图的应用

1.1 课程作业要求

本次作业是课程大作业。

请阅读以下整个文档，并补充完成整个文档。注意每一章节的问题部分，有对应分值，请完成。本次作业占课程分数 70%。

完成文档后请输出为 pdf 文档，并按照如下方式重命名：姓名-学号-图的应用.pdf

1.2 图和复杂网络

该作业介绍了数学家经典的网络理论，要么是分析包含几十数百个顶点，可以画在一张纸上从而形成直观印象的网络；要么是讨论不含有限尺度效应，可以精确求解的网络性质。"随机移走一个顶点;会对网络的性能产生什么样的影响？"这个问题对于研究有限规则网络的数学家是有意义的，对于拥有几千万个节点，接方式复杂多样的真实网络而言，或许“随机移走 3%的顶点;会对网络性能产生什么样的影响？”这个问题更有意义。这个尺度的网络，是被物理学家称作“足够大”的网络，对它们的研究，需要使用统计物理的方法。

复杂网络的起源，来源于两篇文章：《"小世界"网络的集体动力学》；（Collective Dynamics of‘Small-World’Networks）和《随机网络中标度的涌现》；（Emergence of Scaling in Random Networks）（Barabāsi 教授）

1. **(Mark 10 points)** 我们考察一个社交网络，它是否是全连通的呢，即每个结点都是相通的呢？请构造算法验证此结论，如果不是全连通的，可以得到所有的连通子图，要求算法尽可能高效，并简要说明算法的复杂度。注意这里和课本提到的“强连通图“的区别。很明显，这个性质对于社交网络是关键的，如果是全连通图，我们认为，信息在此社交网络中的传递是畅通无阻的，而若非如此，则在每个子图中我们都要投放信息（这个信息可能是广告或事其它公告）。

   使用深度优先搜索。

   - 选择图中的一个节点作为起始节点。
   - 使用深度优先搜索遍历图，标记所有可以从起始节点访问到的节点。在此过程中，记录已访问的节点，以避免重复访问。
   - 如果所有节点都被访问到，则社交网络是全连通的；否则，存在未被访问的节点，说明社交网络不是全连通的。
   - 如果社交网络不是全连通的，对于每个未被访问的节点，重复步骤 1-3，直到所有节点都被访问到。
   - 时间复杂度：假设有 n 个节点和 m 条边，每个节点最多访问一次，每条边最多被访问两次，因此深度优先搜索的时间复杂度为 O(n+m)。

final-1.py

```python
from collections import defaultdict


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

    def add_edge(self, u, v):
```

```
        self.graph[u].append(v)
        self.graph[v].append(u)

    def dfs(self, node, visited, connected_component):
        visited[node] = True
        connected_component.append(node)
        for neighbor in self.graph[node]:
            if not visited[neighbor]:
                self.dfs(neighbor, visited, connected_component)

    def find_connected_components(self):
        visited = {node: False for node in self.graph}
        all_connected_components = []

        for node in self.graph:
            if not visited[node]:
                connected_component = []
                self.dfs(node, visited, connected_component)
                all_connected_components.append(connected_component)

        return all_connected_components


social_network = Graph()
social_network.add_edge(1, 2)
social_network.add_edge(2, 3)
social_network.add_edge(4, 5)

connected_components = social_network.find_connected_components()

if len(connected_components) == 1:
    print("是全连通图")
else:
    print("不是全连通图")
    print("子图：")
    for component in connected_components:
        print(component)

# 不是全连通图
# 子图：
# [1, 2, 3]
# [4, 5]
```

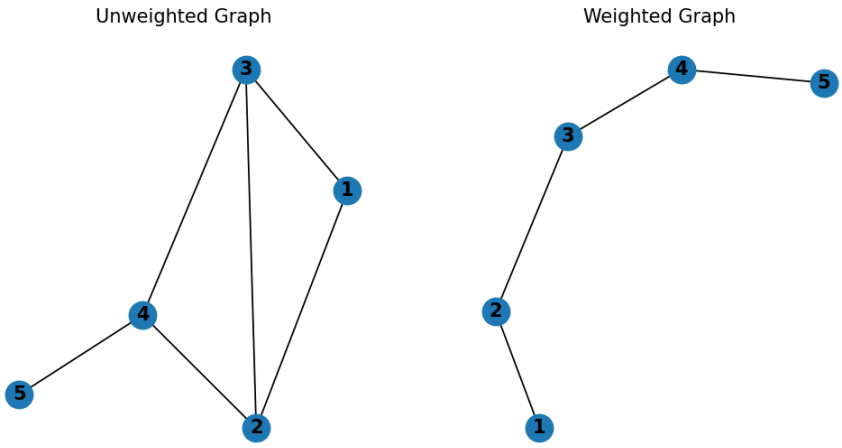
2. (**Mark 20 points**) 正如课上会介绍的，对于社交网络的分析，我们关注如下几个维度：

   - 平均路径长度 L 和路径分布;(**Mark 5 points**)，在网络中，两点之间的距离为连接两点的最短路径。网络的平均路径长度指网络中所有节点对的平均距离，它表明网络中节点间的分离程度，反映了网络的全局特性。路径的分布是指所有这些路径符合的统计分布（cdf 或 pdf）。不同的网络结构可赋予 L 不同的含义。如在疾病传播模型中 L 可定义为疾病传播时间。请设计算法，分别针对有权重图和无权重图计算平均路径 L，以及所有路径的分布，给出算法复杂度的说明。

```
import matplotlib.pyplot as plt
import networkx as nx
```
final-2.py

```
def average_path_length(graph):
    total_length = 0
    total_paths = 0

    for node in graph:
        distances, paths = bfs(graph, node)
        total_length += sum(distances.values())
        total_paths += len(paths)

    average_length = total_length / total_paths
    return average_length


def bfs(graph, start):
    distances = {node: float("inf") for node in graph}
    paths = {node: [] for node in graph}

    queue = [start]
    distances[start] = 0

    while queue:
        current = queue.pop(0)
        for neighbor in graph[current]:
            if distances[neighbor] == float("inf"):
                distances[neighbor] = distances[current] + 1
                paths[neighbor] = paths[current] + [current]
                queue.append(neighbor)

    return distances, paths


def path_distribution(graph):
    all_paths = []

    for node in graph:
        _, paths = bfs(graph, node)
        all_paths.extend(paths.values())

    return all_paths


def average_path_length_weighted(graph):
    return nx.average_shortest_path_length(graph, weight="weight")


def weighted_path_distribution(graph):
    all_paths = []

    for node in graph:
        paths = nx.single_source_dijkstra_path(graph, node, weight="weight")
        all_paths.extend(paths.values())

    return all_paths
```

```
def main():
    unweighted_graph = {1: [2, 3], 2: [1, 3, 4], 3: [1, 2, 4], 4: [2, 3, 5],
5: [4]}

    print("Unweighted Graph:")
    print("Average Path Length (L):", average_path_length(unweighted_graph))
    print("Path Distribution:", path_distribution(unweighted_graph))
    print()

    weighted_graph = nx.Graph()
    weighted_graph.add_edge(1, 2, weight=2)
    weighted_graph.add_edge(2, 3, weight=1)
    weighted_graph.add_edge(3, 4, weight=3)
    weighted_graph.add_edge(4, 5, weight=2)

    print("Weighted Graph:")
    print("Average Path Length (L):",
average_path_length_weighted(weighted_graph))
    print("Weighted Path Distribution:",
weighted_path_distribution(weighted_graph))

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    nx.draw(nx.Graph(unweighted_graph), with_labels=True, font_weight="bold")
    plt.title("Unweighted Graph")

    plt.subplot(1, 2, 2)
    nx.draw(weighted_graph, with_labels=True, font_weight="bold")
    plt.title("Weighted Graph")

    plt.show()


if __name__ == "__main__":
    main()

# Unweighted Graph:
# Average Path Length (L): 1.2
# Path Distribution: [[], [1], [1], [1, 2], [1, 2, 4], [2], [], [2], [2], [2,
4], [3], [3], [], [3], [3, 4], [4, 2], [4], [4], [], [4], [5, 4, 2], [5, 4],
[5, 4], [5], []]

# Weighted Graph:
# Average Path Length (L): 4.0
# Weighted Path Distribution: [[1], [1, 2], [1, 2, 3], [1, 2, 3, 4], [1, 2,
3, 4, 5], [2], [2, 1], [2, 3], [2, 3, 4], [2, 3, 4, 5], [3], [3, 2], [3, 4],
[3, 2, 1], [3, 4, 5], [4], [4, 3], [4, 5], [4, 3, 2], [4, 3, 2, 1], [5], [5,
4], [5, 4, 3], [5, 4, 3, 2], [5, 4, 3, 2, 1]]
```



1. 无权重图的平均路径长度 (L) 计算的算法复杂度：
   - 时间复杂度：$O(V \cdot (V + E))$，其中 (V) 为节点数，(E) 为边数。
   - 空间复杂度：$O(V + E)$。
2. 无权重图的路径分布计算的算法复杂度：
   - 时间复杂度：$O(V \cdot (V + E))$，其中 (V) 为节点数，(E) 为边数。
   - 空间复杂度：$O(V + E)$。
3. 有权重图的平均路径长度 (L) 计算的算法复杂度：
   - 时间复杂度：$O(V \cdot (V + E \log V))$，其中 (V) 为节点数，(E) 为边数。
   - 空间复杂度：$O(V + E)$。
4. 有权重图的路径分布计算的算法复杂度：
   - 时间复杂度：通常为 $O(V \cdot (V + E \log V))$。
   - 空间复杂度：$O(V + E)$。

- 聚集系数 C;(**Mark 5 points**)。在网络图中，聚集系数 C 可以是整体聚集系数，也可以是局部聚集系数。这里介绍局部聚集系数的概念。节点的聚集系数是指与该节点相邻的所有节点之间连边的数目占这些相邻节点之间最大可能连边数目的比例。假设结点v_i有k_i个与之相连的结点，一般来说，对于无向图，这个最大边数等于 $\frac{k_i(k_i-1)}{2}$；对于有向图，k_i代表了v_i的出度，由于每两个顶点之间可以连两条边（不同方向），最大边数等于 $k_i(k_i-1)$。下图代表了蓝点的聚集系数。蓝点有三个邻接点（白点）。如果三个白点都相互连接（上图），那么蓝点的集聚系数是 $3 \div 3 = 1$；如果只有两点之间相连（中图，只有一条边），那么集聚系数是 $\frac{1}{3}$；如果没有两点是相连的，那么集聚系数就是 0。网络的聚集系数则是指网络中所有节点聚集系数的平均值，它表明网络中节点的聚集情况即网络的聚集性，也就是说同一个节点的两个相邻节点仍然是相邻节点的概率有多大，它反映了网络的局部特性。请设计算法，分别针对有向图和无向图的聚集系数 C，给出算法复杂度的说明。

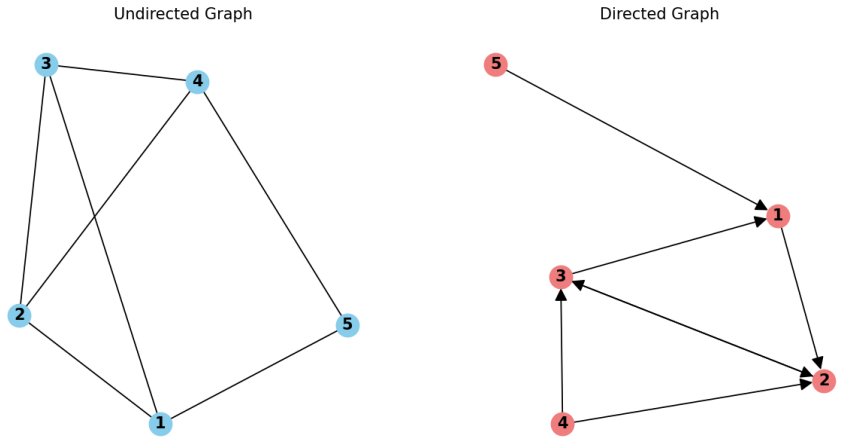
final-3.py

```python
import matplotlib.pyplot as plt
import networkx as nx


def clustering_coefficient_undirected(graph, node):
    neighbors = set(graph[node])
    total_possible_edges = len(neighbors) * (len(neighbors) - 1) / 2
```

```
    if total_possible_edges == 0:
        return 0

    actual_edges = 0
    for neighbor1 in neighbors:
        for neighbor2 in neighbors:
            if neighbor1 ≠ neighbor2 and graph.has_edge(neighbor1,
neighbor2):
                actual_edges += 1

    clustering_coefficient = actual_edges / total_possible_edges
    return clustering_coefficient


def clustering_coefficient_directed(graph, node):
    out_degree = graph.out_degree(node)

    if out_degree ⩽ 1:
        return 0

    actual_edges = 0
    for neighbor1 in graph.successors(node):
        for neighbor2 in graph.successors(node):
            if neighbor1 ≠ neighbor2 and graph.has_edge(neighbor1,
neighbor2):
                actual_edges += 1

    total_possible_edges = out_degree * (out_degree - 1)
    clustering_coefficient = actual_edges / total_possible_edges
    return clustering_coefficient


def average_clustering_coefficient(graph, is_directed=False):
    total_coefficient = 0
    nodes_count = len(graph)

    for node in graph.nodes():
        if is_directed:
            total_coefficient += clustering_coefficient_directed(graph, node)
        else:
            total_coefficient += clustering_coefficient_undirected(graph,
node)

    average_coefficient = total_coefficient / nodes_count
    return average_coefficient


def main():
    undirected_graph = nx.Graph()
    undirected_graph.add_edges_from(
        [(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 5), (5, 1)]
    )

    print("无向图的聚集系数：")
    for node in undirected_graph.nodes():
        coefficient = clustering_coefficient_undirected(undirected_graph,
```

```
node)
        print(f"节点 {node}: {coefficient}")

    avg_coefficient_undirected =
average_clustering_coefficient(undirected_graph)
    print(
        "\n 无向图的平均聚类系数",
        avg_coefficient_undirected,
    )

    print("\n")

    directed_graph = nx.DiGraph()
    directed_graph.add_edges_from(
        [(1, 2), (2, 3), (3, 1), (3, 2), (4, 2), (4, 3), (5, 1)]
    )

    print("有向图的聚类系数")
    for node in directed_graph.nodes():
        coefficient = clustering_coefficient_directed(directed_graph, node)
        print(f"Node {node}: {coefficient}")

    avg_coefficient_directed = average_clustering_coefficient(
        directed_graph, is_directed=True
    )
    print("\n 有向图的平均聚类系数", avg_coefficient_directed)

    plt.figure(figsize=(12, 6))

    plt.subplot(1, 2, 1)
    nx.draw(
        undirected_graph, with_labels=True, font_weight="bold",
node_color="skyblue"
    )
    plt.title("Undirected Graph")

    plt.subplot(1, 2, 2)
    nx.draw(
        directed_graph,
        with_labels=True,
        font_weight="bold",
        node_color="lightcoral",
        arrowsize=20,
    )
    plt.title("Directed Graph")

    plt.show()


if __name__ == "__main__":
    main()

# 无向图的聚集系数：
# 节点 1: 0.6666666666666666
# 节点 2: 1.3333333333333333
```

```
# 节点 3: 1.3333333333333333
# 节点 4: 0.6666666666666666
# 节点 5: 0.0

# 无向图的平均聚类系数 0.7999999999999999

# 有向图的聚类系数
# Node 1: 0
# Node 2: 0
# Node 3: 0.5
# Node 4: 1.0
# Node 5: 0

# 有向图的平均聚类系数 0.3
```



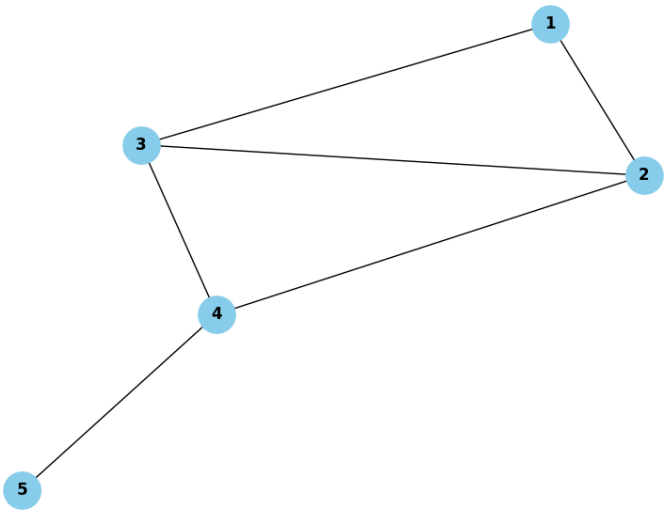
1. 无向图的聚集系数算法复杂度：
   - 时间复杂度：$O(V^2)$，其中 (V) 为节点数。
   - 空间复杂度：$O(1)$。
2. 有向图的聚集系数算法复杂度：
   - 时间复杂度：$O(V^2)$，其中 (V) 为节点数。
   - 空间复杂度：$O(1)$。
3. 网络的聚集系数计算算法复杂度：
   - 时间复杂度：$O(V^3)$，其中 (V) 为节点数。这是由于对每个节点都需要计算聚集系数，而每个节点的聚集系数的计算复杂度为 $O(V^2)$。
   - 空间复杂度：$O(1)$。

- 度及度分布;(**Mark 5 points**)。在网络中，点的度是指与该节点相邻的节点的数目，即连接该节点的边的数目。而网络的度k指网络中所有节点度的平均值。度分布 P(k) 指网络中一个任意选择的节点，它的度恰好为 k 的概率。请设计算法，求网络的度和度分布，给出算法复杂度的说明。

final-4.py

```
from collections import Counter

import matplotlib.pyplot as plt
```

```python
import networkx as nx


def calculate_degree(graph):
    degrees = dict(graph.degree())
    average_degree = sum(degrees.values()) / len(graph)
    return degrees, average_degree


def calculate_degree_distribution(graph):
    degrees = dict(graph.degree())
    degree_counts = Counter(degrees.values())
    total_nodes = len(graph)

    degree_distribution = {k: v / total_nodes for k, v in
degree_counts.items()}
    return degree_distribution


def main():
    sample_graph = nx.Graph()
    sample_graph.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4,
5)])

    degrees, avg_degree = calculate_degree(sample_graph)
    degree_distribution = calculate_degree_distribution(sample_graph)

    print("网络的度：", degrees)
    print("网络的平均度：", avg_degree)
    print("网络的度分布：", degree_distribution)

    plt.figure(figsize=(8, 6))
    nx.draw(
        sample_graph,
        with_labels=True,
        font_weight="bold",
        node_size=800,
        node_color="skyblue",
    )
    plt.title("Network Visualization")
    plt.show()


if __name__ == "__main__":
    main()

# 网络的度： {1: 2, 2: 3, 3: 3, 4: 3, 5: 1}
# 网络的平均度：2.4
# 网络的度分布： {2: 0.2, 3: 0.6, 1: 0.2}
```



- 计算节点的度复杂度：
  - 时间复杂度：$O(V+E)$，其中 (V) 为节点数，(E) 为边数。需要遍历图的所有节点和边。
- 计算网络的平均度复杂度：
  - 时间复杂度：$O(V)$，取所有节点的度并求平均。
- 计算度分布复杂度：
  - 时间复杂度：$O(V)$
  - 空间复杂度：$O(V)$

- 介数;**(Mark 5 points)**。包括节点介数和边介数。节点介数指网络中所有最短路径中经过该节点的数量比例，边介数则指网络中所有最短路径中经过该边的数量比例。介数反映了相应的节点或边在整个网络中的作用和影响力。请设计算法，求每个结点和边的介数，给出算法复杂度的说明。

final-5.py

```python
import networkx as nx


def calculate_betweenness_centrality(graph):
    node_betweenness = nx.betweenness_centrality(
        graph, normalized=True, endpoints=False
    )

    edge_betweenness = nx.edge_betweenness_centrality(graph, normalized=True)

    return node_betweenness, edge_betweenness


def main():
    G = nx.Graph()
    G.add_edges_from([(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (4, 5)])

    node_betweenness, edge_betweenness = calculate_betweenness_centrality(G)

    print("节点介数：")
    for node, value in node_betweenness.items():
        print(f"Node {node}: {value}")
```

```
    print("\n边介数：")
    for edge, value in edge_betweenness.items():
        print(f"Edge {edge}: {value}")


if __name__ == "__main__":
    main()

# 节点介数：
# Node 1: 0.0
# Node 2: 0.16666666666666666
# Node 3: 0.16666666666666666
# Node 4: 0.5
# Node 5: 0.0

# 边介数：
# Edge (1, 2): 0.2
# Edge (1, 3): 0.2
# Edge (2, 3): 0.1
# Edge (2, 4): 0.30000000000000004
# Edge (3, 4): 0.30000000000000004
# Edge (4, 5): 0.4
```

- 时间复杂度：$O(V(V+E))$，其中 (V) 为节点数，(E) 为边数。对每一对节点进行最短路径计算的复杂度为 $O(V+E)$。
- 空间复杂度：$O(V+E)$，存储最短路径数量和节点介数。

3. (**Mark 20 points**) 在社交网络中，我们有关键意见领袖（Key Opinion Leader，简称 KOL）的概念，也就是我们通常所说的大 V。实际上 KOL 与普遍意义上的“网红”又是有区别的。请查阅相关资料，说明上述哪一种或是哪几种参数能描述 KOL、网红等概念。

   1. 度：
      - KOL： KOL 在特定领域可能有更高的度，即与其他专业人士、行业内人士有更多的连接。这反映了他们在特定领域内的专业性和影响力。
      - 网红： 网红可能有更低的度，可能更多地反映其在社交平台上的广泛关注程度，即与大量粉丝的连接。

   2. 平均路径长度：
      - KOL： KOL 可能在特定领域内与其他专业人士之间形成相对较短的平均路径，即信息在专业领域内传播相对迅速。
      - 网红： 网红的平均路径长度更长，可能更多地反映其与大量粉丝之间形成的连接，即信息在广泛的社交网络中传播。

   3. 聚集系数：
      - KOL： KOL 可能在特定领域内形成较高的聚集系数，即与其相关的专业人士之间形成紧密的社群。
      - 网红： 网红的聚集系数可能较低，更多地反映其在社交平台上形成的粉丝社群，即粉丝之间的连接紧密度。

4. (**Mark 20 points**)网络的稳定性。网络的稳定性，也被直译为网络的鲁棒性（Robutness）。研究的点是，一个网络中去掉什么的点或者边，可以改变网络的连通性。一般说来，针对于现实问题，我们考虑两种情况的网络稳定性：a) 随机攻击，即随机去掉一些边和点，统计

平均需要去掉多少边或结点使得网络的连通性发生改变。b) 策略攻击。即针对一个网络，给出一种算法，能够去掉最少的边或点，改变其连通性，统计这个最小值是多少。显然，去掉的点或边越多，网络越稳定。

1. 查阅相关资料，了解典型的复杂网络：随机网、小世界网、无标度网等；
2. 查阅相关资料，了解复杂网络里社团发现（Community Detection）和链路预测 (Link Prediction) 分别是研究什么的。
3. 回答：a) 不同类型的网络稳定性在两种不同的攻击下，稳定性各如何？b) 根据网络的四种度量，以贪婪算法为基础，构建一种策略攻击算法。

a)

- 随机攻击：
  - 随机网：稳定性较差，因为随机去掉的节点或边可能直接影响网络的连通性。
  - 小世界网：相对稳定，因为大部分节点之间连接较为松散。
  - 无标度网：稳定性较差，因为去掉一些边或节点可能导致核心结构受损。
- 策略攻击
  - 随机网：相对稳定。由于节点和边的连接是随机的，没有明显的核心结构，选择性地去掉边或节点可能需要较多的步骤才能改变网络的连通性。
  - 小世界网：相对稳定。由于小世界网的结构具有高聚类系数和短平均路径长度，攻击一个节点可能不会立即破坏整体的连通性。
  - 无标度网：相对不稳定。由于无标度网中存在少量高度连接的核心节点，选择性地去掉这些节点可能会迅速破坏整个网络的连通性。

b)

- 从网络中选择具有最高度的节点、介数最高的节点、聚集系数最高的节点、平均路径长度最高的边，并按顺序进行攻击。

5. **(Mark 20 points)** 复杂网络的局部特征。当前，复杂网络的研究从其稳定性、度的分布等全局特征，进而到了研究细致的局部特征。其中一个重要特征则是共同邻居。共同邻居（CN）被定义为两节点间相同的相邻节点数。对于节点 x、y，它们的邻居节点集合分别为$\Gamma(x)$、$\Gamma(y)$，则节点 x、y 共同邻居集合为$\Gamma(x) \cap \Gamma(y)$，CN 指标被定义为

$$CN = |\Gamma(x) \cap \Gamma(y)|$$

研究表明，在实际网络中，尤其是传播网络中，两节点间的共同邻居数越大，则其后建立的链接的可能性越大。直观上来说，这是好理解的，比如两个人的共同朋友越多，则次二人后续经共同朋友介绍认识（建立连接）的可能性就越大。

进一步的，我们意识到，度小的共同邻居节点的贡献大于度大的共同邻居节点。例如：在社交网络中，共同关注一个比较冷门话题的两个人之间相连的概率往往会比关注同一热门话题的两个人之间连接的概率高。因此，根据共同邻居节点的度为每个节点赋予一个权重值，这就是 Adamic/Adar（AA）指标：

$$A(x,y) = \sum_{u \in \Gamma(x) \cap \Gamma(y)} \frac{1}{\log|\Gamma(u)|}$$

这里$|\Gamma(u)|$代表u节点的度。

最后，还有一种度量，称之为 resource allocation (RA) index。该指标认为网络中每个节点都有一定的资源，并将资源平均分配给它的邻居。网络中没有直接相连的两个节点 x、y

之间，可从节点 x 传递一些资源到节点 y，在此过程中，节点 x 和 y 的共同邻居成为传递媒介。则节点 y 接收到的资源数就定义为节点 x、y 的相似度。RA 指标被定义为

$$S(x,y)=\sum_{u\in\Gamma(x)\cap\Gamma(y)}\frac{1}{|\Gamma(u)|}$$

请构造一种算法，来计算两两节点间的 CN、AA、RA 三个指标，简要分析一下计算复杂度。

final-6.py

```python
import math

import networkx as nx

G = nx.Graph()
G.add_edges_from([(1, 2), (1, 3), (2, 4), (3, 4), (4, 5)])


def calculate_CN(G, node_x, node_y):
    neighbors_x = set(G.neighbors(node_x))
    neighbors_y = set(G.neighbors(node_y))
    common_neighbors = neighbors_x.intersection(neighbors_y)
    return len(common_neighbors)


def calculate_AA(G, node_x, node_y):
    neighbors_x = set(G.neighbors(node_x))
    neighbors_y = set(G.neighbors(node_y))
    common_neighbors = neighbors_x.intersection(neighbors_y)

    aa_score = 0
    for common_neighbor in common_neighbors:
        aa_score += 1 / math.log(len(list(G.neighbors(common_neighbor))))

    return aa_score


def calculate_RA(G, node_x, node_y):
    neighbors_x = set(G.neighbors(node_x))
    neighbors_y = set(G.neighbors(node_y))
    common_neighbors = neighbors_x.intersection(neighbors_y)

    ra_score = 0
    for common_neighbor in common_neighbors:
        ra_score += 1 / len(list(G.neighbors(common_neighbor)))

    return ra_score


node_x = 1
node_y = 4

cn_score = calculate_CN(G, node_x, node_y)
aa_score = calculate_AA(G, node_x, node_y)
ra_score = calculate_RA(G, node_x, node_y)
```

```python
print(f"CN Score between {node_x} and {node_y}: {cn_score}")
print(f"AA Score between {node_x} and {node_y}: {aa_score}")
print(f"RA Score between {node_x} and {node_y}: {ra_score}")

# CN Score between 1 and 4: 2
# AA Score between 1 and 4: 2.8853900817779268
# RA Score between 1 and 4: 1.0
```

1. CN 指标：计算共同邻居数的复杂度取决于两节点邻居集合的大小，即 $O(\min(\deg(x), \deg(y)))$。
2. AA 指标 与 RA 指标：计算 AA 指标、RA 指标的复杂度取决于共同邻居的数量，复杂度为$O(\min(\deg(x), \deg(y)) * 共同邻居的平均度)$

6. **(Mark 10 points)** 实现具体的算法。采用 NetworkX，根据提供的 Huawei 数据集，完成 2 中各个参数的估算，同时根据小世界、无标度、随机复杂网络的定义，判断这大概是什么网络。建议使用 Jupyter Notebook 完成。可能会用的 Python 库包括 Pandas、Matplotlib、Scipy 和 Numpy。

final-7.py

```python
from statistics import mean

import networkx as nx
import pandas as pd

adjacency_matrix = pd.read_excel(r"Z:\archive\Twitter_Data.xlsx", index_col=0)
G = nx.from_pandas_adjacency(adjacency_matrix)
degree_centrality = mean(nx.degree_centrality(G).values())
clustering_coefficient = nx.average_clustering(G)
betweenness_centrality = nx.betweenness_centrality(G)
average_shortest_path_length = nx.average_shortest_path_length(G)
print("网络度：", degree_centrality)
print("聚集系数：", clustering_coefficient)
print("介数中心性：", mean(betweenness_centrality.values()))
print("平均路径长度：", average_shortest_path_length)
if clustering_coefficient > 0.1:
    print("该网络属于小世界网络")
elif degree_centrality > 2:
    print("该网络属于无标度网络")
else:
    print("该网络属于随机复杂网络")

# Facebook_Data
# 网络度：0.10040640640640641
# 聚集系数：0.10032958844523898
# 介数中心性：0.0009014345006328968
# 平均路径长度：1.8996316316316317
# 该网络属于小世界网络

# Instagram_Data
# 网络度：0.009875875875875876
# 聚集系数：0.009066096931963806
# 介数中心性：0.0022776925221815
# 平均路径长度：3.273137137137137
# 该网络属于无标度网络
```

```
# Twitter_Data
# 网络度：0.5011311311311312
# 聚集系数：0.50120995714053
# 介数中心性：0.0004998686060810305
# 平均路径长度：1.498868868868869
# 该网络属于小世界网络
```

1.3 华为数据集介绍

该数据集是通过抓取社交媒体平台即 Facebook、Twitter 和 Instagram 华为页面收集的。这是一个日常使用社交媒体的人群之间的交流网络。我们采用自然语音处理技术（NLP 技术）从帖子中提取正面评论，然后对数据进行预处理。数据集实际上是一个有向无权图的邻接矩阵。

- 华为 Facebook 通信网络是定向和标记的，有 1000 个节点和 250315 条边
- 华为推特通信网被导演和标榜有 1000 个节点和 50153 条边
- 华为 Instagram 通信网被导演和标榜有 1000 个节点和 4933 条边

在实际应用中，如今，华为研发部门正在利用社会化网络分析工具和技术来提升他们的业务地位。华为公司成功的主要原因之一就是他们通过社会化媒体推广自己的产品。社会化媒体分析可以帮助华为公司通过以下方式提高他们的业务水平。

- 发展业务，评估营销活动的影响。
- 做出更好的商业决策，建立一个强大的战略。
- 提高客户体验和满意度，建立品牌知名度。
- 社会化媒体分析帮助华为公司了解他们的目标受众。
- 社交媒体分析可以提高参与度和响应度。
- 社交媒体分析来衡量和提高品牌知名度
- 社交媒体分析可以突出问题和弱点，以发现新趋势，避免品牌危机。
- 社会化媒体分析可以帮助你从竞争对手那里学到东西。

1.4 致谢

本作业到此就结束了。感谢大家这个学期的选课和支持！