**Detect Cycle in a Directed Graph**

Given a directed graph, check whether the graph contains a cycle or not. Your function should return true if the given graph contains at least one cycle, else return false.  
**Example,**

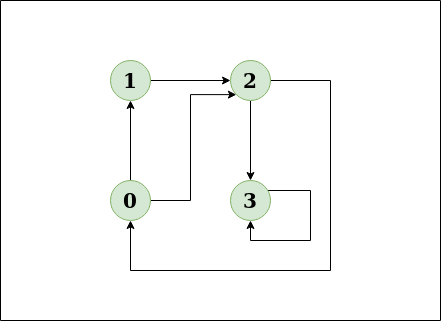
**Input:** n = 4, e = 6

0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

**Output:** Yes

**Explanation:**

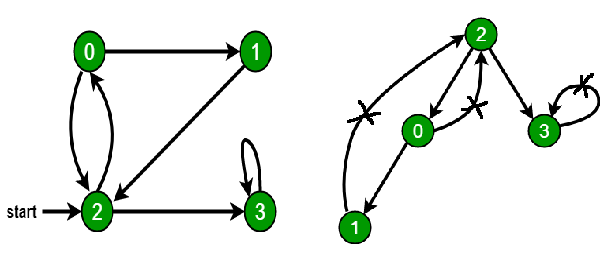
Diagram:



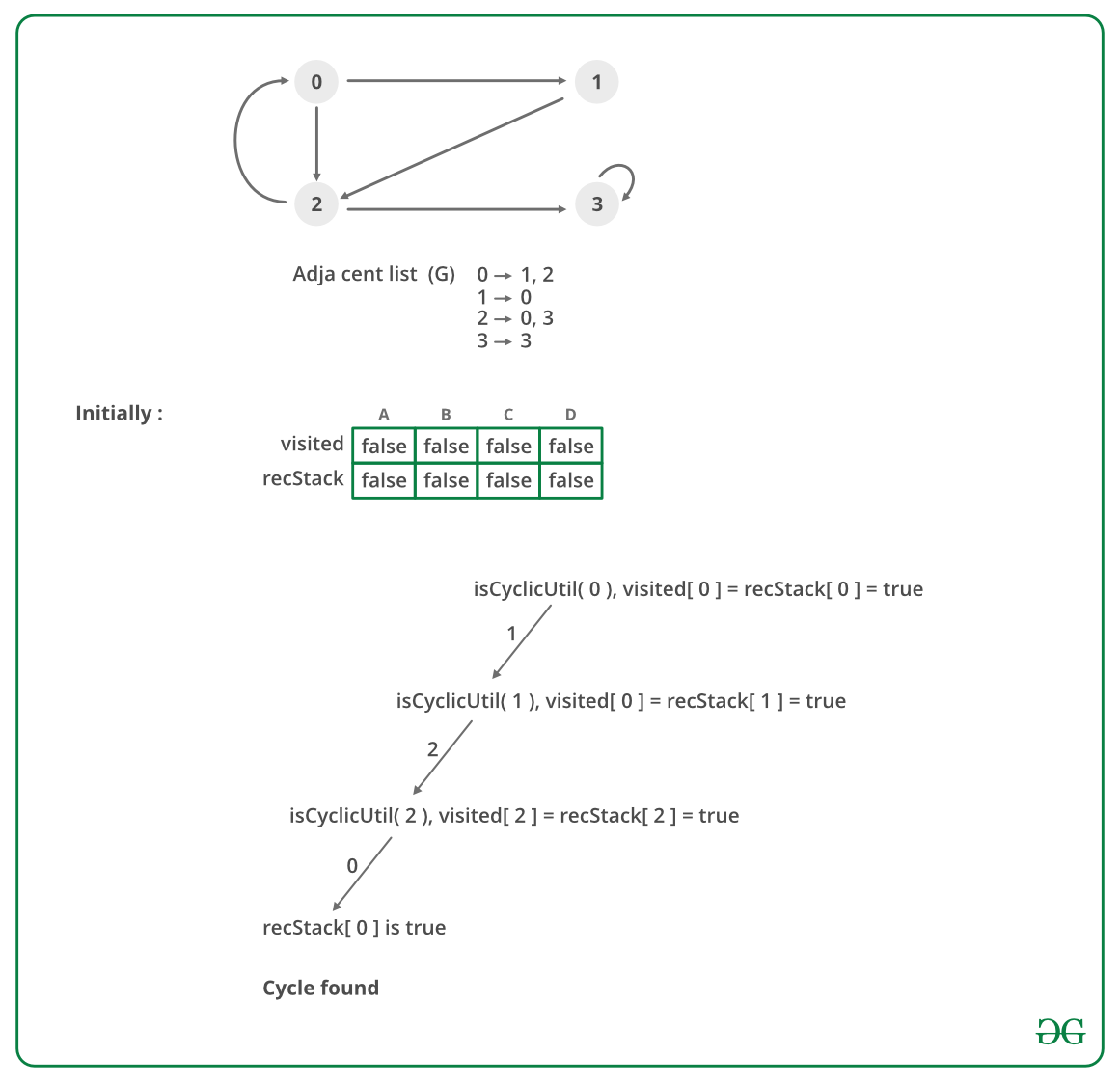
The diagram clearly shows a cycle 0 -> 2 -> 0

**Solution using**[**Depth First Search or DFS**](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/)

* **Approach:** Depth First Traversal can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a [back edge](http://en.wikipedia.org/wiki/Depth-first_search#Output_of_a_depth-first_search) present in the graph. 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. In the following graph, there are 3 back edges, marked with a cross sign. We can observe that these 3 back edges indicate 3 cycles present in the graph.



* For a disconnected graph, Get the DFS forest as output. To detect cycle, 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. The edge that connects the current vertex to the vertex in the recursion stack is a back edge. Use**recStack[]** array to keep track of vertices in the recursion stack.  
  **Dry run of the above approach:**



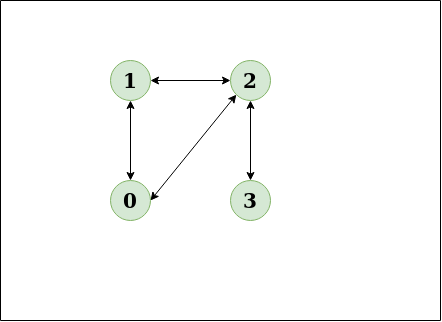
* **Algorithm:**
  1. Create the graph using the given number of edges and vertices.
  2. Create a recursive function that initializes the current index or vertex, visited, and recursion stack.
  3. Mark the current node as visited and also mark the index in recursion stack.
  4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, If the recursive function returns true, return true.
  5. If the adjacent vertices are already marked in the recursion stack then return true.
  6. Create a wrapper class, 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.

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/cycle/DetectCycleInDirectedGraph.java>

**Detect cycle in an undirected graph**

Given an undirected graph, how to check if there is a cycle in the graph?   
**Example,**

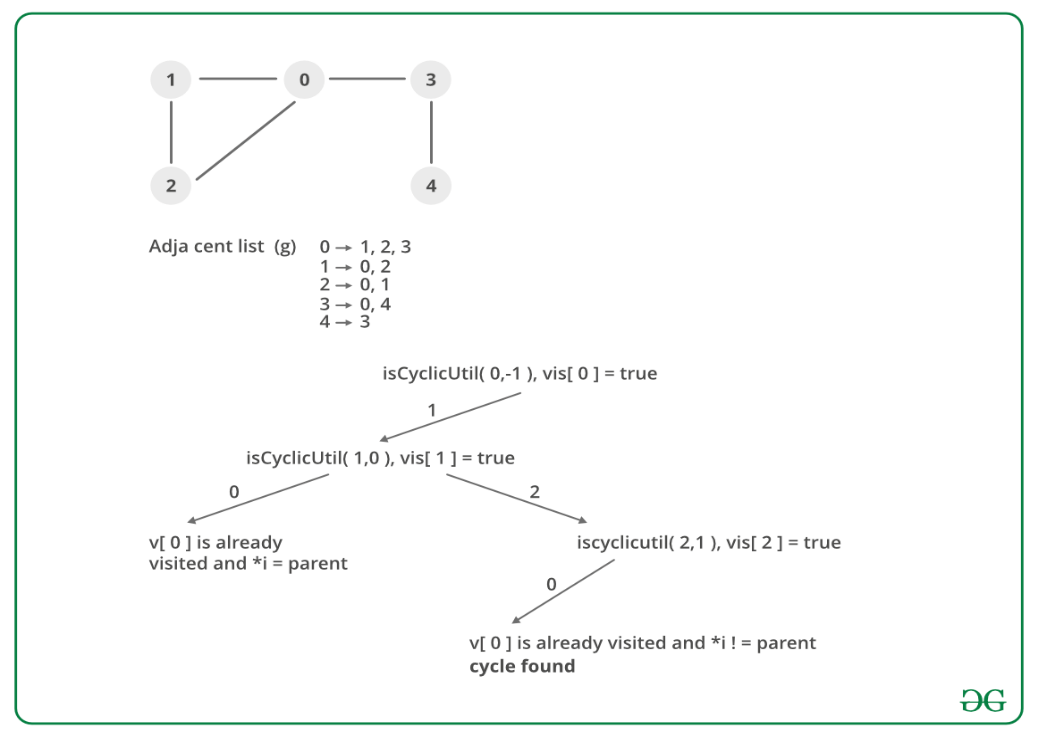
***Input:****n = 4, e = 4****Output:****Yes****Explanation:****0 1, 1 2, 2 3, 0 2   
Diagram:*

**

**Approach:** Run a DFS from every unvisited node. [Depth First Traversal](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/) can be used to detect a cycle in a Graph. DFS for a connected graph produces a tree. There is a cycle in a graph only if there is a back edge present in the graph. A back edge is an edge that is joining a node to itself (self-loop) or one of its ancestors in the tree produced by DFS.   
To find the back edge to any of its ancestors keep a visited array and if there is a back edge to any visited node then there is a loop and return true.  
**Algorithm:**

1. Create the graph using the given number of edges and vertices.
2. Create a recursive function that have current index or vertex, visited array and parent node.
3. Mark the current node as visited.
4. Find all the vertices which are not visited and are adjacent to the current node. Recursively call the function for those vertices, If the recursive function returns true return true.
5. If the adjacent node is not parent and already visited then return true.
6. Create a wrapper class, that calls the recursive function for all the vertices and if any function returns true, return true.
7. Else if for all vertices the function returns false return false.

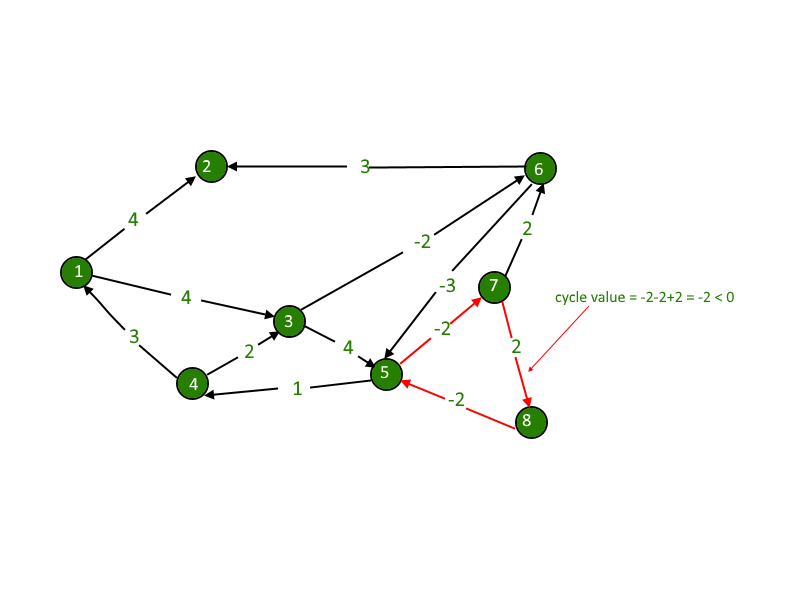
**Dry Run:**



<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/cycle/DetectCycleInUndirectedGraph.java>

**Detect a negative cycle in a Graph | (Bellman Ford)**

We are given a directed graph. We need to compute whether the graph has a negative cycle or not. A negative cycle is one in which the overall sum of the cycle becomes negative.



Negative weights are found in various applications of graphs. For example, instead of paying cost for a path, we may get some advantage if we follow the path.

**Examples:**

Input : 4 4

0 1 1

1 2 -1

2 3 -1

3 0 -1

Output : Yes

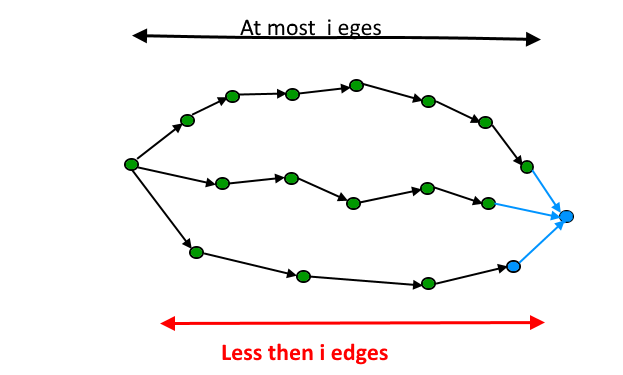
The graph contains a negative cycle.

The idea is to use [Bellman-Ford Algorithm](https://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/).

Below is an algorithm to find if there is a negative weight cycle reachable from the given source.  
**1)** Initialize distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is the source vertex.  
**2)** This step calculates the shortest distances. Do the following |V|-1 times where |V| is the number of vertices in the given graph.   
**a)** Do the following for each edge u-v.  
     **b)**If dist[v] > dist[u] + weight of edge uv, then update dist[v].   
**c)**dist[v] = dist[u] + weight of edge uv.  
**3)** This step reports if there is a negative weight cycle in the graph. Do the following for each edge u-v   
     **a)**If dist[v] > dist[u] + weight of edge uv, then the “Graph has a negative weight cycle”

The idea of step 3 is, step 2 guarantees the shortest distances if the graph doesn’t contain a negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle.

**How does it work?**   
As discussed, the [Bellman-Ford algorithm](https://www.geeksforgeeks.org/dynamic-programming-set-23-bellman-ford-algorithm/), for a given source, first calculates the shortest distances which have at most one edge in the path. Then, it calculates the shortest paths with at-most 2 edges, and so on. After the i-th iteration of the outer loop, the shortest paths with at most i edges are calculated. There can be a maximum |V| – 1 edge on any simple path, that is why the outer loop runs |v| – 1 time. If there is a negative weight cycle, then one more iteration would give a short route.

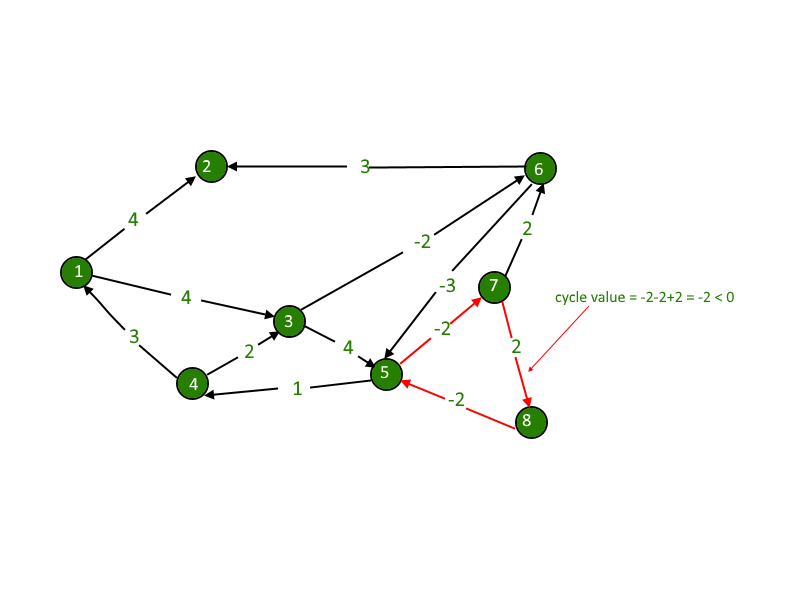


**How to handle a disconnected graph (If the cycle is not reachable from the source)?**   
The above algorithm and program might not work if the given graph is disconnected. It works when all vertices are reachable from source vertex 0.  
To handle disconnected graphs, we can repeat the process for vertices for which distance is infinite.

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/cycle/DetectNegativeCycleBelmanFord.java>

**Detecting negative cycle using Floyd Warshall**

We are given a directed graph. We need compute whether the graph has negative cycle or not. A negative cycle is one in which the overall sum of the cycle comes negative



Negative weights are found in various applications of graphs. For example, instead of paying cost for a path, we may get some advantage if we follow the path.  
Examples:

Input : 4 4

0 1 1

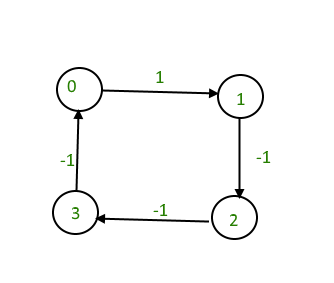
1 2 -1

2 3 -1

3 0 -1

Output : Yes

The graph contains a negative cycle.



We have discussed [Bellman Ford Algorithm](https://www.geeksforgeeks.org/detect-negative-cycle-graph-bellman-ford/) based solution for this problem.  
In this post, [Floyd Warshall Algorithm](https://www.geeksforgeeks.org/dynamic-programming-set-16-floyd-warshall-algorithm/) based solution is discussed that works for both connected and disconnected graphs.  
Distance of any node from itself is always zero. But in some cases, as in this example, when we traverse further from 4 to 1, the distance comes out to be -2, i.e. distance of 1 from 1 will become -2. This is our catch, we just have to check the nodes distance from itself and if it comes out to be negative, we will detect the required negative cycle.

<https://github.com/hareramcse/Datastructure/blob/master/Graph/src/com/hs/cycle/DetectNegativeCycleFloyedWarshall.java>