q.1)

You are given with a connected and undirected simple graph with N vertices and M edges. Your task is to direct each edge in one of two possible directions in such a way that the indegrees of all vertices of
the resulting graph are even. The indegree of a vertex is the number of
edges directed to that vertex from another vertex. Find one possible way
to direct them or determine that it is impossible under the given conditions. The graph on the input is connected, does not contain multiple
edges or self-loops.

For each test case (Output):

If a valid way to direct the edges does not exist, print a single line containing one integer −1.

Otherwise, print a single line containing M space-separated integers.

For each valid i, the i-th of these integers should be 0 if edge i is directed from u i to v i or 1 if it is directed from v i to u i .

Example Input

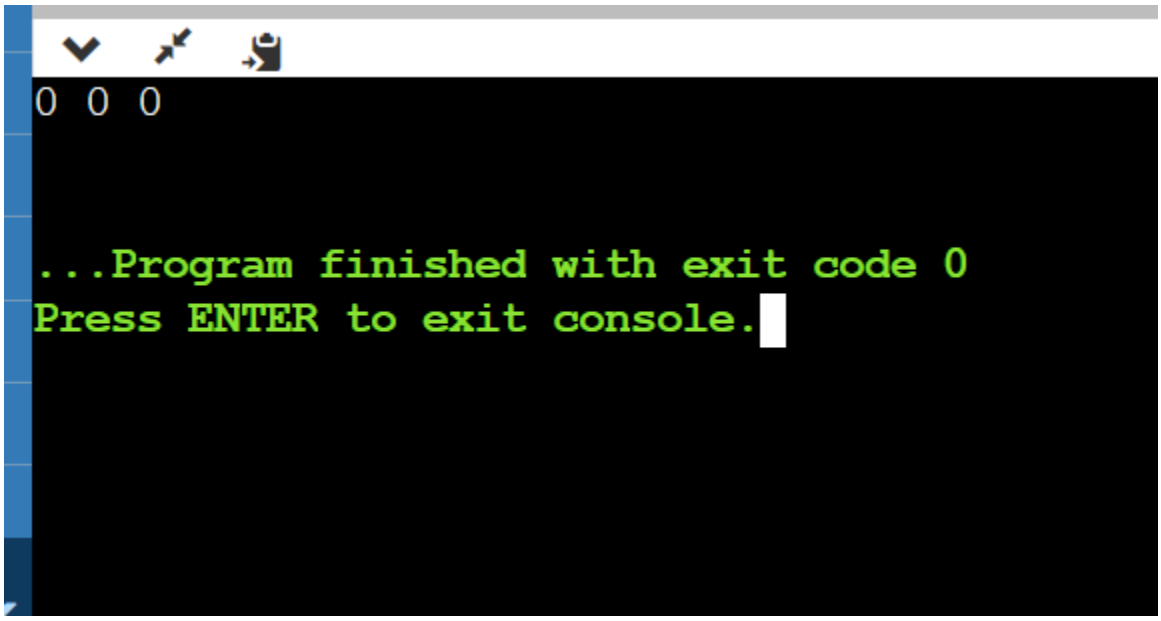
4 4 N M
1 2
1 3
2 4
3 4 Output : 0 0 1 1
3 3 N M
1 2
2 3
1 3
Output: -1

Code:

```
def solve(N, M, edges):
    graph = [[] for _ in range(N)]
    indegrees = [0] * N

    for u, v in edges:
        graph[u - 1].append(v - 1)
        graph[v - 1].append(u - 1)
        indegrees[u - 1] += 1
        indegrees[v - 1] += 1

    directed_edges = [0] * M
    queue = []

    for i in range(N):
        if indegrees[i] % 2 == 1:
            queue.append(i)

    for u in queue:
        for v in graph[u]:
            if indegrees[u] % 2 == 1 and indegrees[v] % 2 == 1:
                directed_edges[graph[u].index(v)] = 1
                indegrees[u] -= 1
                indegrees[v] += 1

    for i in range(M):
        if directed_edges[i] == 0 and indegrees[edges[i][0] - 1] % 2 == 1:
            directed_edges[i] = 1

    if sum(indegrees) % 2 == 0:
        return directed_edges
    else:
        return [-1]

# Example input
N = 3
M = 3
edges = [(1, 2), (2, 3), (1, 3)]

result = solve(N, M, edges)
if result[0] == -1:
    print(-1)
```

```
else:
    print(" ".join(map(str, result)))
```

Output:



q.2)

There are a total n tasks you must pick, labelled from 0 to n-1. Some
tasks may have pre-requisites and for example to pick task 0 you have to
first pick task 1, which is expressed as a pair [0, 1].
Write a function
bool canFinish(int tasks, int [][] prerequsites)
that return true or false if it is possible for you to finish all tasks or not.

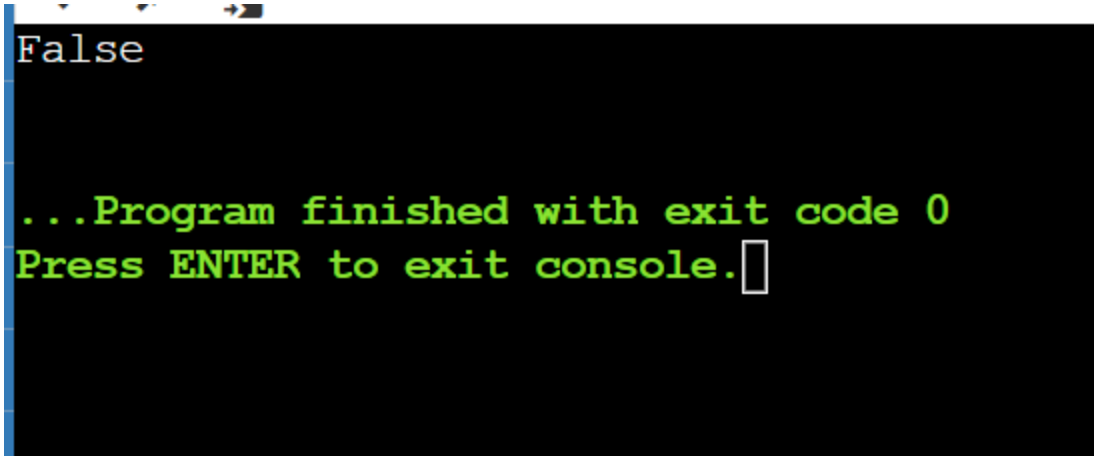
Input: tasks = 2, pre-requsites = [[0,1], [1,0]]
Output: False

Input: tasks=3, pre-requsites = [[1,0], [0,2]]
Output: True

Code:

```
def canFinish(tasks, prerequisites):
    graph = [[] for _ in range(tasks)]
    in_degrees = [0] * tasks

    for u, v in prerequisites:
        graph[v].append(u)
        in_degrees[u] += 1

    queue = []
    for i in range(tasks):
        if in_degrees[i] == 0:
            queue.append(i)

    visited = 0
    while queue:
        node = queue.pop(0)
        visited += 1
        for neighbor in graph[node]:
            in_degrees[neighbor] -= 1
            if in_degrees[neighbor] == 0:
                queue.append(neighbor)

    return visited == tasks

# Test cases
tasks_1 = 2
prerequisites_1 = [[0, 1], [1, 0]]
print(canFinish(tasks_1, prerequisites_1))
```

Output:


False
...Program finished with exit code 0
Press ENTER to exit console.