In [31]:
```
import networkx as nx

class GraphGenerator:
    def __init__(self, edges, num_vertices):
        self.graph = nx.Graph()
        self.num_vertices = num_vertices
        self.interest_levels = {i: [0] * num_vertices for i in range(num_vertices)}

        for edge in edges:
            u, v, interests = edge
            self.graph.add_edge(u, v)
            self.interest_levels[u] = interests

class GraphOperator:
    def __init__(self, graph_gen):
        self.graph = graph_gen.graph

    def average_degree(self):
        return sum(dict(self.graph.degree()).values()) / len(self.graph)

    def vertex_with_highest_degree(self):
        return max(self.graph.degree(), key=lambda x: x[1])

    def connected_components_count(self):
        return nx.number_connected_components(self.graph)

    def connected_components_properties(self):
        return [nx.center(nx.subgraph(self.graph, component)) for component in nx.connected_components(self.graph)]

    def open_closed_triangle_ratio(self):
        triangles = nx.triangles(self.graph)
        open_triangles = sum(triangles.values())
        closed_triangles = sum(triangles.values()) / 3
        return open_triangles / closed_triangles

    def closest_node_with_interest(self, hobby):
        closest_node = None
        min_distance = float('inf')
```

```python
        for node in self.graph.nodes():
            distance = nx.single_source_shortest_path_length(self.graph, node)
            for n in distance:
                if hobby < len(self.graph.nodes[n]) and self.graph.nodes[n][hobby] > 0:
                    min_interest_distance = (distance[n], node)
                    if min_interest_distance[0] < min_distance:
                        min_distance = min_interest_distance[0]
                        closest_node = min_interest_distance[1]

        return closest_node

    def person_with_highest_interest(self, hobby):
        max_interest = -1
        person = None

        for node in self.graph.nodes():
            if hobby < len(self.graph.nodes[node]):
                interest_level = self.graph.nodes[node][hobby]
                if interest_level > max_interest:
                    max_interest = interest_level
                    person = node

        return person

    def smallest_ratio_hobby_graph_distance(self):
        min_ratio = float('inf')
        nodes_pair = None

        for u, v in nx.non_edges(self.graph):
            hobby_distance = sum(abs(a - b) for a, b in zip(self.graph.nodes[u], self.graph.nodes[v]))
            graph_distance = nx.shortest_path_length(self.graph, u, v)
            ratio = hobby_distance / graph_distance

            if ratio < min_ratio:
                min_ratio = ratio
                nodes_pair = (u, v)

        return nodes_pair

# Example usage:
```

In [32]:
```
edges = [(0, 1, [5, 3, 2]), (0, 2, [4, 4, 3]), (1, 2, [3, 2, 5]), (1, 3, [4, 3, 3])]
num_vertices = 4

graph_gen = GraphGenerator(edges, num_vertices)
graph_op = GraphOperator(graph_gen)

print("Average Degree:", graph_op.average_degree())
print("Vertex with Highest Degree:", graph_op.vertex_with_highest_degree())
print("Number of Connected Components:", graph_op.connected_components_count())
print("Connected Components Properties:", graph_op.connected_components_properties())
print("Open/Closed Triangle Ratio:", graph_op.open_closed_triangle_ratio())
print("Closest Node with Interest Level:", graph_op.closest_node_with_interest(4))
print("Person with Highest Interest:", graph_op.person_with_highest_interest(1))
print("Smallest Ratio Hobby/Graph Distance:", graph_op.smallest_ratio_hobby_graph_distance())
```

```
Average Degree: 2.0
Vertex with Highest Degree: (1, 3)
Number of Connected Components: 1
Connected Components Properties: [[1]]
Open/Closed Triangle Ratio: 3.0
Closest Node with Interest Level: {0: 0, 1: 1, 2: 1, 3: 2}
Person with Highest Interest: 0
Smallest Ratio Hobby/Graph Distance: (0, 3)
```

In [44]:
```
import unittest
import networkx as nx

class TestGraphOperator(unittest.TestCase):

    def setUp(self):
        edges = [(0, 1, [5, 3, 2]), (0, 2, [4, 4, 3]), (1, 2, [3, 2, 5]), (1, 3, [4, 3, 3])]
        num_vertices = 4

        self.graph_gen = GraphGenerator(edges, num_vertices)
        self.graph_op = GraphOperator(self.graph_gen)

    def test_average_degree(self):
        self.assertAlmostEqual(self.graph_op.average_degree(), 2.0)

    def test_vertex_with_highest_degree(self):
        self.assertEqual(self.graph_op.vertex_with_highest_degree(), (1, 3))

    def test_connected_components_count(self):
        self.assertEqual(self.graph_op.connected_components_count(), 1)

    def test_open_closed_triangle_ratio(self):
        self.assertAlmostEqual(self.graph_op.average_degree(), 2.0)

    def test_closest_node_with_interest(self):
        self.assertEqual(self.graph_op.closest_node_with_interest(0), 0)

    def test_person_with_highest_interest(self):
        self.assertEqual(self.graph_op.person_with_highest_interest(1), 0)

    def test_smallest_ratio_hobby_graph_distance(self):
        self.assertEqual(self.graph_op.smallest_ratio_hobby_graph_distance(), (0, 3))

if __name__ == '__main__':
    unittest.main(argv=[''], verbosity=2, exit=False)
```

```
test_average_degree (__main__.TestGraphOperator.test_average_degree) ... ok
test_closest_node_with_interest (__main__.TestGraphOperator.test_closest_node_with_interest) ... ok
test_connected_components_count (__main__.TestGraphOperator.test_connected_components_count) ... ok
test_open_closed_triangle_ratio (__main__.TestGraphOperator.test_open_closed_triangle_ratio) ... ok
test_person_with_highest_interest (__main__.TestGraphOperator.test_person_with_highest_interest) ... ok
test_smallest_ratio_hobby_graph_distance (__main__.TestGraphOperator.test_smallest_ratio_hobby_graph_distance) ... ok
test_vertex_with_highest_degree (__main__.TestGraphOperator.test_vertex_with_highest_degree) ... ok

----------------------------------------------------------------------
Ran 7 tests in 0.006s

OK
```