**Eulerian Path and Cycles**

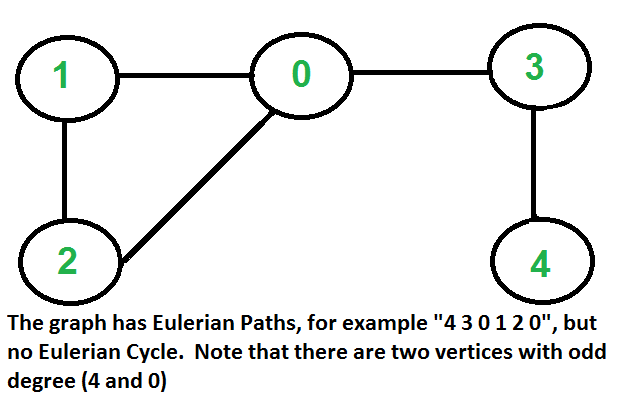
Eulerian Path: A path that traverses every edges in the graph **exactly once** (might end at another node)

**For an undirected graph: exist if**

* *Every vertex should have an even degree or only two vertices should have odd degrees.*

**For a directed graph: exist if**

* *Each vertex should have the same in-degree and out-degree except for two of them.*
* *One of these vertex will be the start vertex has one more out-going edge than in-going edges. The other one will be the end vertex which has one more in-going edge than out-going edges.*



Eulerian Cycle: A path that traverses every edges in the graph **exactly once** and come back to the start

**For an undirected graph: exist if**

* *Each vertex should have an even degree.*

**For a directed graph: exist if**

* *Every vertex should have equal in-degree and out-degree edges.*

A screen shot of a computer program

Description automatically generated with medium confidence\*Algorithm for **finding Eulerian Cycles** - Hierolzer Algorithm

1. We can choose any arbitrary vertex as a starting point,
2. We follow the outgoing edges of the vertex that we haven’t followed before. We can follow whichever edge we want, the only rule is not to follow previously traversed ones.
3. We should apply Step 2 until we are stuck. At some point, we will visit a vertex and there will be no edges to follow.
4. Push the vertex that we stuck to the top of the stack data structure which holds the Eulerian Cycle (Pop out of exploring path)
5. Backtrack from this vertex to the previous one
6. If there are edges to follow, we have to return to Step 2 (do not include the edges we went on)
7. If there are no vertices left to traverse, now the stack holds the complete Eulerian Cycle, and we are done. Otherwise, go back to Step 5.

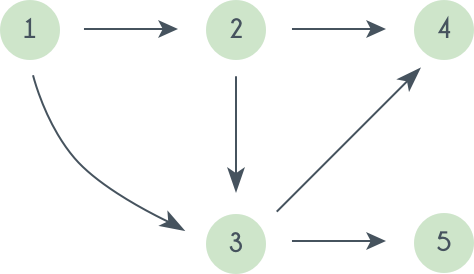
**Detecting Cycle**

* 1. **Undirected Graph**

Using DFS 🡪 Simply need to check if we visited a node again. If yes, there is a cycle

* 1. **Directed Graph (Directed Acyclic Graph)**

We cannot use the above algorithm anymore, because it will say that in this case, there's a cycle while there isn't (can visit 4 **twice** without creating a cycle)



🡪 We'll need **2 visited arrays**:

- One for **"outer loop" visited**, which marks that we have already done DFS "starting" from this node.

- Another for **"dfs" visited**, which marks the visited for current DFS traversal. This will detect if there is a cycle or not

\*\*\*\* NOTE: Due to the difference from Undirected Graph, a node may be travelled through by multiple DFS paths but not create a cycle, so we MUST mark the node as DFS **unvisited** when we return from its DFS recursive call