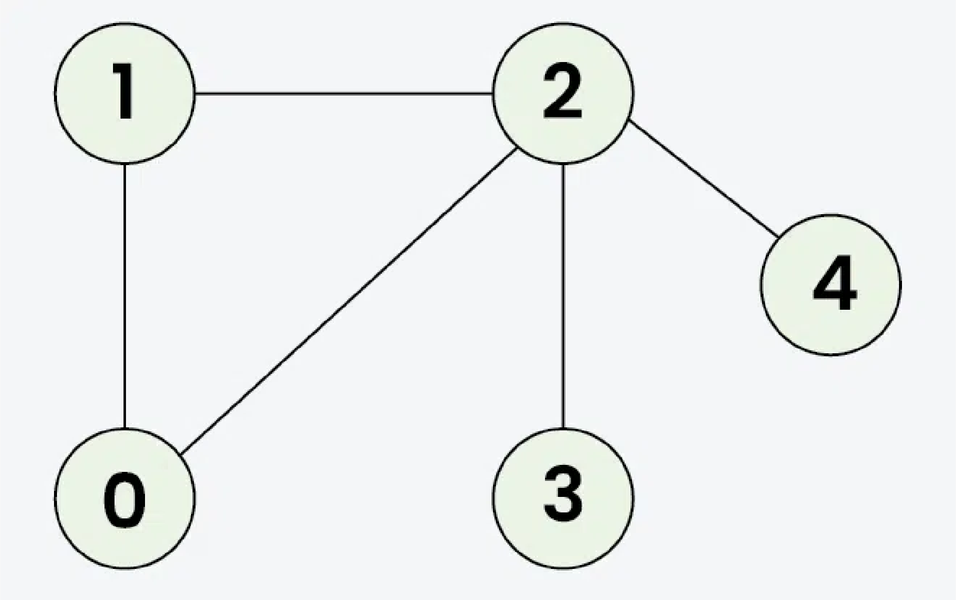
Depth First Search or DFS for a Graph

Last Updated : 29 Mar, 2025

In Depth First Search (or DFS) for a graph, we traverse all adjacent vertices one by one. When we traverse an adjacent vertex, we completely finish the traversal of all vertices reachable through that adjacent vertex. This is similar to a tree, where we first completely traverse the left subtree and then move to the right subtree. The key difference is that, unlike trees, graphs may contain cycles (a node may be visited more than once). To avoid processing a node multiple times, we use a boolean visited array.

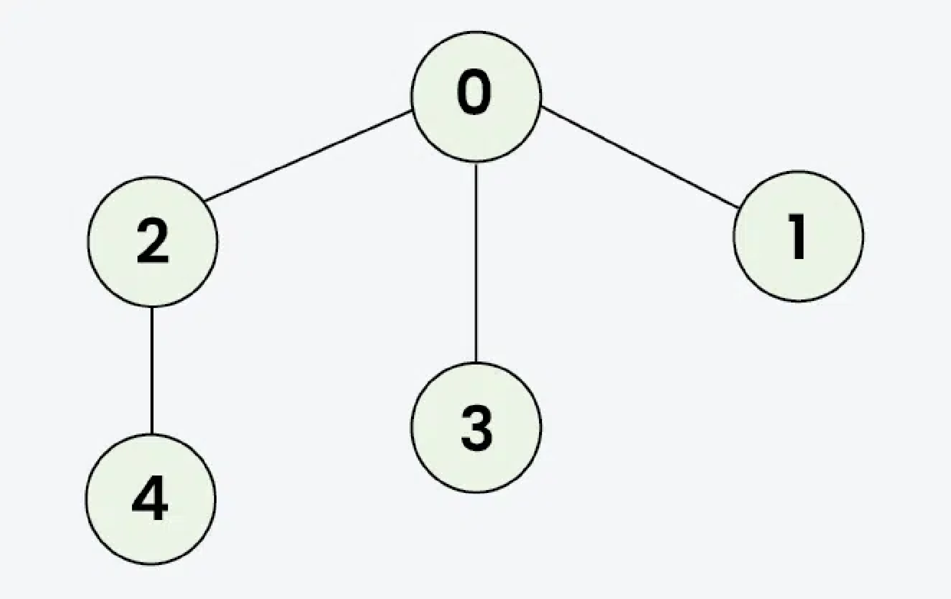
Example:

> ***Note :*** *There can be multiple DFS traversals of a graph according to the order in which we pick adjacent vertices. Here we pick vertices as per the insertion order.*
>
> ***Input:*** *adj = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]*
>
> 
>
>
> ***Output:*** *[0 1 2 3 4]*
> ***Explanation:*** *The source vertex s is 0. We visit it first, then we visit an adjacent.*
> *Start at 0: Mark as visited. Output: 0*
> *Move to 1: Mark as visited. Output: 1*
> *Move to 2: Mark as visited. Output: 2*
> *Move to 3: Mark as visited. Output: 3 (backtrack to 2)*

Move to 4: Mark as visited. Output: 4 (backtrack to 2, then backtrack to 1, then to 0)

Not that there can be more than one DFS Traversals of a Graph. For example, after 1, we may pick adjacent 2 instead of 0 and get a different DFS. Here we pick in the insertion order.

***Input:** [[2,3,1], [0], [0,4], [0], [2]]*



***Output:** [0 2 4 3 1]*
***Explanation:** DFS Steps:*

Start at 0: Mark as visited. Output: 0
Move to 2: Mark as visited. Output: 2
Move to 4: Mark as visited. Output: 4 (backtrack to 2, then backtrack to 0)
Move to 3: Mark as visited. Output: 3 (backtrack to 0)
Move to 1: Mark as visited. Output: 1 (backtrack to 0)

Table of Content

DFS from a Given Source of Undirected Graph:

The algorithm starts from a given source and explores all reachable vertices from the given source. It is similar to Preorder Tree Traversal where we visit

> *the root, then recur for its children. In a graph, there might be loops. So we use an extra visited array to make sure that we do not process a vertex again.*

Let us understand the working of **Depth First Search** with the help of the following **Illustration:** for the **source as 0**.

2 / 10

C++ Java Python C# JavaScript

```cpp
#include <bits/stdc++.h>
using namespace std;

// Recursive function for DFS traversal
void dfsRec(vector<vector<int>> &adj, vector<bool> &visited, int s, vector<int> &res)
{

    visited[s] = true;

    res.push_back(s);

    // Recursively visit all adjacent vertices
    // that are not visited yet
    for (int i : adj[s])
        if (visited[i] == false)
```

```
            dfsRec(adj, visited, i, res);
}

// Main DFS function that initializes the visited array
// and call DFSRec
vector<int> DFS(vector<vector<int>> &adj)
{
    vector<bool> visited(adj.size(), false);
    vector<int> res;
    dfsRec(adj, visited, 0, res);
    return res;
}

// To add an edge in an undirected graph
void addEdge(vector<vector<int>> &adj, int s, int t)
{
    adj[s].push_back(t);
    adj[t].push_back(s);
}

int main()
{
    int V = 5;
    vector<vector<int>> adj(V);

    // Add edges
    vector<vector<int>> edges = {{1, 2}, {1, 0}, {2, 0}, {2, 3}, {2, 4}};
    for (auto &e : edges)
        addEdge(adj, e[0], e[1]);

    // Starting vertex for DFS
    vector<int> res = DFS(adj); // Perform DFS starting from the source verte 0;

    for (int i = 0; i < V; i++)
        cout << res[i] << " ";
}
```

Output

```
0 1 2 3 4
```

Time complexity: O(V + E), where V is the number of vertices and E is the number of edges in the graph.
Auxiliary Space: O(V + E), since an extra visited array of size V is required, And stack size for recursive calls to dfsRec function.

Please refer Complexity Analysis of Depth First Search for details.

DFS for Complete Traversal of Disconnected Undirected Graph

> *The above implementation takes a source as an input and prints only those vertices that are reachable from the source and would not print all vertices in case of disconnected graph. Let us now talk about the algorithm that prints all vertices without any source and the graph maybe disconnected.*

The idea is simple, instead of calling DFS for a single vertex, we call the above implemented DFS for all all non-visited vertices one by one.

C++ Java Python C# JavaScript

```cpp
#include <bits/stdc++.h>
using namespace std;

void addEdge(vector<vector<int>> &adj, int s, int t)
{
    adj[s].push_back(t);
    adj[t].push_back(s);
}

// Recursive function for DFS traversal
void dfsRec(vector<vector<int>> &adj, vector<bool> &visited, int s, vector<int> &res)
{
    // Mark the current vertex as visited
    visited[s] = true;

    res.push_back(s);

    // Recursively visit all adjacent vertices that are not visited yet
    for (int i : adj[s])
        if (visited[i] == false)
            dfsRec(adj, visited, i, res);
}

// Main DFS function to perform DFS for the entire graph
vector<int> DFS(vector<vector<int>> &adj)
{
    vector<bool> visited(adj.size(), false);
    vector<int> res;
    // Loop through all vertices to handle disconnected graph
    for (int i = 0; i < adj.size(); i++)
    {
        if (visited[i] == false)
        {
            // If vertex i has not been visited,
            // perform DFS from it
            dfsRec(adj, visited, i, res);
        }
    }

    return res;
}

int main()
{
    int V = 6;
    // Create an adjacency list for the graph
    vector<vector<int>> adj(V);

    // Define the edges of the graph
    vector<vector<int>> edges = {{1, 2}, {2, 0}, {0, 3}, {4, 5}};

    // Populate the adjacency list with edges
    for (auto &e : edges)
        addEdge(adj, e[0], e[1]);
    vector<int> res = DFS(adj);

    for (auto it : res)
        cout << it << " ";
    return 0;
}
```

Output

```
0 2 1 3 4 5
```

Time complexity: O(V + E). Note that the time complexity is same here because we visit every vertex at most once and every edge is traversed at most once (in directed) and twice in undirected.

Auxiliary Space: O(V + E), since an extra visited array of size V is required, And stack size for recursive calls to dfsRec function.

Related Articles:

- Depth First Search or DFS on Directed Graph
- Breadth First Search or BFS for a Graph

Depth First Search

Visit Course ↗

Comment

More info

Advertise with us

Next Article >

C Program for Depth First Search or DFS for a Graph

Similar Reads

Depth First Search or DFS for a Graph

In Depth First Search (or DFS) for a graph, we traverse all adjacent vertices one by one. When we traverse an adjacent vertex, we completely finish the traversal of all vertices reachable through that...

13 min read

DFS in different language

Iterative Depth First Traversal of Graph

Given a directed Graph, the task is to perform Depth First Search of the given graph. Note: Start DFS from node 0, and traverse the nodes in the same order as adjacency list. Note : There can be multiple DFS...

🕓 10 min read

Applications, Advantages and Disadvantages of Depth First Search (DFS)

Depth First Search is a widely used algorithm for traversing a graph. Here we have discussed some applications, advantages, and disadvantages of the algorithm. Applications of Depth First Search:1....

🕓 4 min read

Difference between BFS and DFS

Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental algorithms used for traversing or searching graphs and trees. This article covers the basic difference between Breadth-First...

🕓 2 min read

Depth First Search or DFS for disconnected Graph

Given a Disconnected Graph, the task is to implement DFS or Depth First Search Algorithm for this Disconnected Graph. Example: Input: Output: 0 1 2 3 Algorithm for DFS on Disconnected Graph:In the...

🕓 7 min read

Printing pre and post visited times in DFS of a graph

Depth First Search (DFS) marks all the vertices of a graph as visited. So for making DFS useful, some additional information can also be stored. For instance, the order in which the vertices are visited while...

🕓 8 min read

Tree, Back, Edge and Cross Edges in DFS of Graph

Given a directed graph, the task is to identify tree, forward, back and cross edges present in the graph. Note: There can be multiple answers. Example: Input: Graph Output:Tree Edges: 1->2, 2->4, 4->6, 1->3,...

🕓 9 min read

Transitive Closure of a Graph using DFS

Given a directed graph, find out if a vertex v is reachable from another vertex u for all vertex pairs (u, v) in the given graph. Here reachable means that there is a path from vertex u to v. The reach-ability matrix is...

🕓 8 min read

Variations of DFS implementations