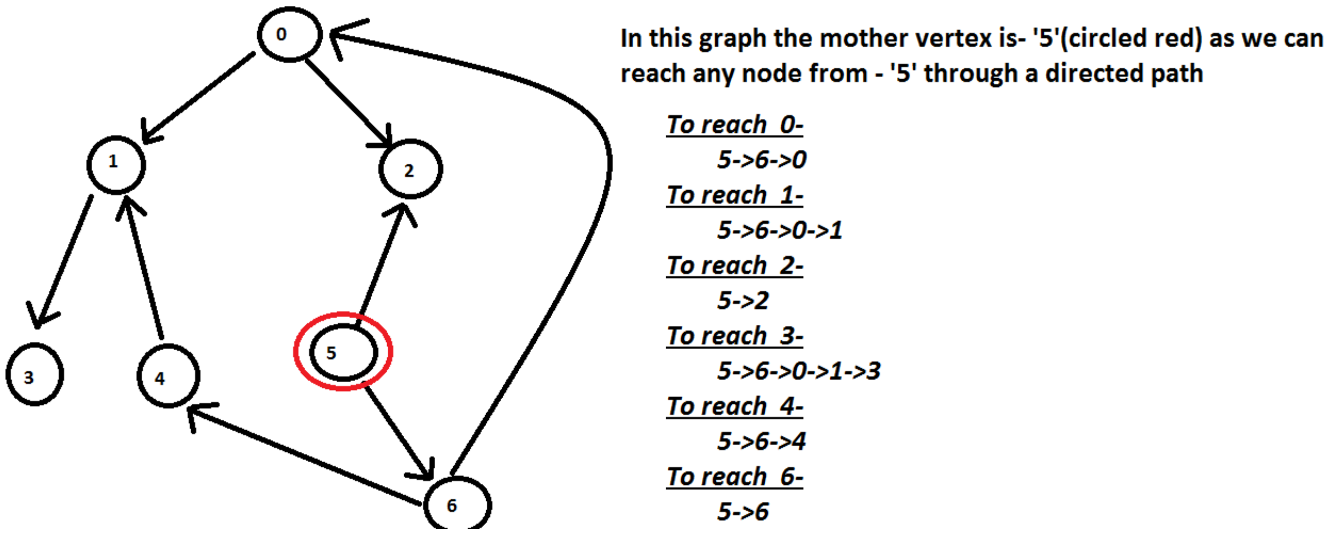
Find a Mother Vertex in a Graph

Last Updated: 17-07-2018

What is a Mother Vertex?

A mother vertex in a graph G = (V,E) is a vertex v such that all other vertices in G can be reached by a path from v.

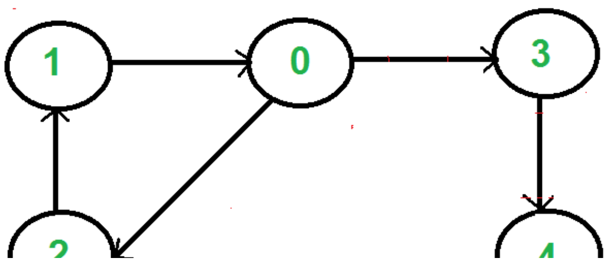
```
Input  : Below Graph
Output : 5
```



In this graph the mother vertex is- '5'(circled red) as we can reach any node from - '5' through a directed path

To reach 0-
5->6->0
To reach 1-
5->6->0->1
To reach 2-
5->2
To reach 3-
5->6->0->1->3
To reach 4-
5->6->4
To reach 6-
5->6

There can be more than one mother vertices in a graph. We need to output anyone of them. For example, in the below graph, vertices 0, 1 and 2 are mother vertices.



We strongly recommend you to minimize your browser and try this yourself first.

How to find mother vertex?

- ***Case 1:- Undirected Connected Graph :*** In this case, all the vertices are mother vertices as we can reach to all the other nodes in the graph.
- ***Case 2:- Undirected/Directed Disconnected Graph*** : In this case, there is no mother vertices as we cannot reach to all the other nodes in the graph.
- ***Case 3:- Directed Connected Graph*** : In this case, we have to find a vertex -v in the graph such that we can reach to all the other nodes in the graph through a directed path.

A Naive approach :
A trivial approach will be to perform a DFS/BFS on all the vertices and find whether we can reach all the vertices from that vertex. This approach takes O(V(E+V)) time, which is very inefficient for large graphs.

Can we do better?
We can find a mother vertex in O(V+E) time. The idea is based on Kosaraju's Strongly Connected Component Algorithm. In a graph of strongly connected components, mother vertices are always vertices of source component in component graph. The idea is based on below fact.

If there exist mother vertex (or vertices), then one of the mother vertices is the last finished vertex in DFS. (Or a mother vertex has the maximum finish time in DFS traversal).

A vertex is said to be finished in DFS if a recursive call for its DFS is over, i.e., all descendants of

How does the above idea work?

Let the last finished vertex be v. Basically, we need to prove that there cannot be an edge from another vertex u to v if u is not another mother vertex (Or there cannot exist a non-mother vertex u such that u-→v is an edge). There can be two possibilities.

1. Recursive DFS call is made for u before v. If an edge u-→v exists, then v must have finished before u because v is reachable through u and a vertex finishes after all its descendants.
2. Recursive DFS call is made for v before u. In this case also, if an edge u-→v exists, then either v must finish before u (which contradicts our assumption that v is finished at the end) OR u should be reachable from v (which means u is another mother vertex).

Algorithm :

1. Do DFS traversal of the given graph. While doing traversal keep track of last finished vertex 'v'. This step takes O(V+E) time.
2. If there exist mother vertex (or vetices), then v must be one (or one of them). Check if v is a mother vertex by doing DFS/BFS from v. This step also takes O(V+E) time.

Below is implementation of above algorithm.

C/C++

```
// C++ program to find a mother vertex in O(V+E) time
#include <bits/stdc++.h>
using namespace std;

class Graph
{
    int V;    // No. of vertices
    list<int> *adj;    // adjacency lists

    // A recursive function to print DFS starting from v
    void DFSUtil(int v, vector<bool> &visited);
public:
    Graph(int V);
    void addEdge(int v, int w);
    int findMother();
};

Graph::Graph(int V)
{
    this->V = V;
```

```
// A recursive function to print DFS starting from v
void Graph::DFSUtil(int v, vector<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);
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}

// Returns a mother vertex if exists. Otherwise returns -1
int Graph::findMother()
{
    // visited[] is used for DFS. Initially all are
    // initialized as not visited
    vector <bool> visited(V, false);

    // To store last finished vertex (or mother vertex)
    int v = 0;

    // Do a DFS traversal and find the last finished
    // vertex
    for (int i = 0; i < V; i++)
    {
        if (visited[i] == false)
        {
            DFSUtil(i, visited);
            v = i;
        }
    }

    // If there exist mother vertex (or vetices) in given
    // graph, then v must be one (or one of them)

    // Now check if v is actually a mother vertex (or graph
    // has a mother vertex).  We basically check if every vertex
    // is reachable from v or not.

    // Reset all values in visited[] as false and do
    // DFS beginning from v to check if all vertices are
    // reachable from it or not.
    fill(visited.begin(), visited.end(), false);
    DFSUtil(v, visited);
```

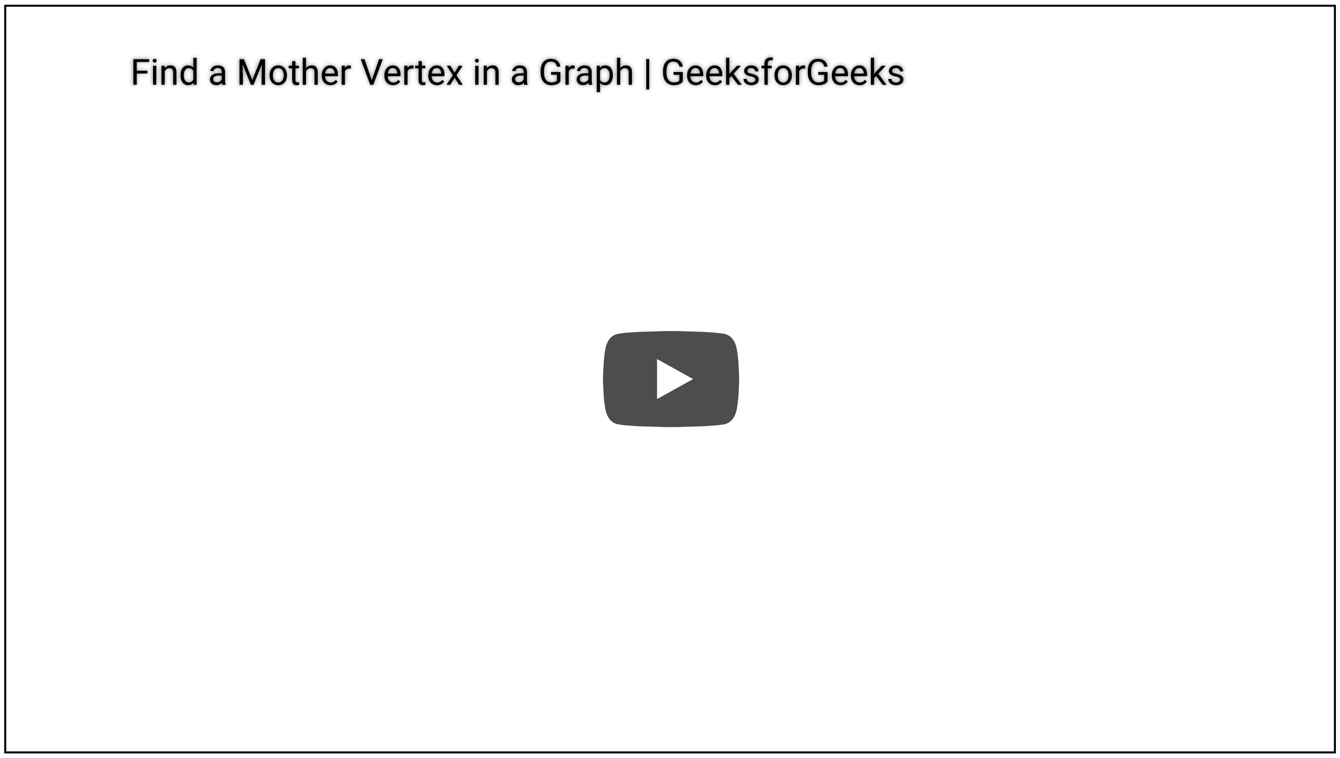
```
        return v;
    }

    // Driver program to test above functions
    int main()
    {
        // Create a graph given in the above diagram
        Graph g(7);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 3);
        g.addEdge(4, 1);
        g.addEdge(6, 4);
        g.addEdge(5, 6);
        g.addEdge(5, 2);
        g.addEdge(6, 0);

        cout << "A mother vertex is " << g.findMother();

        return 0;
    }
```

Python

```
# program to find a mother vertex in O(V+E) time
from collections import defaultdict

# This class represents a directed graph using adjacency list
# representation
class Graph:

    def __init__(self,vertices):
        self.V = vertices #No. of vertices
        self.graph = defaultdict(list) # default dictionary

    # A recursive function to print DFS starting from v
    def DFSUtil(self, v, visited):

        # Mark the current node as visited and print it
        visited[v] = True

        # Recur for all the vertices adjacent to this vertex
        for i in self.graph[v]:
            if visited[i] == False:
                self.DFSUtil(i, visited)
    # Returns a mother vertex if exists. Otherwise returns -1
    def findMother(self):

        # visited[] is used for DFS. Initially all are
        # initialized as not visited
        visited =[False]*(self.V)

        # To store last finished vertex (or mother vertex)
        v=0

        # Do a DFS traversal and find the last finished
        # vertex
        for i in range(self.V):
            if visited[i]==False:
                self.DFSUtil(i,visited)
                v = i

        # If there exist mother vertex (or vetices) in given
        # graph, then v must be one (or one of them)

        # Now check if v is actually a mother vertex (or graph
        # has a mother vertex). We basically check if every vertex
        # is reachable from v or not.

        # Reset all values in visited[] as false and do
        # DFS beginning from v to check if all vertices are
        # reachable from it or not.
        visited = [False]*(self.V)
        self.DFSUtil(v, visited)
        if any(i == False for i in visited):
            return -1
        else:
            return v

# Create a graph given in the above diagram
g = Graph(7)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(4, 1)
g.addEdge(6, 4)
g.addEdge(5, 6)
g.addEdge(5, 2)
g.addEdge(6, 0)
print "A mother vertex is " + str(g.findMother())

# This code is contributed by Neelam Yadav
```

Output :

Time Complexity : O(V + E)


Find a Mother Vertex in a Graph | GeeksforGeeks

This article is contributed by **Rachit Belwariar**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Attention reader! Don’t stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready.

Find a Mother vertex in a Graph using Bit Masking

Find dependencies of each Vertex in a Directed Graph

Check if vertex X lies in subgraph of vertex Y for the given Graph

All vertex pairs connected with exactly k edges in a graph

Find k-cores of an undirected graph

Find the maximum value permutation of a graph

Find if there is a path between two vertices in an undirected graph

Find any simple cycle in an undirected unweighted Graph

Find the maximum component size after addition of each edge to the graph

Print all possible paths in a DAG from vertex whose indegree is 0

Sum of Nodes and respective Neighbors on the path from root to a vertex V

Find K vertices in the graph which are connected to at least one of remaining vertices

Bridges in a graph

Biconnected graph

Check if a given graph is Bipartite using DFS

Check whether a given graph is Bipartite or not

Check if a given graph is tree or not

Depth First Search or DFS for a Graph

Count of different groups using Graph

Check if a given Graph is 2-edge connected or not

Improved By : ShubhamDixit

Article Tags : Graph DFS graph-connectivity

Practice Tags : DFS Graph

66

3.3

Feedback/ Suggest Improvement 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