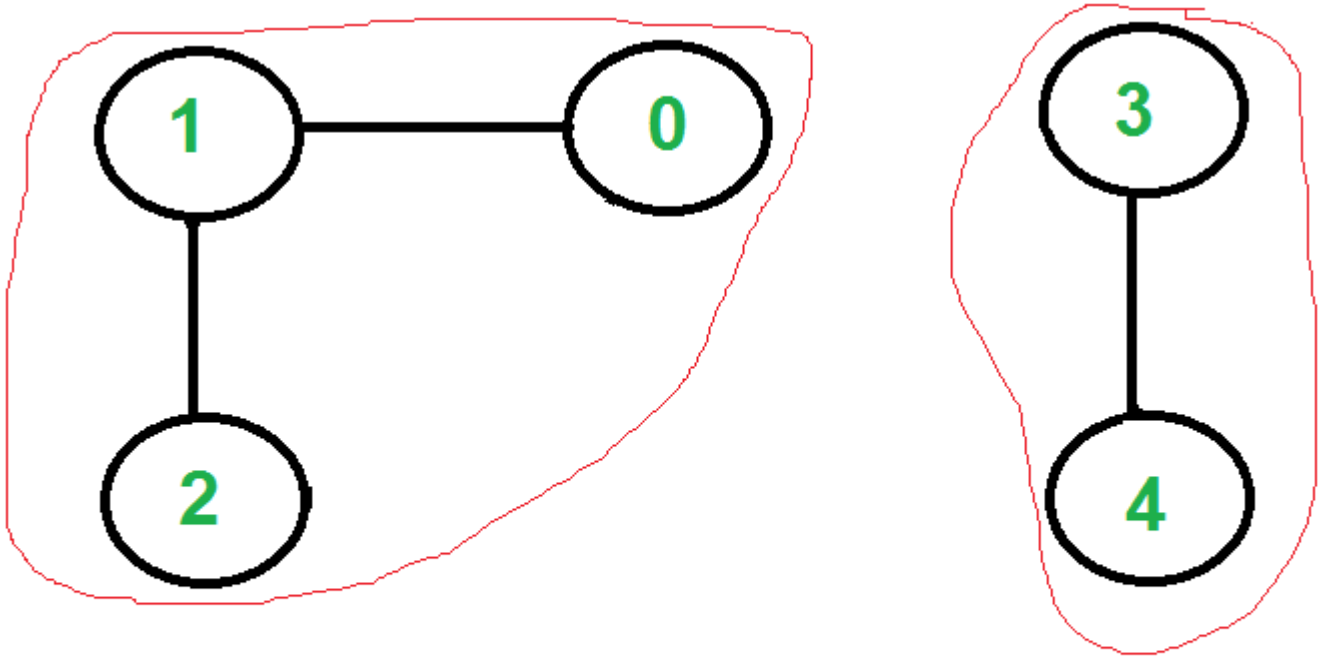
Dash
All
Articles
Videos
Problems
Quiz
Contest
Prev
Next

Number of Strongly Connected Components in an Undirected Graph

Problem: Given an Undirected Graph. The task is to find the count of the number of *strongly connected components* in the given Graph. A **Strongly Connected Component** is defined as a subgraph of this graph in which every pair of vertices has a path in between.



There are two connected components in above undirected graph
0 1 2
3 4

unvisited vertices, and for each unvisited vertex print, it's DFS or BFS traversal.

Below is the algorithm following the DFS traversal to find all connected components in an undirected graph:

```
1) Initialize all vertices as not visited.
2) Do following for every vertex 'v'.
       (a) If 'v' is not visited before, call DFSUtil(v)
       (b) Print new line character
```

```
// This Function performs DFS traversal
// of vertex v.
DFSUtil(v)
1) Mark 'v' as visited.
2) Print 'v'
3) Do following for every adjacent 'u' of 'v'.
     If 'u' is not visited, then recursively call DFSUtil(u)
```

Implementation:

C++ Java

```java
// Java program to print connected components in
// an undirected graph
import java.util.LinkedList;
class Graph {
    // A user define class to represent a graph.
    // A graph is an array of adjacency lists.
    // Size of array will be V (number of vertices
    // in graph)
    int V;
    LinkedList<Integer>[] adjListArray;

    // constructor
    Graph(int V)
    {
        this.V = V;
        // define the size of array as
        // number of vertices
        adjListArray = new LinkedList[V];
```

```
        // Create a new list for each vertex
        // such that adjacent nodes can be stored

        for (int i = 0; i < V; i++) {
            adjListArray[i] = new LinkedList<Integer>();
        }
    }

    // Adds an edge to an undirected graph
    void addEdge(int src, int dest)
    {
        // Add an edge from src to dest.
        adjListArray[src].add(dest);

        // Since graph is undirected, add an edge from dest
        // to src also
        adjListArray[dest].add(src);
    }

    void DFSUtil(int v, boolean[] visited)
    {
        // Mark the current node as visited and print it
        visited[v] = true;
        System.out.print(v + " ");
        // Recur for all the vertices
        // adjacent to this vertex
        for (int x : adjListArray[v]) {
            if (!visited[x])
                DFSUtil(x, visited);
        }
    }
    void connectedComponents()
    {
        // Mark all the vertices as not visited
        boolean[] visited = new boolean[V];
        for (int v = 0; v < V; ++v) {
            if (!visited[v]) {
                // print all reachable vertices
                // from v
                DFSUtil(v, visited);
                System.out.println();
```

```
                }
            }
        }

        // Driver program to test above
        public static void main(String[] args)
        {
            // Create a graph given in the above diagram
            Graph g = new Graph(5); // 5 vertices numbered from 0 to 4

            g.addEdge(1, 0);
            g.addEdge(1, 2);
            g.addEdge(3, 4);
            System.out.println("Following are connected components");
            g.connectedComponents();
        }
    }
```

Output:

```
Following are connected components
0 1 2
3 4
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

If you are facing any issue on this page. Please let us know.