GeeksforGeeks

A computer science portal for geeks

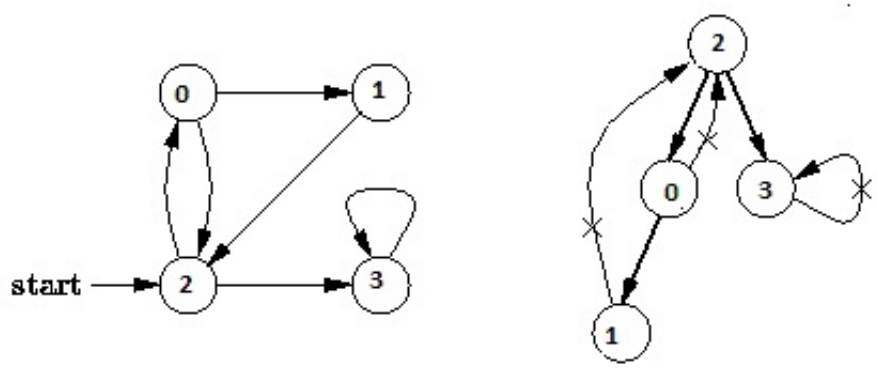
GeeksQuiz

- Home
- Algorithms
- DS
- GATE
- Interview Corner
- Q&A
- C
- C++
- Java
- Books
- Contribute
- Contests
- Jobs

Array
Bit Magic
C/C++
Articles
GFacts
Linked List
MCQ
Misc
Output
String
Tree
Graph

Depth First Traversal for a Graph

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.
For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. Depth First Traversal of the following graph is 2, 0, 1, 3



See this post for all applications of Depth First Traversal.
Following is C++ implementation of simple Depth First Traversal. The implementation uses adjacency list representation of graphs. STL's list container is used to store lists of adjacent nodes.

```
#include<iostream>
#include <list>

using namespace std;

// Graph class represents a directed graph using adjacency list representatio
class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // Pointer to an array containing adjacency lists
    void DFSUtil(int v, bool visited[]);  // A function used by DFS
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // function to add an edge to graph
    void DFS(int v);    // DFS traversal of the vertices reachable from v
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for(i = adj[v].begin(); i != adj[v].end(); ++i)
```

```
        if(!visited[*i])
            DFSUtil(*i, visited);
}

// DFS traversal of the vertices reachable from v. It uses recursive DFSUtil(
void Graph::DFS(int v)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to print DFS traversal
    DFSUtil(v, visited);
}

int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal (starting from vertex 2) \n";
    g.DFS(2);

    return 0;
}
```

Output:

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3
```

Note that the above code traverses only the vertices reachable from a given source vertex. All the vertices may not be reachable from a given vertex (example Disconnected graph). To do complete DFS traversal of such graphs, we must call DFSUtil() for every vertex. Also, before calling DFSUtil(), we should check if it is already printed by some other call of DFSUtil(). Following implementation does the complete graph traversal even if the nodes are unreachable. The differences from the above code are highlighted in the below code.

```
#include<iostream>
#include <list>
using namespace std;

class Graph
{
    int V;    // No. of vertices
```

```
    list<int> *adj;    // Pointer to an array containing adjacency lists
    void DFSUtil(int v, bool visited[]);  // A function used by DFS
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // function to add an edge to graph
    void DFS();    // prints DFS traversal of the complete graph
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v's list.
}

void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i;
    for(i = adj[v].begin(); i != adj[v].end(); ++i)
        if(!visited[*i])
            DFSUtil(*i, visited);
}

// The function to do DFS traversal. It uses recursive DFSUtil()
void Graph::DFS()
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];
    for(int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to print DFS traversal
    // starting from all vertices one by one
    for(int i = 0; i < V; i++)
        if(visited[i] == false)
            DFSUtil(i, visited);
}

int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
```

```
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal (starting from vertex 0) \n";
    g.DFS();

    return 0;
}
```

Time Complexity: O(V+E) where V is number of vertices in the graph and E is number of edges in the graph.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Related Topics:

- Assign directions to edges so that the directed graph remains acyclic
- K Centers Problem | Set 1 (Greedy Approximate Algorithm)
- Find the minimum cost to reach destination using a train
- Applications of Breadth First Traversal
- Optimal read list for given number of days
- Print all paths from a given source to a destination
- Minimize Cash Flow among a given set of friends who have borrowed money from each other
- Boggle (Find all possible words in a board of characters)

Tags: Graph

Writing code in comment? Please use **ideone.com** and share the link here.

Recommend 2 Share Sort by Newest

Join the discussion…

vikas mangal • 18 days ago

I tried to print cross edges and forward edges along with the back edges in the following code.

http://ideone.com/18xfGW

Please have a look and tell me where it can fail.

The logic I used is:

If we visit a node already visited and we have not yet departed from that node, than it makes a back edge.

Else, there are two possibilities – cross edge or forward edge. In both these cases we visit a node from which we have alredy departed. So here, we can distinguish between them using the arrival time. If it is a forward edge (going from ancestor to descendent, u -> v), arrival time of u will be less v and if it is cross edge, arrival time of u will be greater than v.

Reply • Share ›

Andy Toh • 23 days ago

Here is my compile-time solution:

http://ideone.com/4hh65L

Among the various outputs tested, 2 0 1 3 is the one that matches the run-time output given above (starting at vertex 2).

Reply • Share ›

Krishna • 2 months ago

This blog's article is better explained with a simple C code... https://theoryofprogramming.wo...
....

1 Reply • Share ›

Anushkha Singh • 3 months ago

The following link is really useful:
Depth First Search

1 Reply • Share ›

Chanakya Nani • 4 months ago

Java implementation (adjacency list)
http://ideone.com/U3pGNE

3 ^ | v • Reply • Share ›

Guest • 5 months ago

We can easily do this by depature time of the vertex!

http://ideone.com/078vDs

1 ^ | v • Reply • Share ›

Leen • 6 months ago

This is really theoretical. In real applications, nodes often don't have a unique number. How could we store the visit information? Maybe we can use hash table, but it's inefficient.

^ | v • Reply • Share ›

Anton Zuykov ↗ Leen • 5 months ago

if vertices don't have unique numbers, then how do you traverse and know that that's the vertex you haven't visited yet? That doesn't make sense!

Vertices must have some kind of identification for DFS to work.

At least, your nodes will have a unique number by the virtue of being stored at different locations in memory.

5 ^ | v • Reply • Share ›

vidit • 6 months ago

should the dfsutil function use a pointer to array visited otherwise there will be no link between visited array in dfs function and dfs util function and some nodes may be printed more than once

^ | v • Reply • Share ›

RK ↗ vidit • 3 months ago

You are right, and that's exactly what the coder has done in this code. Variable 'visited' is a pointer to the 'visited array'.

Did you mean to ask if we should use a pointer to a pointer? That's not necessary as we are not changing the address pointed by the pointer.

^ | v • Reply • Share ›

Kaushal Pranav • 7 months ago

If they ask only for depth first traversal and give the initial vertex and if the graph is discontinuous then should we cover all the vertices or should we only go for the part in which the vertex is present

^ | v • Reply • Share ›

Anurag Singh ➚ Kaushal Pranav • 7 months ago

In case of any traversal, all vertices should be traversed (unless explicitly specified not to do that)

1 ^ | v • Reply • Share ›

Kaushal Pranav ➚ Anurag Singh • 7 months ago

In Mark Allen Weiss book and also our sir had told that DFT is only a part of the graph if the graph is discontinuous . But in Adam Drozdek book it was given as Depth First Search (not traversal) and covered all the vertices . Can u give a clear explanation of what all these mean?

^ | v • Reply • Share ›

Anurag Singh ➚ Kaushal Pranav • 7 months ago

Based on what you said above:
In a disconnected graph, if we do a DFS, we traverse from some node and find all the connected nodes. This can be thought of as one traversal. Then you start your traversal from any other unvisited node and keep repeating until all nodes are visited. So we can think that if a disconnected graph has "N" connected components then there will be "N" DFT in one DFS.

But most of the time, DFT and DFS mean same thing (just cover ALL vertices) and they are used interchangeably.

^ | v • Reply • Share ›

Sameer Agarwal • 7 months ago

in second program
cout << "Following is Depth First Traversal (starting from vertex 2) \n"

cout statement is wrong its not from vertex 2

^ | v • Reply • Share ›

Anurag Singh ➚ Sameer Agarwal • 7 months ago

Yes. It starts with 0, not 2. Will be corrected soon. Thanks !!

^ | v • Reply • Share ›

Pilos • 8 months ago

why have you used ++i instead of i++ ?

^ | v • Reply • Share ›

codingsage ➚ Pilos • 8 months ago

They dont matter

^ | v • Reply • Share ›

Anton Zuykov ➚ codingsage • 5 months ago

they do matter.

++i - increments value first and then you read it.
i++ -- reads value first and then it gets incremented (so the value you get and the value that is there - are different).

Big difference in some cases.

1 ^ | v • Reply • Share ›

Vãîbhåv Joshî • a year ago

JAVA implantation For DFS....

http://ideone.com/2fZ4K2

1 ^ | v • Reply • Share ›

Sriram Ganesh • a year ago

Here is an easy JAVA implementation of the same code using ArrayList..:-)
http://ideone.com/koM1t8

^ | v • Reply • Share ›

gansai ➚ Sriram Ganesh • 4 months ago

@Sriram Ganesh:

When you initialize a boolean array in java, you dont need to initialize it to false, by default, it will be initialized to false.

1 ^ | v • Reply • Share ›

Sriram Ganesh ➚ gansai • 4 months ago

Yes that is right I got carried away to deliver it quick.

^ | v • Reply • Share ›

samthebest • a year ago

if instead of adjacency list representation ,matrix represtation is used ...
will the time complexity will be differentt or not ??

3 ^ | v • Reply • Share ›

Vivek ➚ samthebest • 10 months ago

here with adjacency list the time complexity is O(V+E) since each vertex and each edge is visited once. But with adjacency matrix with n vertices , you have to build a matrix of size nxn. Performing DFS in it would result in O(n^2) time.

^ | v • Reply • Share ›

Guest • a year ago

```
#include<stdio.h>
#include<stdlib.h>
#include<stdbool.h>
#define MAX 50
struct node
{
int data;
struct node *e;
bool visited;
}*nodes[6];

struct node *new_node(int data)
{
struct node *new=malloc(sizeof(struct node));
new->data=data;
new->e=NULL;
new->visited=0;
return new;
```

see more

• Reply • Share ›

Himanshu Dagar • a year ago

Code which shows different different output when in graph no node is connected to anyone. can refer to below link

http://ideone.com/sBmjrv

• Reply • Share ›

Well • a year ago

How to implement it in NON-recursive approach?

• Reply • Share ›

samthbest ➚ Well • a year ago

use stack

• Reply • Share ›

rahuldey85 ➚ samthbest • 10 months ago

thats simulating recursion. recursion uses stacks behind the scenes.

• Reply • Share ›

khushal • 2 years ago

cyclic graph won't have topological order. your code doesn't say a word if graph is cyclic

Alien • 2 years ago

```
/*
C program for DFS traversal of a graph where graph is presented using Adjacency Matrix
*/
#include<stdio.h>
#define V 5
void dfsUtil(int graph[V][V], int visited[], int v)
{
visited[v] = 1;
printf("\nvertex: %d",v);
int i;
// DFS search for each connected component
```


Alien • 2 years ago

```
/*
C program for DFS traversal of a graph where graph is presented using Adjacency list
*/
#include <stdio.h>
#include <stdlib.h>
// A structure to represent an adjacency list node
struct AdjListNode
{
int vertex_num;
struct AdjListNode *next;
};
```

see more

• Reply • Share ›

Vijay Apurva • 2 years ago

here is the code in C

```
#include<stdio.h>
#include<stdlib.h>

struct gnode{
int data ;
struct gnode * next ;
};

struct adnode{
struct gnode * top ;
};

struct graph{
int v;
struct adnode * array ;
};
```

---

see more

4 • Reply • Share ›

Dhruv Bansal • 2 years ago

@jayanta you have just ensured that output is printed from different starting point but what about making sure that whole graph is printed when it is a disconnected graph?
i think 2nd code is for printing whole graph but in your different connected component will be printed.

• Reply • Share ›

Mohanad Ibrahim Mahasneh • 2 years ago

just c++

1 • Reply • Share ›

avanee • 2 years ago

```
//program for dfs
#include
#include
using namespace std;
int edgei_j[100][100],n;
int visited[1000];
```

```
int visited[1000];
void dfs(int);
stack stk;
int main()
{

int i,j,m,tops;
scanf("%d %d",&n,&m);
for(i=0;i<m;i++)
{
scanf("%d %d",&i,&j);
edgei_j[i][j]=1;
}
```

see more

• Reply • Share ›

avanee • 2 years ago

```
//simple imlementation for dfs
#include
#include
using namespace std;
int edgei_j[100][100],n;
int visited[1000];
void dfs(int);
stack stk;
int main()
{

int i,j,m,tops;
scanf("%d %d",&n,&m);
for(i=0;i<m;i++)
{
scanf("%d %d",&i,&j);
edgei_j[i][j]=1;
}
```

see more

1 • Reply • Share ›

DAKSHESH avanee • 13 days ago

IT IS FACK

• Reply • Share ›

amitmac2 • 2 years ago

```
 #include<stdio.h>
#include<stdlib.h>
#include<conio.h>
int time;
struct node
{
        int color;
        int parent;
        int key;
    int dest;
    struct node* next;  //next pointer
        struct node* head;      //head pointer
};
struct graph
{
    int v;      //number of vertices
    struct node* array; //array of all vertices
};
```

see more

• Reply • Share ›

amitmac2 ➚ amitmac2 • 2 years ago

there is error in above program...

1 • Reply • Share ›

AAZ • 2 years ago

How is complexity O(V+E) ?
Don't we have one for loop inside DFSUtil Function call which is itself inside for loop ? Shouldn't it be O(VE) ?

1 • Reply • Share ›

Nabila SOhail • 2 years ago

bakwasssssss

1 • Reply • Share ›

Jayanta • 3 years ago

// Pasted the modified (correct) dfs code.

```
 #include<iostream>
#include <list>

using namespace std;
```

```
class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // Pointer to an array containing adjacency lists
    void DFSUtil(int v, bool visited[]);  // A function used by DFS
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // function to add an edge to graph
    void DFS();    // prints DFS traversal of the complete graph
};
```

see more

1 • Reply • Share ›

deep ➚ Jayanta • 3 years ago

@Jayanta you are right

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

kartik ➚ deep • 2 years ago

@Jayanta and @deep: could you please let us know the problem with the code given in above post? In below comment, you mentioned that the given code will print "0 1 2 3" for given graph. Isn't "0 1 2 3" correct output? Please let me know if my understanding is incorrect.

• Reply • Share ›

Jayanta • 3 years ago

The second code for visiting all nodes in DFS() is wrong as it does not set false to visited[v] ever.

For the following main() it will also print 0 1 2 3.

int main()
{
// Create a graph given in the above diagram
Graph g(4);
g.addEdge(0, 1);
g.addEdge(2, 3);

cout << "Following is Depth First Traversal (starting from vertex 2) \n";
g.DFS();

return 0;
}

• Reply • Share ›

Venki • 3 years ago

Iterative DFS, looking for an efficient implementation.

I know an explicit stack can be used. One way is, use explicit stack, and keep pushing all the adjacent nodes while exploring each path node in unexplored path. Following this approach, in the worst case, explicit stack will have all the adjacent nodes in the longest path of a node in graph.

Where as in recursive approach, the system stack will have max nodes proportional to the depth of the longest node.

Any thoughts? I am thinking to keep track of visited node and current exploring node in it's adjacent list. In other words, mitigating system stack functionality by storing stack frame explicitly.

• Reply • Share ›

Venki ➚ Venki • 3 years ago

Here is pseudo code,

```
dfs(source)
   stack.push(source)
   v = stack.top()

   while( stack not empty )
       stack.pop()
       visited[v] = true

       for( each x in G.ADJ(v) )
           if( !visited[x] )
               stack.push(x)

       v = stack.top()
       while( (stack not empty) AND (v is visited) )
           v = stack.top()
           stack.pop()
```

• Reply • Share ›

Sreenivas Doosa ➚ Venki • 2 years ago

The following is the code for DFS Iterative...

[sourcecode language="java"]

```
import java.util.ArrayList;
import java.util.Stack;

public class Graph {

private int V; // no of vertices
private ArrayList<ArrayList<Integer>> adj; // adjacency list

public Graph(int V) {

this.V = V;
this.adj = new ArrayList<ArrayList<Integer>>(V);
for(int i = 0; i < V; i++) {
this.adj.add(new ArrayList<Integer>());
}
}
```

see more

Reply • Share ›

Load more comments

-

Popular Posts

- All permutations of a given string
- Memory Layout of C Programs
- Understanding “extern” keyword in C
- Median of two sorted arrays
- Tree traversal without recursion and without stack!
- Structure Member Alignment, Padding and Data Packing
- Intersection point of two Linked Lists
- Lowest Common Ancestor in a BST.
- Check if a binary tree is BST or not
- Sorted Linked List to Balanced BST

- Follow @GeeksforGeeks

Recent Comments

- Mission Peace

  https://youtu.be/9mod_xRB-O0 Video on above...

  Dynamic Programming | Set 22 (Box Stacking Problem) · 3 hours ago

- Kartikeya Khatri

  we can also build a graph.. and count the...

  Count number of ways to reach a given score in a game · 4 hours ago

- Baba

  Great work dude :)

  Topological Sorting · 4 hours ago

- Amitesh Sinha

  Hi Vineel, shouldnt the answer for...

  Directi Interview Questions · 4 hours ago

- Rishi Verma

  Have a look :...

  Top 25 Interview Questions · 4 hours ago

- Rishi Verma

  http://www.geeksforgeeks.org/1...

  Top 25 Interview Questions · 4 hours ago