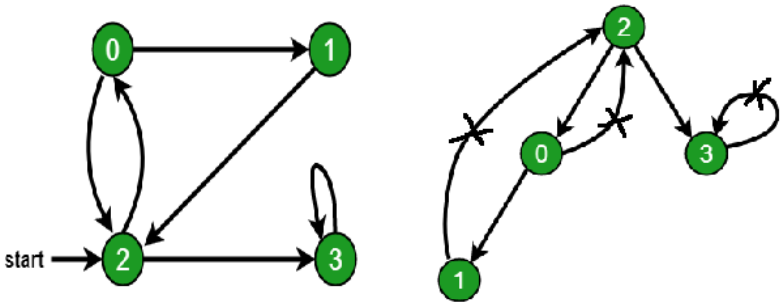
Detect Cycle in a Directed Graph

Given a directed graph, check whether the graph contains a cycle or not. Your function should return true if the given graph contains at least one cycle, else return false. For example, the following graph contains three cycles 0->2->0, 0->1->2->0 and 3->3, so your function must return true.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is from a node to itself (self-loop) or one of its ancestor in the tree produced by DFS. In the following graph, there are 3 back edges, marked with a cross sign. We can observe that these 3 back edges indicate 3 cycles present in the graph.



For a disconnected graph, we get the DFS forest as output. To detect cycle, we can check for a cycle in individual trees by checking back edges.

To detect a back edge, we can keep track of vertices currently in recursion stack of function for DFS traversal. If we reach a vertex that is already in the recursion stack, then there is a cycle in the tree. The edge that connects current vertex to the vertex in the recursion stack is a back edge. We have used recStack[] array to keep track of vertices in the recursion stack.

C++

```
// A C++ Program to detect cycle in a graph
#include<iostream>
#include <list>
#include <limits.h>

using namespace std;

class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // Pointer to an array containing adjacency lists
    bool isCyclicUtil(int v, bool visited[], bool *rs);  // used by isCyclic()
public:
    Graph(int V);   // Constructor
    void addEdge(int v, int w);   // to add an edge to graph
    bool isCyclic();    // returns true if there is a cycle in this 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.
}

// This function is a variation of DFSUytil() in https://www.geeksforgeeks.org/archives/18212
bool Graph::isCyclicUtil(int v, bool visited[], bool *recStack)
{
    if(visited[v] == false)
    {
        // Mark the current node as visited and part of recursion stack
        visited[v] = true;
        recStack[v] = true;

        // 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] && isCyclicUtil(*i, visited, recStack) )
                return true;
            else if (recStack[*i])
                return true;
        }

    }
    recStack[v] = false;  // remove the vertex from recursion stack
    return false;
}

// Returns true if the graph contains a cycle, else false.
// This function is a variation of DFS() in https://www.geeksforgeeks.org/archives/18212
bool Graph::isCyclic()
{
    // Mark all the vertices as not visited and not part of recursion
    // stack
    bool *visited = new bool[V];
    bool *recStack = new bool[V];
    for(int i = 0; i < V; i++)
    {
        visited[i] = false;
        recStack[i] = false;
    }

    // Call the recursive helper function to detect cycle in different
    // DFS trees
```

```
    for(int i = 0; i < V; i++)
        if (isCyclicUtil(i, visited, recStack))
            return true;

    return false;
}

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);

    if(g.isCyclic())
        cout << "Graph contains cycle";
    else
        cout << "Graph doesn't contain cycle";
    return 0;
}
```

Run on IDE

Java

```
// A Java Program to detect cycle in a graph
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

class Graph {

    private final int V;
    private final List<List<Integer>> adj;

    public Graph(int V)
    {
        this.V = V;
        adj = new ArrayList<>(V);

        for (int i = 0; i < V; i++)
            adj.add(new LinkedList<>());
    }

    // This function is a variation of DFSUytil() in
    // https://www.geeksforgeeks.org/archives/18212
    private boolean isCyclicUtil(int i, boolean[] visited,
                                 boolean[] recStack)
    {

        // Mark the current node as visited and
        // part of recursion stack
        if (recStack[i])
            return true;

        if (visited[i])
            return false;

        visited[i] = true;

        recStack[i] = true;
        List<Integer> children = adj.get(i);

        for (Integer c: children)
            if (isCyclicUtil(c, visited, recStack))
                return true;
```

```
        recStack[i] = false;

        return false;
    }

    private void addEdge(int source, int dest) {
        adj.get(source).add(dest);
    }

    // Returns true if the graph contains a
    // cycle, else false.
    // This function is a variation of DFS() in
    // https://www.geeksforgeeks.org/archives/18212
    private boolean isCyclic()
    {

        // Mark all the vertices as not visited and
        // not part of recursion stack
        boolean[] visited = new boolean[V];
        boolean[] recStack = new boolean[V];


        // Call the recursive helper function to
        // detect cycle in different DFS trees
        for (int i = 0; i < V; i++)
            if (isCyclicUtil(i, visited, recStack))
                return true;

        return false;
    }

    // Driver code
    public static void main(String[] args)
    {
        Graph graph = new Graph(4);
        graph.addEdge(0, 1);
        graph.addEdge(0, 2);
        graph.addEdge(1, 2);
        graph.addEdge(2, 0);
        graph.addEdge(2, 3);
        graph.addEdge(3, 3);

        if(graph.isCyclic())
            System.out.println("Graph contains cycle");
        else
            System.out.println("Graph doesn't "
                               + "contain cycle");
    }
}

// This code is contributed by Sagar Shah.
```

Run on IDE

Python

```
# Python program to detect cycle
# in a graph

from collections import defaultdict

class Graph():
    def __init__(self,vertices):
        self.graph = defaultdict(list)
        self.V = vertices

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def isCyclicUtil(self, v, visited, recStack):
```

```
        # Mark current node as visited and
        # adds to recursion stack
        visited[v] = True
        recStack[v] = True

        # Recur for all neighbours
        # if any neighbour is visited and in
        # recStack then graph is cyclic
        for neighbour in self.graph[v]:
            if visited[neighbour] == False:
                if self.isCyclicUtil(neighbour, visited, recStack) == True:
                    return True
            elif recStack[neighbour] == True:
                return True

        # The node needs to be poped from
        # recursion stack before function ends
        recStack[v] = False
        return False

    # Returns true if graph is cyclic else false
    def isCyclic(self):
        visited = [False] * self.V
        recStack = [False] * self.V
        for node in range(self.V):
            if visited[node] == False:
                if self.isCyclicUtil(node,visited,recStack) == True:
                    return True
        return False

g = Graph(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)
if g.isCyclic() == 1:
    print "Graph has a cycle"
else:
    print "Graph has no cycle"

# Thanks to Divyanshu Mehta for contributing this code
```

Run on IDE

Output:

```
Graph contains cycle
```

Time Complexity of this method is same as time complexity of DFS traversal which is O(V+E).

In the below article, another O(V + E) method is discussed :
Detect Cycle in a direct graph using colors

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

Improved By : SagarShah1

Practice Tags : Microsoft Amazon Flipkart Oracle Samsung Adobe MakeMyTrip BankBazaar Rockstand DFS Graph

Article Tags : Graph Adobe Amazon BankBazaar DFS Flipkart graph-cycle MakeMyTrip Microsoft Oracle Rockstand Samsung

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content. Login to Improve this Article

Recommended Posts:

Disjoint Set (Or Union-Find) | Set 1 (Detect Cycle in an Undirected Graph)
Detect cycle in an undirected graph
Detect Cycle in a directed graph using colors
Depth First Search or DFS for a Graph
Topological Sorting
Minimum cost path from source node to destination node via an intermediate node
Print all the cycles in an undirected graph
Dominant Set of a Graph
Print the DFS traversal step-wise (Backtracking also)
Find alphabetical order such that words can be considered sorted

(Login to Rate)

2.9 Average Difficulty : **2.9/5.0**
Based on **236** vote(s)

Basic | Easy | Medium | Hard | Expert

Add to TODO List
Mark as DONE

Feedback | Add Notes | Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments | Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org