Assignment-2

Question:

Write a python program that takes an undirected graph as input add

- Checks whether the graph is Eulerian or not.
- If found Eulerian, then your program should compute and print a Euler tour in the graph using Hierholzer's algorithm.
- Your program should also print the cycles as they are discovered and merged (spliced) with the existing cycle during the execution of the algorithm.

You may use adjacency list or adjacency matrix as the data structure to store the input graph.

Solution:

code.py

```
from collections import defaultdict, deque


class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = defaultdict(list)

    def add_edge(self, u, v):
        self.graph[u].append(v)
        self.graph[v].append(u)

    def is_connected(self):
        visited = [False] * self.V
        start = -1
        for i in range(self.V):
            if len(self.graph[i]) > 0:
                start = i
                break
        if start == -1:
            return True

        queue = deque([start])
        visited[start] = True
        while queue:
            u = queue.popleft()
            for v in self.graph[u]:
                if not visited[v]:
                    visited[v] = True
                    queue.append(v)

        for i in range(self.V):
            if not visited[i] and len(self.graph[i]) > 0:
```

```
                return False
        return True

    # Check for Eular Path
    def is_eulerian(self):
        if not self.is_connected():
            return 0
         odd_degree = sum(1 for i in range(self.V) if len(self.graph[i]) % 2 !=
0)
        if odd_degree == 0:
            return 2
        elif odd_degree == 2:
            return 1
        else:
            return 0

    # If Eular Path is present, then print the Eular Path
    def print_euler_tour(self):
        status = self.is_eulerian()
        if status == 0:
            print("Graph is NOT Eulerian!")
            return
        elif status == 2:
            print("Graph has an Euler Circuit.\n")
        else:
            print("Graph has an Euler Path (but not a circuit).\n")

        print("Running Hierholzer's Algorithm...")
        self._hierholzer()

    def _find_cycle_from(self, start, graph_copy):
        path = [start]
        while True:
            cur = path[-1]
            if not graph_copy[cur]:
                return None
            nxt = graph_copy[cur].pop(0)
            if cur in graph_copy[nxt]:
                graph_copy[nxt].remove(cur)
            if nxt in path:
                idx = path.index(nxt)
                cycle = path[idx:] + [nxt]
                return cycle
            else:
                path.append(nxt)

    def _hierholzer(self):
        graph_copy = {i: list(self.graph[i]) for i in range(self.V)}
        start = 0
        for i in range(self.V):
            if len(graph_copy[i]) > 0:
                start = i
                break

        initial_cycle = self._find_cycle_from(start, graph_copy)
        if not initial_cycle:
            print("\nFinal Euler Tour: []")
            return
```

```
        print("Splicing Cycle:", initial_cycle[::-1])
        main_cycle = initial_cycle[:]

        while True:
            idx_with_extra = None
            for idx in range(len(main_cycle) - 1):
                v = main_cycle[idx]
                if graph_copy[v]:
                    idx_with_extra = idx
                    break

            if idx_with_extra is None:
                break

            v = main_cycle[idx_with_extra]
            new_cycle = self._find_cycle_from(v, graph_copy)
            if not new_cycle:
                break

            print("Splicing Cycle:", new_cycle)

            main_cycle = (
                main_cycle[:idx_with_extra]
                + new_cycle
                + main_cycle[idx_with_extra + 1 :]
            )

        print("\nFinal Euler Tour:", main_cycle)


if __name__ == "__main__":
    V = int(input("Enter number of vertices: "))
    E = int(input("Enter number of edges: "))

    g = Graph(V)
    print("Enter edges (format: u v):")
    for _ in range(E):
        u, v = map(int, input().split())
        g.add_edge(u, v)

    print()
    g.print_euler_tour()
```

input

```
Enter number of vertices: 7
Enter number of edges: 9
Enter edges (format: u v):
0 1
0 2
1 2
1 3
3 4
1 4
2 5
5 6
2 6
```

output

```
Graph has an Euler Circuit.

Running Hierholzer's Algorithm...
Splicing Cycle: [0, 2, 1, 0]
Splicing Cycle: [1, 3, 4, 1]
Splicing Cycle: [2, 5, 6, 2]

Final Euler Tour: [0, 1, 3, 4, 1, 2, 5, 6, 2, 0]
```

input

```
Enter number of vertices: 17
Enter number of edges: 22
Enter edges (format: u v):
0 1
1 2
2 3
3 4
4 5
5 6
6 7
0 7
1 6
2 5
6 8
8 9
9 10
10 11
11 5
1 12
2 12
12 13
13 14
14 15
15 16
12 16
```

output

```
Graph has an Euler Circuit.

Running Hierholzer's Algorithm...
Splicing Cycle: [0, 7, 6, 5, 4, 3, 2, 1, 0]
Splicing Cycle: [1, 6, 8, 9, 10, 11, 5, 2, 12, 1]
Splicing Cycle: [12, 13, 14, 15, 16, 12]

Final Euler Tour: [0, 1, 6, 8, 9, 10, 11, 5, 2, 12, 13, 14, 15, 16, 12, 1, 2,
3, 4, 5, 6, 7, 0]
```