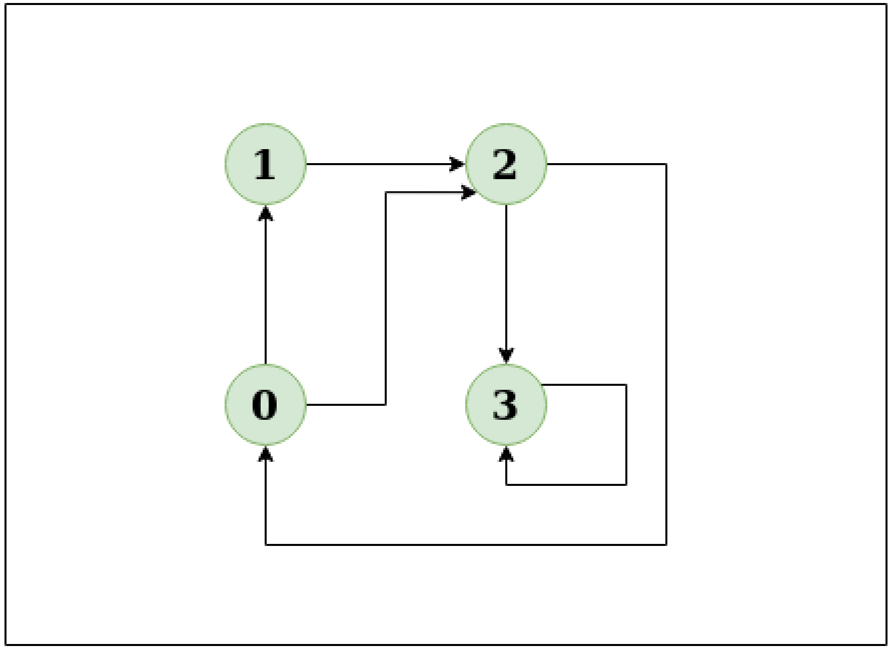
Dash
All
Articles
Videos
Problems
Quiz
Contest
« Prev
Next »

Detect Cycle in a Directed Graph

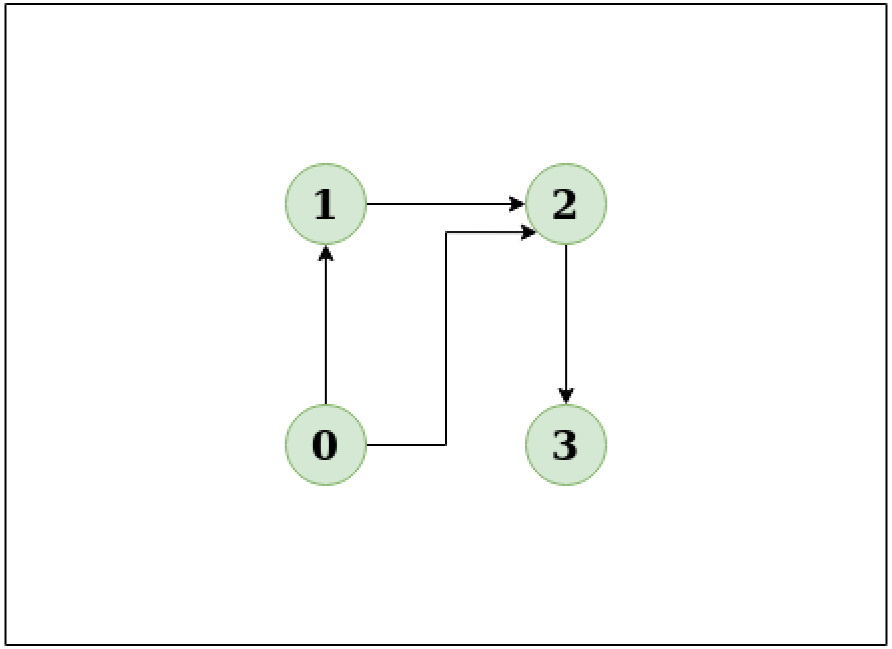
Given the root of a **Directed graph**, The task is to check whether the graph contains a cycle if yes then return **true**, return **false** otherwise.

Examples:

> **Input:** N = 4, E = 6
>
> 
>
>
> **Output:** Yes
> **Explanation:** The diagram clearly shows a cycle 0 -> 2 -> 0

Dash
All
Articles
Videos
Problems
Quiz
Contest

Input: N = 4, E = 4



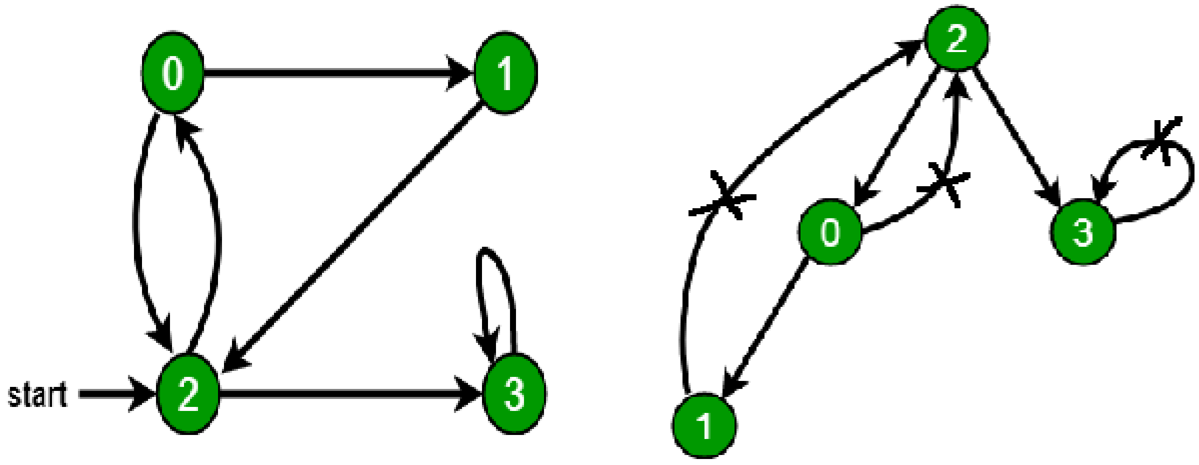
Output: No
Explanation: The diagram clearly shows no cycle

There is a cycle in a graph only if there is a back edge present in the graph. Depth First Traversal can be used to detect a cycle in a Graph, DFS for a connected graph produces a tree.

If the graph is disconnected then get the DFS forest and check for a cycle in individual trees by checking back edges. To detect a back edge, keep track of vertices currently in the recursion stack of function for DFS traversal. If a vertex is reached that is already in the recursion stack then there is a cycle in the tree.

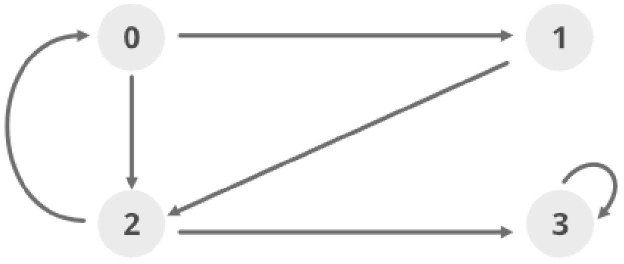
Note: A Back edge is an edge that is from a node to itself (self-loop) or one of its ancestors in the tree produced by DFS. Thus the edge that connects the current vertex to the vertex in the recursion stack is a back edge.

Illustration:

> In the following graph, there are 3 back edges, marked with a cross sign. Observe that these 3 back edges indicate 3 cycles present in the graph.
>
> 
>
>
> Use **recStack[]** array to keep track of vertices in the recursion stack.

Dry run of the above approach:

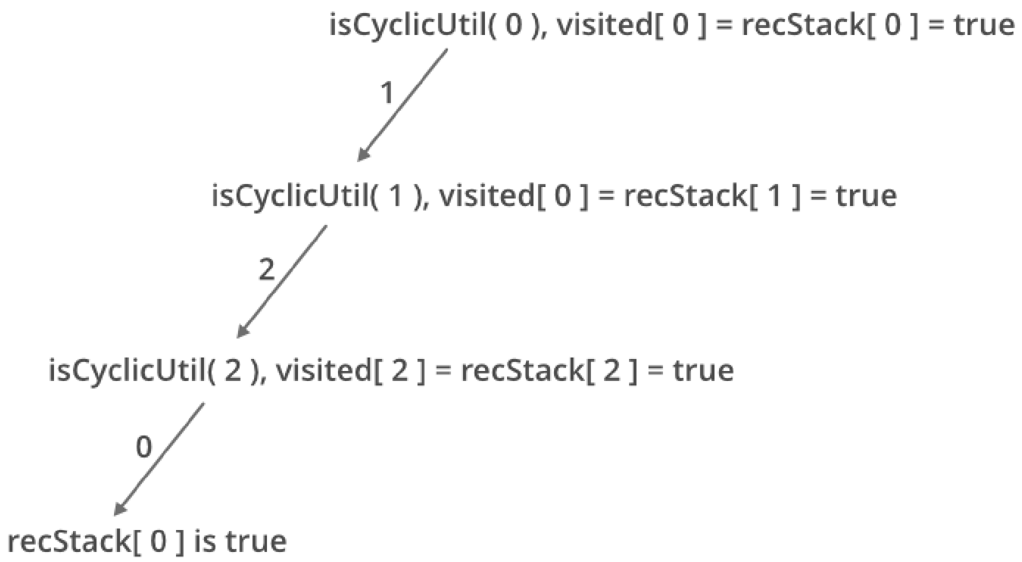
Dash
All
Articles
Videos
Problems
Quiz
Contest



Adja cent list (G) 0 → 1, 2
1 → 0
2 → 0, 3
3 → 3

Initially :

	A	B	C	D
visited	false	false	false	false
recStack	false	false	false	false


isCyclicUtil(0), visited[0] = recStack[0] = true

isCyclicUtil(1), visited[0] = recStack[1] = true

isCyclicUtil(2), visited[2] = recStack[2] = true

recStack[0] is true

Dash
All
Articles
Videos
Problems
Quiz
Contest

Follow the below steps to Implement the idea:

- Create the graph using the given number of edges and vertices.
- Create a recursive function that initializes the **current vertex, visited array, and recursion stack**.
- Mark the current node as visited and also mark the index in the recursion stack.
- Find all the vertices which are not visited and are adjacent to the current node and recursively call the function for those vertices
  - If the recursive function returns true, return **true**.
  - If the adjacent vertices are already marked in the recursion stack then return **true**.
- Create a wrapper function, that calls the recursive function for all the vertices, and
  - If any function returns true return **true**.
  - Else if for all vertices the function returns false return **false**.

Below is the implementation of the above approach:

C++ Java

```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;
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
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 DFSUtil() in
// https://www.cdn.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);
}
```

Dash
All
Articles
Videos

```
    // Returns true if the graph contains a
    // cycle, else false.
    // This function is a variation of DFS() in
    // https://www.cdn.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;
    }
```

Get 90% Refund!
Courses Tutorials Jobs Practice Contests
Quiz
Contest

```
        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");
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

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

Output

```
Graph contains cycle
```

Time Complexity: O(V+E), the Time Complexity of this method is the same as the time complexity of DFS traversal which is O(V+E).
Auxiliary Space: O(V). To store the visited and recursion stack O(V) space is needed.

Mark as Read

Report An Issue

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