



**DEPARTMENT OF**

**COMPUTER SCIENCE AND ENGINEERING**

Discover. Learn. Empower.

## EXPERIMENT -6

### GRAPHS

**Name:- Nikhil Kumar**

**Branch:- CSE**

**Semester:- 6**

**Subject: CC-II**

**UID:- 20BCS1817**

**Section:- 20BCS\_716/B**

**Date of performance:- 10/04/2023**

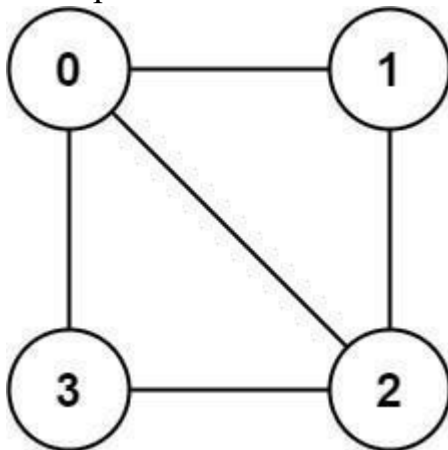
### **1. Objective**

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:



Input: `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:

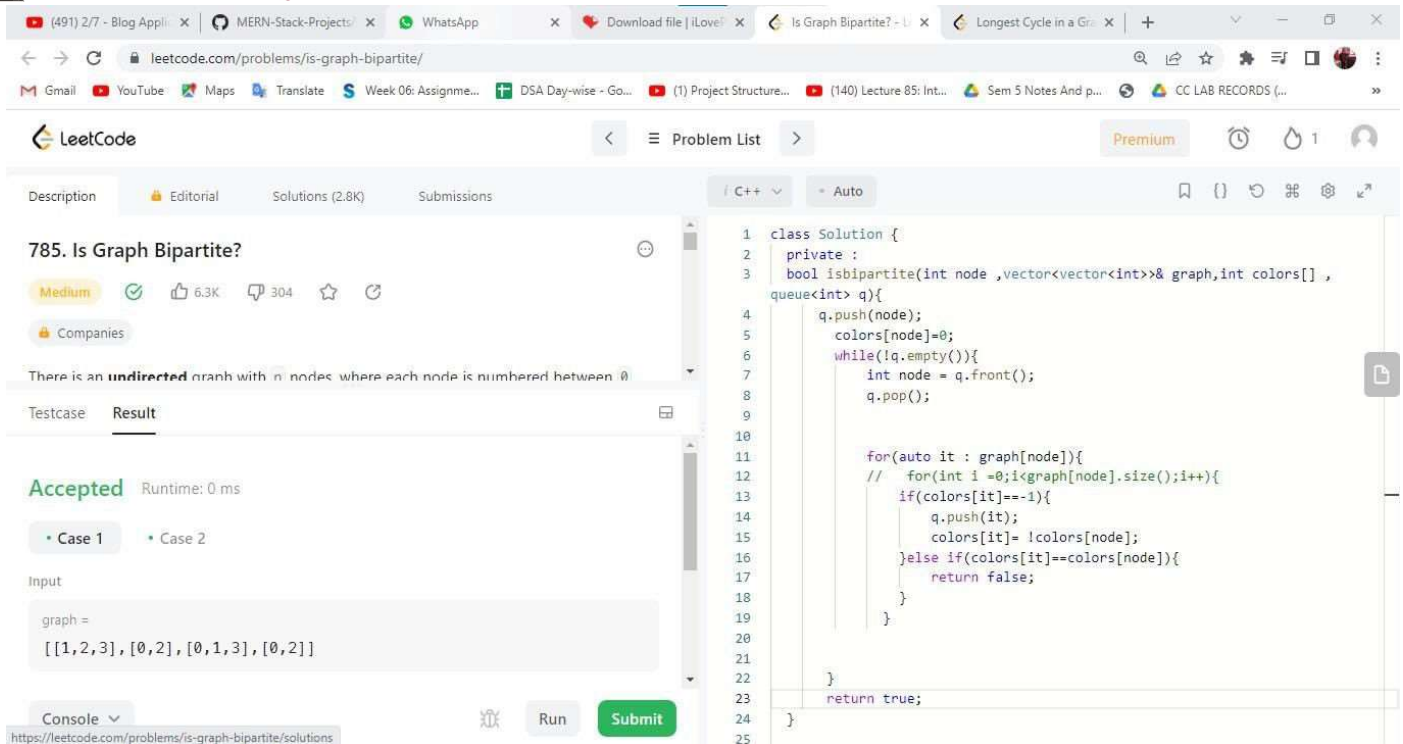
```
class Solution {
private :
bool isbipartite(int node ,vector<vector<int>>& graph,int colors[] , queue<int> q){
    q.push(node);
    colors[node]=0;
    while(!q.empty()){ int
        node = q.front();
        q.pop();

        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:



The screenshot shows the LeetCode interface for problem 785, "Is Graph Bipartite?". The problem description states: "There is an undirected graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D integer array edges where edges[i] = [ai, bi] indicates an undirected edge between nodes ai and bi." The input provided is graph = [[1,2,3], [0,2], [0,1,3], [0,2]]. The solution is implemented in C++ using a BFS approach to check if the graph is bipartite. The code uses a queue to traverse the graph and assigns colors (0 or 1) to nodes. If a node is already colored and its color differs from the current node's color, the graph is not bipartite. If the graph is bipartite, the function returns true; otherwise, it returns false.

```

1 class Solution {
2 private:
3     bool isBipartite(int node, vector<vector<int>>& graph, int colors[],
4         queue<int> q) {
5         q.push(node);
6         colors[node] = 0;
7         while (!q.empty()) {
8             int node = q.front();
9             q.pop();
10
11             for (auto it : graph[node]) {
12                 // for(int i = 0; i < graph[node].size(); i++) {
13                 if (colors[it] == -1) {
14                     q.push(it);
15                     colors[it] = 1 - colors[node];
16                 } else if (colors[it] == colors[node]) {
17                     return false;
18                 }
19             }
20         }
21         return true;
22     }
23 }

```

## Ques 6.2

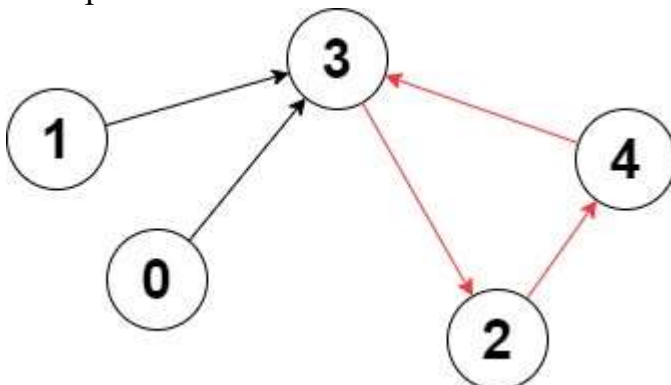
### 1. Objective

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: 3 Explanation: 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];
                }
            }
        }
        return ans;
    }
};
```



DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

Discover. Learn. Empower.

```
    } ans = max(cnt,ans);
```

```
}
```

```
} return
```

```
ans;
```

```
}
```

```
};
```

**Output:**

2360. Longest Cycle in a Graph

Hard 2K 39

Companies

Testcase Result

Accepted Runtime: 4 ms

Case 1 Case 2

Input

edges = [3,3,4,2,3]

Output

Console

Run Submit

```
1 class Solution {
2 public:
3     int longestCycle(vector<int>& edges) {
4         int n = edges.size();
5         vector<int> indegree(n);
6         queue<int> q;
7         vector<int> visited(n);
8
9         for(int i =0;i<n;i++){
10             if(edges[i]!=-1){
11                 indegree[edges[i]]++;
12             }
13         }
14
15         for(int i =0;i<n;i++){
16             if(indegree[i]==0){
17                 q.push(i);
18             }
19         }
20
21         while(!q.empty()){
22             int node = q.front();
23             q.pop();
24             visited[node]=1;
25         }
26     }
```