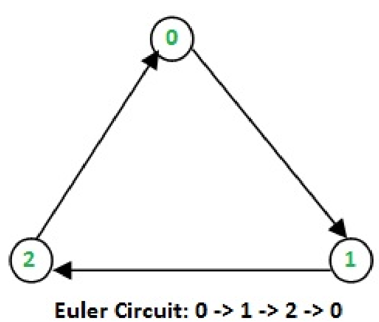
Hierholzer's Algorithm for directed graph

Given a directed Eulerian graph, print an Euler circuit. Euler circuit is a path that traverses every edge of a graph, and the path ends on the starting vertex.

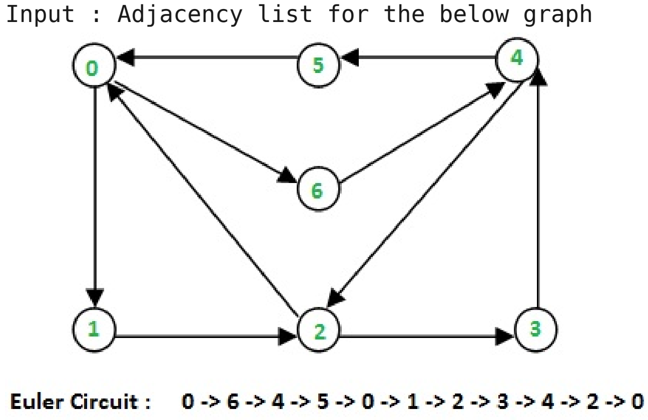
Examples:

```
Input : Adjacency list for the below graph
```




```
Output : 0 -> 1 -> 2 -> 0

Input : Adjacency list for the below graph
```




```
Output : 0 -> 6 -> 4 -> 5 -> 0 -> 1
         -> 2 -> 3 -> 4 -> 2 -> 0
Explanation:
In both the cases, we can trace the Euler circuit
by following the edges as indicated in the output.
```

Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.

is O(E*E). Using Hierholzer's Algorithm, we can find the circuit/path in O(E), i.e., linear time.

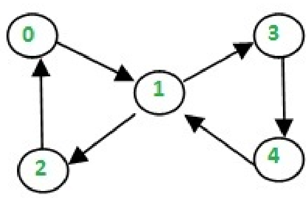
Below is the Algorithm: ref (wiki). Remember that a directed graph has an Eulerian cycle if following conditions are true (1) All vertices with nonzero degree belong to a single strongly connected component. (2) In degree and out degree of every vertex is same. The algorithm assumes that the given graph has Eulerian Circuit.

- Choose any starting vertex v, and follow a trail of edges from that vertex until returning to v. It is not possible to get stuck at any vertex other than v, because indegree and outdegree of every vertex must be same, when the trail enters another vertex w there must be an unused edge leaving w.
  The tour formed in this way is a closed tour, but may not cover all the vertices and edges of the initial graph.
- As long as there exists a vertex u that belongs to the current tour but that has adjacent edges not part of the tour, start another trail from u, following unused edges until returning to u, and join the tour formed in this way to the previous tour.

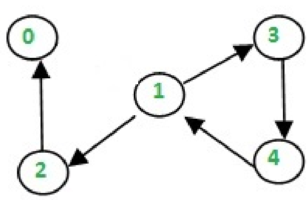
Thus the idea is to keep following unused edges and removing them until we get stuck. Once we get stuck, we back-track to the nearest vertex in our current path that has unused edges, and we repeat the process until all the edges have been used. We can use another container to maintain the final path.

Let's take an example:

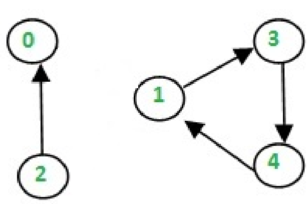
```
Let the initial directed graph be as below
```



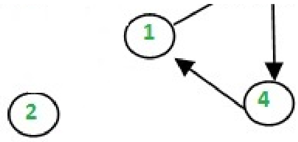

```
Let's start our path from 0.
Thus, curr_path = {0} and circuit = {}
Now let's use the edge 0->1
```




```
Now, curr_path = {0,1} and circuit = {}
similarly we reach up to 2 and then to 0 again as
```




```
Now, curr_path = {0,1,2} and circuit = {}
Then we go to 0, now since 0 haven't got any unused
edge we put 0 in circuit and back track till we find
```




```
We then have curr_path = {0,1,2} and circuit = {0}
Similarly when we backtrack to 2, we don't find any
unused edge. Hence put 2 in circuit and backtrack
again.

curr_path = {0,1} and circuit = {0,2}

After reaching 1 we go to through unused edge 1->3 and
then 3->4, 4->1 until all edges have been traversed.

The contents of the two containers look as:
curr_path = {0,1,3,4,1} and circuit = {0,2}

now as all edges have been used, the curr_path is
popped one by one into circuit.
Finally we've circuit = {0,2,1,4,3,1,0}

We print the circuit in reverse to obtain the path followed.
i.e., 0->1->3->4->1->1->2->0
```

Below is the C++ program for the same.

```
// A C++ program to print Eulerian circuit in given
// directed graph using Hierholzer algorithm
#include <bits/stdc++.h>
using namespace std;

void printCircuit(vector< vector<int> > adj)
{
    // adj represents the adjacency list of
    // the directed graph
    // edge_count represents the number of edges
    // emerging from a vertex
    unordered_map<int,int> edge_count;

    for (int i=0; i<adj.size(); i++)
    {
        //find the count of edges to keep track
        //of unused edges
        edge_count[i] = adj[i].size();
    }

    if (!adj.size())
        return; //empty graph

    // Maintain a stack to keep vertices
    stack<int> curr_path;

    // vector to store final circuit
    vector<int> circuit;

    // start from any vertex
    curr_path.push(0);
```

```
        // If there's remaining edge
        if (edge_count[curr_v])
        {
            // Push the vertex
            curr_path.push(curr_v);

            // Find the next vertex using an edge
            int next_v = adj[curr_v].back();

            // and remove that edge
            edge_count[curr_v]--;
            adj[curr_v].pop_back();

            // Move to next vertex
            curr_v = next_v;
        }

        // back-track to find remaining circuit
        else
        {
            circuit.push_back(curr_v);

            // Back-tracking
            curr_v = curr_path.top();
            curr_path.pop();
        }
    }

    // we've got the circuit, now print it in reverse
    for (int i=circuit.size()-1; i>=0; i--)
    {
        cout << circuit[i];
        if (i)
           cout<<" -> ";
    }
}

// Driver program to check the above function
int main()
{
    vector< vector<int> > adj1, adj2;

    // Input Graph 1
    adj1.resize(3);

    // Build the edges
    adj1[0].push_back(1);
    adj1[1].push_back(2);
    adj1[2].push_back(0);
    printCircuit(adj1);
    cout << endl;

    // Input Graph 2
    adj2.resize(7);
    adj2[0].push_back(1);
    adj2[0].push_back(6);
    adj2[1].push_back(2);
    adj2[2].push_back(0);
    adj2[2].push_back(3);
    adj2[3].push_back(4);
    adj2[4].push_back(2);
    adj2[4].push_back(5);
    adj2[5].push_back(0);
    adj2[6].push_back(4);
    printCircuit(adj2);
```

Output:

```
0 -> 1 -> 2 -> 0
0 -> 6 -> 4 -> 5 -> 0 -> 1 -> 2 -> 3 -> 4 -> 2 -> 0
```

Time Complexity : O(V+E).

This article is contributed by **Ashutosh Kumar**. The article contains also inputs from **Nitish Kumar**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

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

Recommended Posts:

Convert the undirected graph into directed graph such that there is no path of length greater than 1
Euler Circuit in a Directed Graph
Detect Cycle in a Directed Graph using BFS
Clone a Directed Acyclic Graph
Check if a directed graph is connected or not
Detect Cycle in a Directed Graph
All Topological Sorts of a Directed Acyclic Graph
Shortest Path in Directed Acyclic Graph
Longest Path in a Directed Acyclic Graph
Detect Cycle in a directed graph using colors
Longest Path in a Directed Acyclic Graph | Set 2
Find if there is a path between two vertices in a directed graph
Shortest path with exactly k edges in a directed and weighted graph
Number of shortest paths in an unweighted and directed graph
Check if a given directed graph is strongly connected | Set 2 (Kosaraju using BFS)

Article Tags : Graph Euler-Circuit

Practice Tags : Graph

2

To-do Done

3.5

Based on **13** vote(s)

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.

6 Comments GeeksforGeeks 1 Login

Recommend 2 Tweet Share Sort by Newest

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

Rohit Agarwal • 5 months ago

Why do you need edge_count when clearly we are destroying the input graph? Can't we test it directly by adj[i].size() ?

Reply • Share ›

Deepak chaudhary • 2 years ago • edited

A simple code to print euler tour

```
#include<bits stdc++.h="">
using namespace std;
vector<vector<int> >v(1007);
int main()
{
int next=0,curr;
v[0].push_back(1);
v[1].push_back(2);
v[2].push_back(0);

vector<int> ans;
ans.push_back(next);
while(1){
if(v[next].empty())
break;
curr = v[next].back();
```

see more

2 Reply • Share ›

Nishan Tamng → Deepak chaudhary • a year ago

Nice and easy implementation! Thanks

Reply • Share ›

Sudhanshu Monga • 2 years ago

http://ideone.com/pANMWg

simple recursive code, please report bugs, if any.

1 Reply • Share ›

Anas Mahmoud • 2 years ago

Here you pushed vertex 0 TWICE to curr_path for the first run of the algorithm (at line 1 and line 5), or did I miss something ?

```
// start from any vertex
1- curr_path.push(0);
2- int curr_v = 0; // Current vertex

3- while (!curr_path.empty())
{
// If there's remaining edge
4- if (edge_count[curr_v])
{
// Push the vertex
5- curr_path.push(curr_v);
```

• Reply • Share ›

Kfir Berger → Anas Mahmoud • 2 years ago

yes, same question here...

• Reply • Share ›