Depth First Search

Introduction to the Algorithm:

The DFS algorithm is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected.

This recursive nature of DFS can be implemented using stacks. The basic idea is as follows:

Pick a starting node and push all its adjacent nodes into a stack.

Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack.

Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

Algorithm:

DFS-iterative (G, s):

let S be stack

S.push( s )

mark s as visited.

while ( S is not empty):

//Pop a vertex from stack to visit next

v = S.top( )

S.pop( )

//Push all the neighbours of v in stack that are not visited

for all neighbours w of v in Graph G:

if w is not visited :

S.push( w )

mark w as visited

DFS-recursive(G, s):

mark s as visited

for all neighbours w of s in Graph G:

if w is not visited:

DFS-recursive(G, w)

A graph is said to be disconnected if it is not connected, i.e. if two nodes exist in the graph such that there is no edge in between those nodes. In an undirected graph, a connected component is a set of vertices in a graph that are linked to each other by paths.

Code:

# Python program to print DFS traversal from a

# given given graph

from collections import defaultdict

# This class represents a directed graph using

# adjacency list representation

class Graph:

# Constructor

def \_\_init\_\_(self):

# default dictionary to store graph

self.graph = defaultdict(list)

# function to add an edge to graph

def addEdge(self,u,v):

self.graph[u].append(v)

# A function used by DFS

def DFSUtil(self,v,visited):

# Mark the current node as visited and print it

visited[v]= True

print v,

# Recur for all the vertices adjacent to this vertex

for i in self.graph[v]:

if visited[i] == False:

self.DFSUtil(i, visited)

# The function to do DFS traversal. It uses

# recursive DFSUtil()

def DFS(self,v):

# Mark all the vertices as not visited

visited = [False]\*(len(self.graph))

# Call the recursive helper function to print

# DFS traversal

self.DFSUtil(v,visited)

# Driver code

# Create a graph given in the above diagram

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 DFS from (starting from vertex 2)"

g.DFS(2)

Output Observed for different Inputs:

Input passed:

Input passed:

g.BFS(1)

Output Obtained:

Following is Depth First Traversal

1 2 0 3