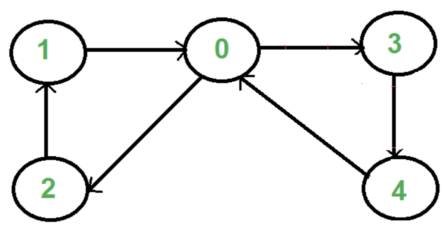
monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now.
START FREE TRIAL

Euler Circuit in a Directed Graph

Eulerian Path is a path in graph that visits every edge exactly once. Eulerian Circuit is an Eulerian Path which starts and ends on the same vertex.

A graph is said to be eulerian if it has eulerian cycle. We have discussed eulerian circuit for an undirected graph. In this post, same is discussed for a directed graph.

For example, the following graph has eulerian cycle as {1, 0, 3, 4, 0, 2, 1}



How to check if a directed graph is eulerian?
A directed graph has an eulerian cycle if following conditions are true (Source: Wiki)
1) All vertices with nonzero degree belong to a single strongly connected component.
2) In degree and out degree of every vertex is same.

We can detect singly connected component using Kosaraju's DFS based simple algorithm.
To compare in degree and out degree, we need to store in degree an out degree of every vertex. Out degree can be obtained by size of adjacency list. In degree can be stored by creating an array of size equal to number of vertices.

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

Following are C++ and Java implementations of above approach.

C++

```
// A C++ program to check if a given directed graph is Eulerian or not
#include<iostream>
#include <list>
#define CHARS 26
using namespace std;

// A class that represents an undirected graph
class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // A dynamic array of adjacency lists
    int *in;
public:
    // Constructor and destructor
    Graph(int V);
    ~Graph()   { delete [] adj; delete [] in; }
```

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now. START FREE TRIAL

```
    bool isEulerianCycle();

    // Method to check if all non-zero degree vertices are connected
    bool isSC();

    // Function to do DFS starting from v. Used in isConnected();
    void DFSUtil(int v, bool visited[]);

    Graph getTranspose();
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
    in = new int[V];
    for (int i = 0; i < V; i++)
       in[i] = 0;
}

/* This function returns true if the directed graph has an eulerian
   cycle, otherwise returns false  */
bool Graph::isEulerianCycle()
{
    // Check if all non-zero degree vertices are connected
    if (isSC() == false)
        return false;

    // Check if in degree and out degree of every vertex is same
    for (int i = 0; i < V; i++)
        if (adj[i].size() != in[i])
            return false;

    return true;
}

// A recursive function to do DFS starting from v
void Graph::DFSUtil(int v, bool visited[])
{
    // Mark the current node as visited and print it
    visited[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])
            DFSUtil(*i, visited);
}

// Function that returns reverse (or transpose) of this graph
// This function is needed in isSC()
Graph Graph::getTranspose()
{
    Graph g(V);
    for (int v = 0; v < V; v++)
    {
        // Recur for all the vertices adjacent to this vertex
        list<int>::iterator i;
        for(i = adj[v].begin(); i != adj[v].end(); ++i)
        {
            g.adj[*i].push_back(v);
            (g.in[v])++;
        }
    }
    return g;
}
```

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now. START FREE TRIAL

```
bool Graph::isSC()
{
    // Mark all the vertices as not visited (For first DFS)
    bool visited[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Find the first vertex with non-zero degree
    int n;
    for (n = 0; n < V; n++)
        if (adj[n].size() > 0)
          break;

    // Do DFS traversal starting from first non zero degree vertex.
    DFSUtil(n, visited);

     // If DFS traversal doesn't visit all vertices, then return false.
    for (int i = 0; i < V; i++)
        if (adj[i].size() > 0 && visited[i] == false)
              return false;

    // Create a reversed graph
    Graph gr = getTranspose();

    // Mark all the vertices as not visited (For second DFS)
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Do DFS for reversed graph starting from first vertex.
    // Staring Vertex must be same starting point of first DFS
    gr.DFSUtil(n, visited);

    // If all vertices are not visited in second DFS, then
    // return false
    for (int i = 0; i < V; i++)
        if (adj[i].size() > 0 && visited[i] == false)
             return false;

    return true;
}

// Driver program to test above functions
int main()
{
    // Create a graph given in the above diagram
    Graph g(5);
    g.addEdge(1, 0);
    g.addEdge(0, 2);
    g.addEdge(2, 1);
    g.addEdge(0, 3);
    g.addEdge(3, 4);
    g.addEdge(4, 0);

    if (g.isEulerianCycle())
       cout << "Given directed graph is eulerian n";
    else
       cout << "Given directed graph is NOT eulerian n";
    return 0;
}
```

Java

```
// A Java program to check if a given directed graph is Eulerian or not
```

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now.
HIDE AD • AD VIA BUYSELLADS
START FREE TRIAL

```
import java.util.LinkedList;

// This class represents a directed graph using adjacency list
class Graph
{
    private int V;   // No. of vertices
    private LinkedList<Integer> adj[];//Adjacency List
    private int in[];            //maintaining in degree

    //Constructor
    Graph(int v)
    {
        V = v;
        adj = new LinkedList[v];
        in = new int[V];
        for (int i=0; i<v; ++i)
        {
            adj[i] = new LinkedList();
            in[i]  = 0;
        }
    }

    //Function to add an edge into the graph
    void addEdge(int v,int w)
    {
        adj[v].add(w);
        in[w]++;
    }

    // A recursive function to print DFS starting from v
    void DFSUtil(int v,Boolean visited[])
    {
        // Mark the current node as visited
        visited[v] = true;

        int n;

        // Recur for all the vertices adjacent to this vertex
        Iterator<Integer> i =adj[v].iterator();
        while (i.hasNext())
        {
            n = i.next();
            if (!visited[n])
                DFSUtil(n,visited);
        }
    }

    // Function that returns reverse (or transpose) of this graph
    Graph getTranspose()
    {
        Graph g = new Graph(V);
        for (int v = 0; v < V; v++)
        {
            // Recur for all the vertices adjacent to this vertex
            Iterator<Integer> i = adj[v].listIterator();
            while (i.hasNext())
            {
                g.adj[i.next()].add(v);
                (g.in[v])++;
            }
        }
        return g;
    }

    // The main function that returns true if graph is strongly
    // connected
```

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now. START FREE TRIAL

```
        Boolean visited[] = new Boolean[V];
        for (int i = 0; i < V; i++)
            visited[i] = false;

        // Step 2: Do DFS traversal starting from first vertex.
        DFSUtil(0, visited);

        // If DFS traversal doesn't visit all vertices, then return false.
        for (int i = 0; i < V; i++)
            if (visited[i] == false)
                return false;

        // Step 3: Create a reversed graph
        Graph gr = getTranspose();

        // Step 4: Mark all the vertices as not visited (For second DFS)
        for (int i = 0; i < V; i++)
            visited[i] = false;

        // Step 5: Do DFS for reversed graph starting from first vertex.
        // Staring Vertex must be same starting point of first DFS
        gr.DFSUtil(0, visited);

        // If all vertices are not visited in second DFS, then
        // return false
        for (int i = 0; i < V; i++)
            if (visited[i] == false)
                return false;

        return true;
    }

    /* This function returns true if the directed graph has an eulerian
       cycle, otherwise returns false  */
    Boolean isEulerianCycle()
    {
        // Check if all non-zero degree vertices are connected
        if (isSC() == false)
            return false;

        // Check if in degree and out degree of every vertex is same
        for (int i = 0; i < V; i++)
            if (adj[i].size() != in[i])
                return false;

        return true;
    }

    public static void main (String[] args) throws java.lang.Exception
    {
        Graph g = new Graph(5);
        g.addEdge(1, 0);
        g.addEdge(0, 2);
        g.addEdge(2, 1);
        g.addEdge(0, 3);
        g.addEdge(3, 4);
        g.addEdge(4, 0);

        if (g.isEulerianCycle())
            System.out.println("Given directed graph is eulerian ");
        else
            System.out.println("Given directed graph is NOT eulerian ");
    }
}
//This code is contributed by Aakash Hasija
```

Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now. START FREE TRIAL

```
# A Python program to check if a given
# directed graph is Eulerian or not

from collections import defaultdict

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = defaultdict(list)
        self.IN = [0] * vertices

    def addEdge(self, v, u):

        self.graph[v].append(u)
        self.IN[u] += 1

    def DFSUtil(self, v, visited):
        visited[v] = True
        for node in self.graph[v]:
            if visited[node] == False:
                self.DFSUtil(node, visited)

    def getTranspose(self):
        gr = Graph(self.V)

        for node in range(self.V):
            for child in self.graph[node]:
                gr.addEdge(child, node)

        return gr

    def isSC(self):
        visited = [False] * self.V

        v = 0
        for v in range(self.V):
            if len(self.graph[v]) > 0:
                break

        self.DFSUtil(v, visited)

        # If DFS traversal doesn't visit all
        # vertices, then return false.
        for i in range(self.V):
            if visited[i] == False:
                return False

        gr = self.getTranspose()

        visited = [False] * self.V
        gr.DFSUtil(v, visited)

        for i in range(self.V):
            if visited[i] == False:
                return False

        return True

    def isEulerianCycle(self):

        # Check if all non-zero degree vertices
        # are connected
        if self.isSC() == False:
            return False
```

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now.
START FREE TRIAL

```
                return False

        return True


g = Graph(5);
g.addEdge(1, 0);
g.addEdge(0, 2);
g.addEdge(2, 1);
g.addEdge(0, 3);
g.addEdge(3, 4);
g.addEdge(4, 0);
if g.isEulerianCycle():
   print "Given directed graph is eulerian";
else:
   print "Given directed graph is NOT eulerian";

# This code is contributed by Divyanshu Mehta
```

Output:

```
Given directed graph is eulerian
```

Time complexity of the above implementation is O(V + E) as Kosaraju's algorithm takes O(V + E) time. After running Kosaraju's algorithm we traverse all vertices and compare in degree with out degree which takes O(V) time.

See following as an application of this.
Find if the given array of strings can be chained to form a circle.

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

Recommended Posts:

Minimum edges required to add to make Euler Circuit
Convert the undirected graph into directed graph such that there is no path of length greater than 1
Eulerian path and circuit for undirected graph
Program to find Circuit Rank of an Undirected Graph
Check if a directed graph is connected or not
Detect Cycle in a Directed Graph using BFS
Hierholzer's Algorithm for directed graph
Detect Cycle in a Directed Graph
Clone a Directed Acyclic Graph
Longest Path in a Directed Acyclic Graph | Set 2
All Topological Sorts of a Directed Acyclic Graph
Detect Cycle in a directed graph using colors
Shortest Path in Directed Acyclic Graph
Longest Path in a Directed Acyclic Graph
Find if there is a path between two vertices in a directed graph

monday.com Project tracking, teamwork & client reporting like you've never seen before. Start Your Free Trial Now.

START FREE TRIAL

Practice Tags : DFS Graph

Be the First to upvote.

To-do Done

3.7

Based on **24** vote(s)

Feedback/ Suggest Improvement Add Notes Improve Article

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

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

Load Comments

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY
About Us
Careers
Privacy Policy
Contact Us

LEARN
Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

PRACTICE
Courses
Company-wise
Topic-wise
How to begin?

CONTRIBUTE
Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved