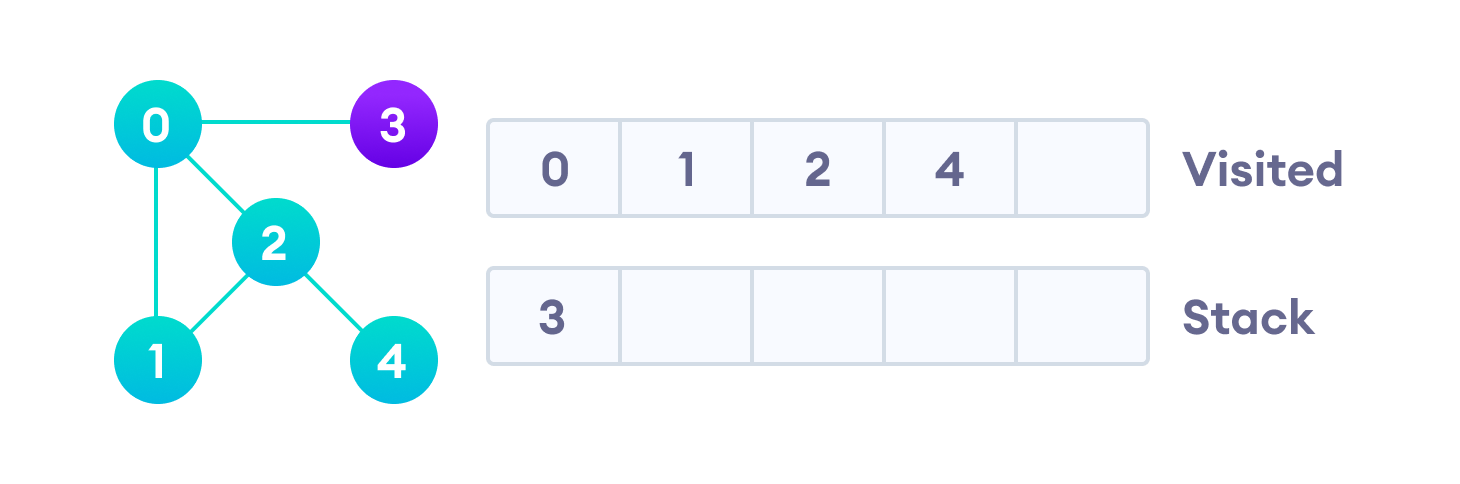


Visit the element at the top of stack

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

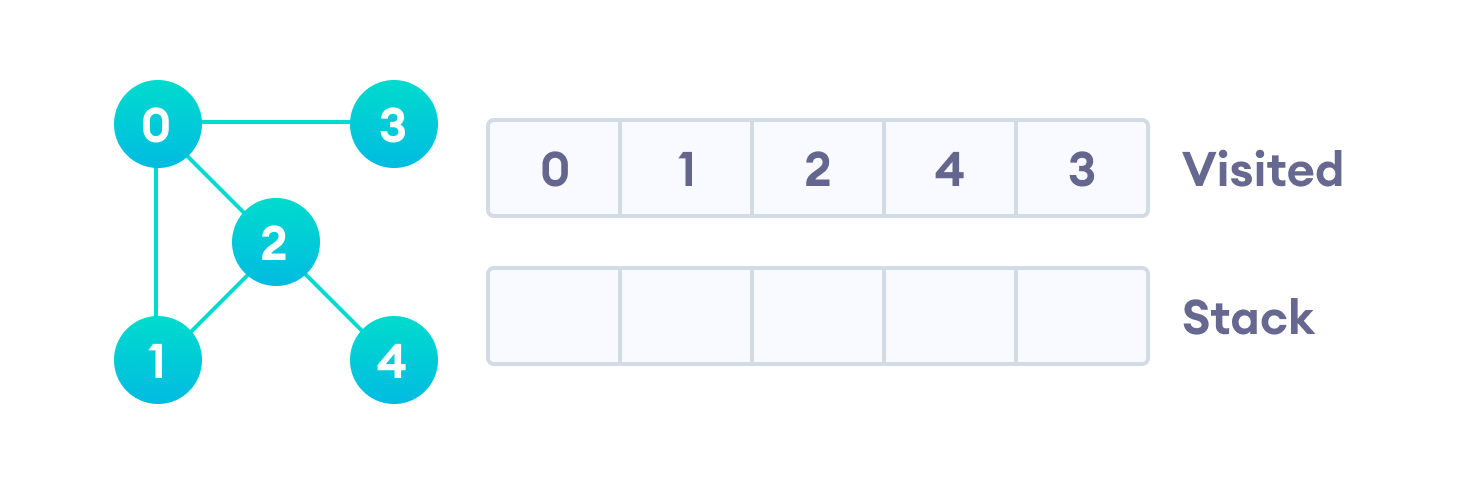


Vertex 2 has an unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.



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 of the graph.



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

## DFS Pseudocode (recursive implementation)

The pseudocode for DFS is shown below. In the init() function, notice that we run the DFS function on every node. This is because the graph might have two different disconnected parts so to make sure that we cover every vertex, we can also run the DFS algorithm on every node.

DFS(G, u)

u.visited = true

for each v ∈ G.Adj[u]

if v.visited == false

DFS(G,v)

init() {

For each u ∈ G

u.visited = false

For each u ∈ G

DFS(G, u)

}

## DFS Implementation in Python, Java and C/C++

The code for the Depth First Search Algorithm with an example is shown below. The code has been simplified so that we can focus on the algorithm rather than other details.

// DFS algorithm in C++

#include <iostream>

#include <list>

using namespace std;

class Graph {

int numVertices;

list<int> \*adjLists;

bool \*visited;

public:

Graph(int V);

void addEdge(int src, int dest);

void DFS(int vertex);

};

// Initialize graph

Graph::Graph(int vertices) {

numVertices = vertices;

adjLists = new list<int>[vertices];

visited = new bool[vertices];

}

// Add edges

void Graph::addEdge(int src, int dest) {

adjLists[src].push\_front(dest);

}

// DFS algorithm

void Graph::DFS(int vertex) {

visited[vertex] = true;

list<int> adjList = adjLists[vertex];

cout << vertex << " ";

list<int>::iterator i;

for (i = adjList.begin(); i != adjList.end(); ++i)

if (!visited[\*i])

DFS(\*i);

}

int main() {

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 3);

g.DFS(2);

return 0;

}