**Eulerian Paths**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

Eulerian paths are trails in a graph that visit each edge exactly once but may not necessarily return to the starting vertex. To find an Eulerian path in a graph, we look for a vertex with an odd degree (the number of edges incident to it). If there are exactly two vertices with an odd degree, one of them will be the starting vertex, and the other will be the ending vertex of the Eulerian path. If there are no vertices with an odd degree, the graph may have an Eulerian circuit, which is a closed path that visits all edges and vertices exactly once.

Eulerian circuits, also known as Eulerian cycles, are a special case of Eulerian paths where the path starts and ends at the same vertex. This means that an Eulerian circuit traverses all edges and vertices of the graph exactly once and forms a closed loop. A graph has an Eulerian circuit if and only if every vertex has an even degree..

**Use cases:**

The applications of Eulerian paths and circuits are vast and varied. In transportation planning, these concepts help optimize routes for vehicles and goods distribution, ensuring efficient logistics and reduced costs. In computer networks and communication systems, they play a vital role in designing algorithms for data routing and transmission, improving network performance and reducing congestion. In biology, Eulerian paths and circuits are used in DNA sequencing and genome assembly, aiding in understanding genetic information and evolutionary relationships. Additionally, they find applications in electrical engineering for circuit analysis and design, as well as in social network analysis and various other fields where interconnected structures need to be efficiently navigated.

**Conditions of Existence:**

To find Eulerian paths and circuits, there are rules that the graph has to conform with. Here’s a table that explains everything:

|  |  |  |
| --- | --- | --- |
|  | **Eulerian Circuit** | **Eulerian Path** |
| **Undirected Graph** | Every node has an even degree. | Either it’s an Eulerian circuit or there are exactly two nodes that have odd degree |
| **Directed Graph** | Every node has equal indegree and outdegree | At most one vertex has (outdegree) – (indegree)=1 and at most one vertex has (indegree) – (outdegree) = 1 and all other vertices have equal in and out degrees |

A degree of a node is the number of edges it has, a indegree is the number of input edges it has and the outdegree is the number output edges it has. Note that the graph should be a single component.

**Algorithm For Finding an Eulerian Path and/or Circuit:**

Before finding the Eulerian path in a graph, we need to make sure it exists first, then we should find the right starting and ending nodes, the starting node is the node that has an extra outgoing edge and the ending node is the one that has one extra ingoing edge. If all nodes have equal in and out degrees then we have an Eulerian circuit in which we can start at any random node. In this algorithm we’re going to use an edited version of DFS algorithm to find the path. This algorithm runs at a time complexity of .

1. #Variables and constants

2. graph = adjacency list

3. def degrees(graph, n):

4.     indegrees = [0] \* n

5.     outdegrees = [0] \* n

6.     for i in range(n):

7.         outdegrees[i] = len(graph[i])

8.         for el in graph[i]:

9.             indegrees[el] += 1

10.     return indegrees, outdegrees

11.

12. def circuitexists(indegrees, outdegrees):

13.     for i,o in zip(indegrees, outdegrees):

14.         if not i==o: return False

15.     return True

16.

17. def pathexists(indegrees, outdegrees):

18.     count1 = count2 = 0

19.     for i,o in zip(indegrees, outdegrees):

20.         if count1>1 or count2>1 or abs(i-o)>1: return False

21.         count1 += i-o == 1

22.         count2 += o-i == 1

23.     return True

24.

25. def findstart(indegrees, outdegrees, n):

26.     start = 0

27.     for i in range(n):

28.         if outdegrees[i] - indegrees[i] == 1: return i

29.         if outdegrees[i]>0: start = i

30.     return start

31.

32. def dfs(graph, i, path, outdegrees):

33.     while not outdegrees[i] == 0:

34.         outdegrees[i] -= 1

35.         dfs(graph, graph[i][outdegrees[i]] , path, outdegrees)

36.     path.insert(0, i)

37.

38. def findpath(graph):

39.     path = []

40.     n = len(graph)

41.     indegrees, outdegrees = degrees(graph, n)

42.     if pathexists(indegrees, outdegrees):

43.         dfs(graph, findstart(indegrees, outdegrees, n), path, outdegrees)

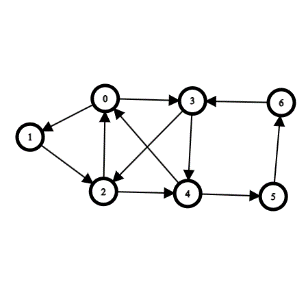
44.     return path

45.

46. print(findpath(graph))

**Example:**

Here’s a small example illustrating an example of input outputs for the Eulerian Circuit Finding Algorithm:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. #Variables and constants

2. graph = [

3.     [1, 3],

4.     [2],

5.     [0, 4],

6.     [2, 4],

7.     [0, 5],

8.     [6],

9.     [3]

10. ]

11. def degrees(graph, n):

12.     indegrees = [0] \* n

13.     outdegrees = [0] \* n

14.     for i in range(n):

15.         outdegrees[i] = len(graph[i])

16.         for el in graph[i]:

17.             indegrees[el] += 1

18.     return indegrees, outdegrees

19. def circuitexists(indegrees, outdegrees):

20.     for i,o in zip(indegrees, outdegrees):

21.         if not i==o: return False

22.     return True

23. def pathexists(indegrees, outdegrees):

24.     count1 = count2 = 0

25.     for i,o in zip(indegrees, outdegrees):

26.         if count1>1 or count2>1 or abs(i-o)>1: return False

27.         count1 += i-o == 1

28.         count2 += o-i == 1

29.     return True

30. def findstart(indegrees, outdegrees, n):

31.     start = 0

32.     for i in range(n):

33.         if outdegrees[i] - indegrees[i] == 1: return i

34.         if outdegrees[i]>0: start = i

35.     return start

36. def dfs(graph, i, path, outdegrees):

37.     while not outdegrees[i] == 0:

38.         outdegrees[i] -= 1

39.         dfs(graph, graph[i][outdegrees[i]] , path, outdegrees)

40.     path.insert(0, i)

41. def findpath(graph):

42.     path = []

43.     n = len(graph)

44.     indegrees, outdegrees = degrees(graph, n)

45.     if pathexists(indegrees, outdegrees):

46.         dfs(graph, findstart(indegrees, outdegrees, n), path, outdegrees)

47.     return path

48. print(findpath(graph))

The corresponding output is:

1. Python >> [6, 3, 4, 0, 3, 2, 0, 1, 2, 4, 5, 6]

