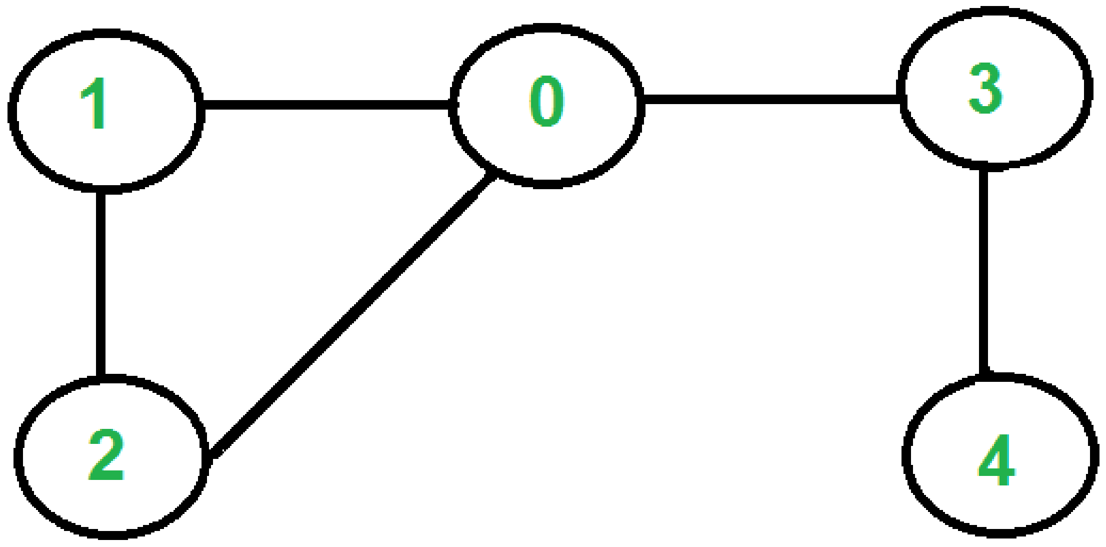
Eulerian path and circuit for undirected graph

Given an **undirected connected graph** with **v** nodes, and **e** edges, with adjacency list **adj.** We need to write a function thatreturns 2 if the graph contains an **eulerian circuit or cycle**, else if the graph contains an **eulerian path,** returns 1, otherwise, returns 0.

A graph is said to be Eulerian if it contains an **Eulerian Cycle,** a cycle that visits every edge exactly once and starts and ends at the same vertex.
If a graph contains an **Eulerian Path,** a path that visits every edge exactly once but starts and ends at different vertices, it is called **Semi-Eulerian**.

Examples:

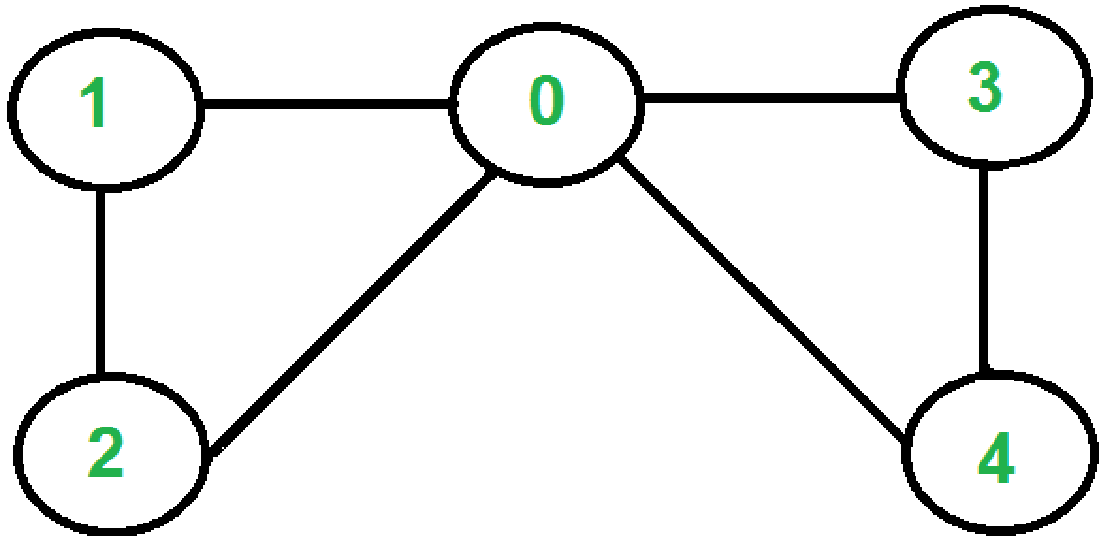
Input:



The graph has Eulerian Paths, for example "4 3 0 1 2 0", but no Eulerian Cycle. Note that there are two vertices with odd degree (4 and 0)

Output: 1

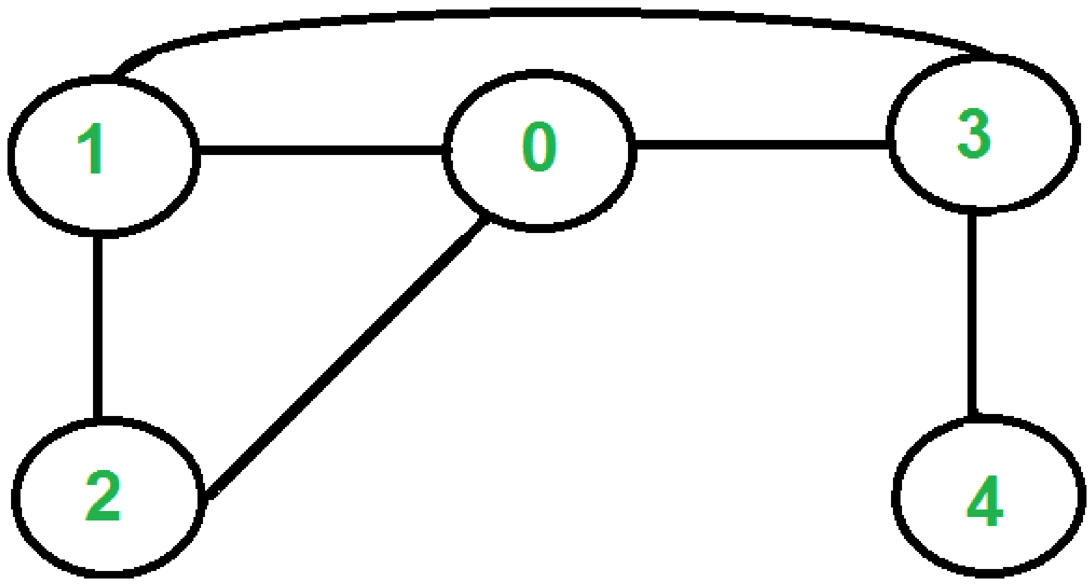
Input:



The graph has Eulerian Cycles, for example "2 1 0 3 4 0 2"
Note that all vertices have even degree

Output: 2

Input:



The graph is not Eulerian. Note that there are four vertices with odd degree (0, 1, 3 and 4)

Output: 0

The problem can be framed as follows: "Is it possible to draw a graph without lifting your pencil from the paper and without retracing any edge?"
At first glance, this may seem similar to the **Hamiltonian Path** problem, which is **NP-complete** for general graphs. However, determining whether a graph has an Eulerian Path or Cycle is much more efficient: It can be solved in **O(v + e) time.**

Approach:

The idea is to use some key **properties** of undirected graphs that help determine whether they are **Eulerian** (i.e., contain an Eulerian Path or Cycle) or not.

Eulerian Cycle

A graph has an Eulerian Cycle **if and only if** the below two conditions are **true**

- All vertices with non-zero degree are part of a single connected component. (We ignore isolated vertices— those with zero degree — as they do not affect the cycle.)
- Every vertex in the graph has an even degree.

Eulerian Path

A graph has an Eulerian Path **if and only if** the below two conditions are **true**

- All vertices with non-zero degree must belong to the same connected component. (Same as Eulerian Cycle)
- Exactly **Zero or Two** Vertices with **Odd** Degree:
  - If **zero** vertices have odd degree → Eulerian **Cycle** exists (which is also a path).
  - If **two** vertices have odd degree → Eulerian **Path** exists (but not a cycle).
  - If **one** vertex has odd degree → **Not possible** in an undirected graph. (Because the sum of all degrees in an undirected graph is always even.)

Note: A graph with no edges is **trivially Eulerian**. There are no edges to traverse, so by definition, it satisfies both Eulerian Path and Cycle conditions.

How Does This Work?

- In an **Eulerian Path**, whenever we enter a vertex (except start and end), we must also leave it. So all **intermediate vertices** must have even degree.
- In an **Eulerian Cycle**, since we start and end at the same vertex, **every** vertex must have even degree. This ensures that every entry into a vertex can be paired with an exit.

Steps to implement the above idea:

- Create an adjacency list to represent the **graph** and initialize a **visited** array for DFS traversal.
- Perform **DFS** starting from the first vertex having **non-zero degree** to check graph connectivity.
- After DFS, ensure all **non-zero degree** vertices were visited to confirm the graph is connected.
- Count the number of vertices with **odd degree** to classify the graph as Eulerian or not.

- If all degrees are even, the graph has an **Eulerian Circuit**; if exactly two are odd, it's a **Path**.
- If more than two vertices have odd degree or graph isn't connected, it's **not Eulerian**.
- Return 2 for Circuit, 1 for Path, and 0 when graph fails Eulerian conditions.

Time Complexity: O(v + e), DFS traverses all vertices and edges to check connectivity and degree.
Space Complexity: O(v + e), Space used for visited array and adjacency list to represent graph.