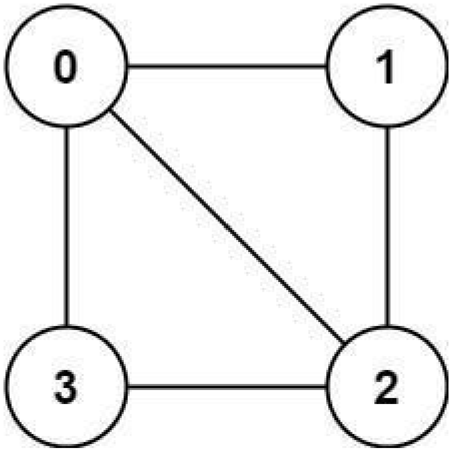
# EXPERIMENT -6
# GRAPHS

**Name:- Nikhil Kumar** **UID:- 20BCS1817**
**Branch:- CSE** **Section:- 20BCS_716/B**
**Semester:- 6** **Date of performance:- 10/04/2023**
**Subject: CC-II**

## 1. Objective

There is an undirected graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v.
A graph is bipartite if the nodes can be partitioned into two independent sets A and B such that every edge in the graph connects a node in set A and a node in set B.
Return true if and only if it is bipartite.

Example 1:



yInput: graph = [[1,2,3],[0,2],[0,1,3],[0,2]]

Output: false

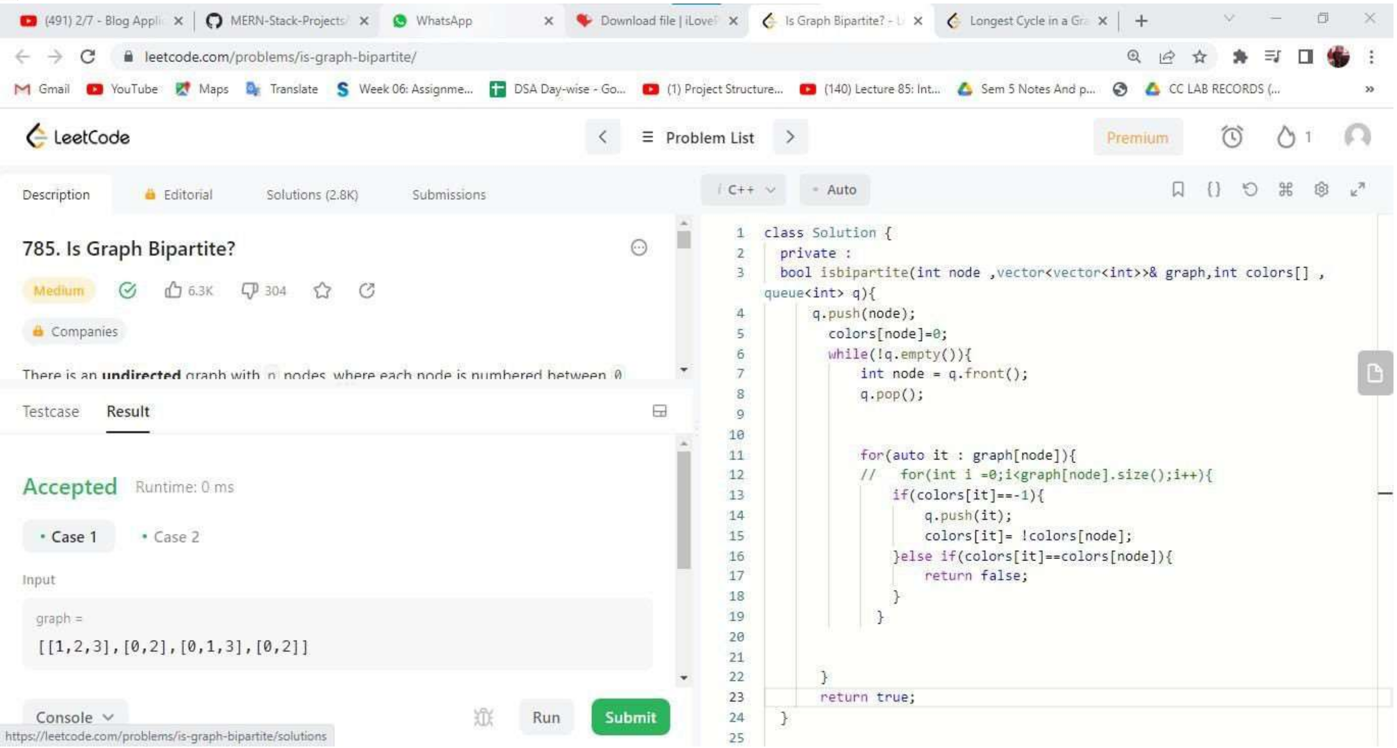
Explanation: There is no way to partition the nodes into two independent sets such that every edge connects a node in one and a node in the other.

### 2. Script and Output

**Code:**

```
class Solution {
  private :
  bool isbipartite(int node ,vector<vector<int>>& graph,int colors[] , queue<int> q){
      q.push(node);
        colors[node]=0;
        while(!q.empty()){ int
        node = q.front();
        q.pop();



            for(auto it : graph[node]){
            //  for(int i =0;i<graph[node].size();i++){ if(colors[it]==-1){
                    q.push(it); colors[it]=
                    !colors[node];
                }else if(colors[it]==colors[node]){
                    return false;
                }
              }



       }
       return true;
  }
public:
    bool isBipartite(vector<vector<int>>& graph)
        { int n = graph.size(); int colors[n];
        for(int i =0;i<n;i++) colors[i]=-1;
        queue<int> q; q.push(0);

      for(int i =0;i<n;i++){ if(colors[i]==-1){ if(isbipartite(i ,
          graph ,colors,q)==false) return false; }
      }
return true;


    }
};
```

**Output:**



785. Is Graph Bipartite?

There is an **undirected** graph with n nodes, where each node is numbered between 0

Accepted Runtime: 0 ms

Input

graph =
[[1,2,3],[0,2],[0,1,3],[0,2]]

```
class Solution {
  private :
  bool isbipartite(int node ,vector<vector<int>>& graph,int colors[] ,
queue<int> q){
        q.push(node);
          colors[node]=0;
          while(!q.empty()){
              int node = q.front();
              q.pop();


              for(auto it : graph[node]){
              //   for(int i =0;i<graph[node].size();i++){
                  if(colors[it]==-1){
                      q.push(it);
                      colors[it]= !colors[node];
                  }else if(colors[it]==colors[node]){
                      return false;
                  }
                }


         }
         return true;
  }

```

## Ques 6.2

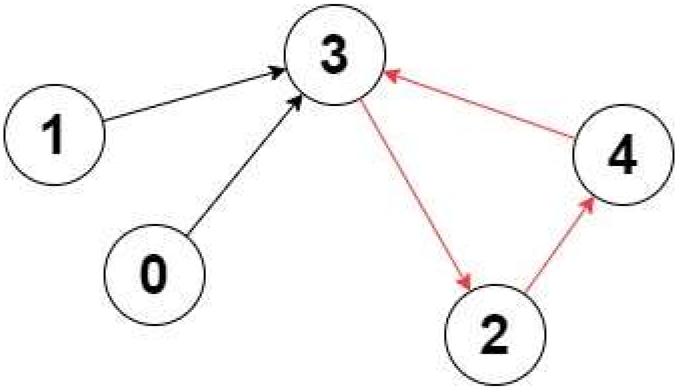
### 1. Objective

You are given a directed graph of n nodes numbered from 0 to n - 1, where each node has at most one outgoing edge.

The graph is represented with a given 0-indexed array edges of size n, indicating that there is a directed edge from node i to node edges[i]. If there is no outgoing edge from node i, then edges[i] == -1.

Return the length of the longest cycle in the graph. If no cycle exists, return -1.

Example 1:



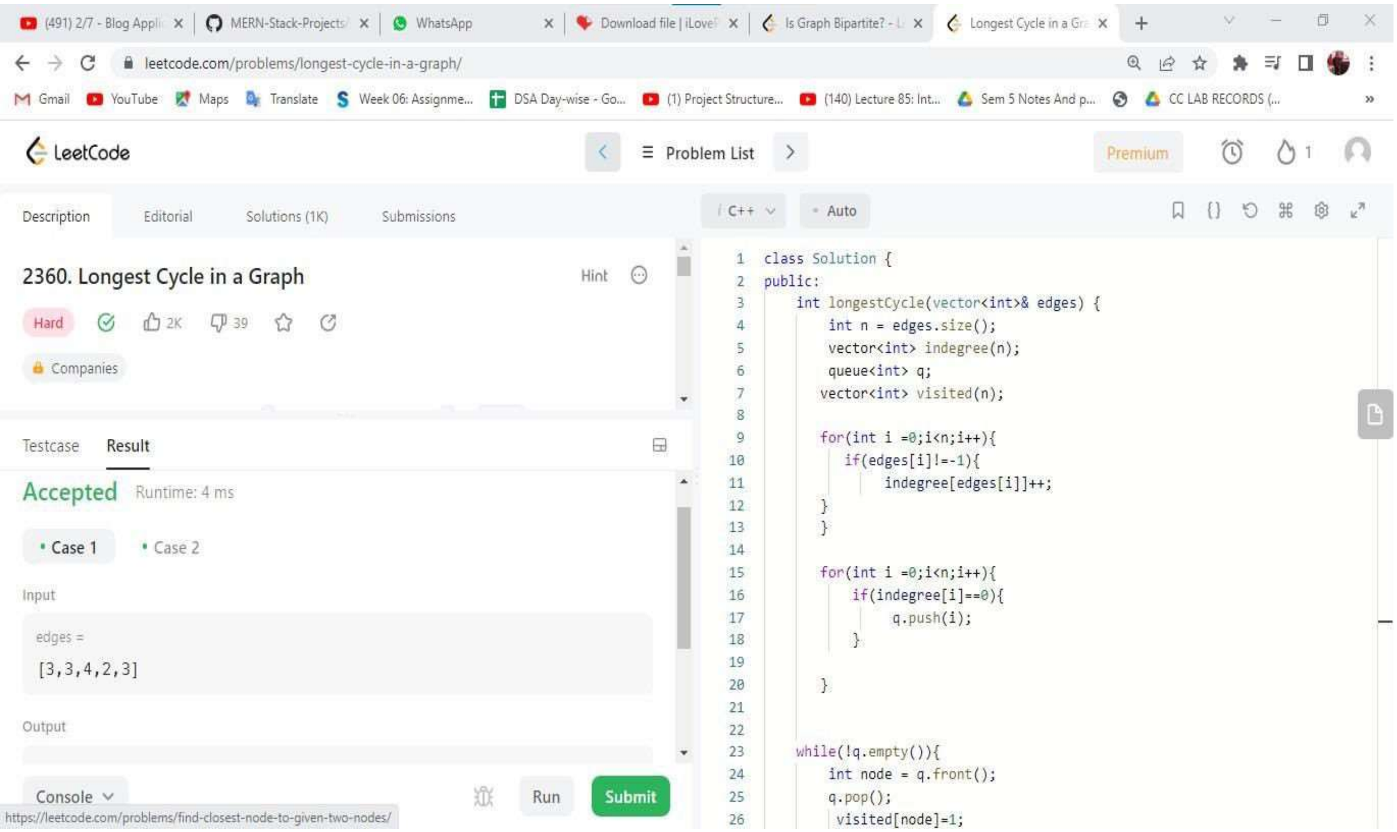
Input: edges = [3,3,4,2,3]Output: 3Explanation: The longest cycle in the graph is the cycle: 2 -> 4 -> 3 -> 2.

## 2. Script and Output

Code:

```
    class Solution {
public:
    int longestCycle(vector<int>& edges) {
        int n = edges.size(); vector<int>
        indegree(n);
       queue<int> q; vector<int>
       visited(n);

       for(int i =0;i<n;i++){
          if(edges[i]!=-1){
          indegree[edges[i]]++;
       }
       }

       for(int i =0;i<n;i++){
           if(indegree[i]==0){
                q.push(i);
           }

       }

while(!q.empty()){

        int node = q.front();
        q.pop();
        visited[node]=1;
        if(edges[node]!=-1 ){
indegree[edges[node]]--;
            if(indegree[edges[node]]==0)
            {
                 q.push(edges[node]);
        }
        }
    }  int ans =-1;
    for(int i =0;i<n;i++){
        if(visited[i]==0){
        visited[i]=1; int cnt =1;
        int neighbor = edges[i];
        while(neighbor!=i){
                visited[neighbor]=1;
                cnt++;
                neighbor=edges[neighbor];
            } ans = max(cnt,ans);


        }


    } return
     ans;
    }
};
```

**Output:**



2360. Longest Cycle in a Graph

Accepted Runtime: 4 ms

Input

edges =

[3,3,4,2,3]

Output

```
class Solution {
public:
    int longestCycle(vector<int>& edges) {
        int n = edges.size();
        vector<int> indegree(n);
        queue<int> q;
       vector<int> visited(n);

       for(int i =0;i<n;i++){
          if(edges[i]!=-1){
               indegree[edges[i]]++;
       }
       }

       for(int i =0;i<n;i++){
           if(indegree[i]==0){
                q.push(i);
           }

       }


    while(!q.empty()){
        int node = q.front();
        q.pop();
         visited[node]=1;
```