THE UNIVERSITY OF DODOMA

COLLEGE OF INFORMATICS AND VIRTUAL EDUCATION

ACADEMIC YEAR**: 2024/2025**

COURSE CODE:**CS 427**

TYPE OF WORK**: LAB 3**

GROUP No: **12**

PARTICIPANTS:

STUDENT'S NAME	**REGISTRATION NUMBER**	**PROGAMME**
1. ISSA SEIF PAMUI	T21-03-10935	BSc. CSDFE4
2. ABDALLAH SAIDI MWIRU	T21-03-04495	BSc. CSDFE4
3. IDDY ADILI MANUMBU	T21-03-14964	BSc. CSDFE4

INTRODUCTION

Social network analysis (SNA) explores the structure of relationships between entities, typically represented as nodes (individuals) and edges (relationships or interactions). One of the most studied datasets in network science is Zachary's Karate Club, which models friendships between 34 members of a karate club. This report outlines the process of analyzing this dataset using centrality measures, community detection algorithms, and network visualizations.

DATASET DESCRIPTION

Zachary's Karate Club dataset is a well-known undirected graph where:

- ❖ Nodes (34) represent individual members.
- ❖ Edges (78) represent friendships.
- ❖ The dataset reflects a real-life conflict in the club that led to its division into two factions.

The dataset is included with the NetworkX library as `nx.karate_club_graph()`.

LIBRARY & TOOLS

- ❖ NetworkX: Graph data structures and analysis tools
- ❖ Matplotlib: Network visualization
- ❖ Pandas and NumPy: Data handling and numerical computation
- ❖ Community (python-louvain): Community detection with the Louvain algorithm

Installation:

pip install networkx matplotlib pandas numpy python-louvain

THE CODE:

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import community as community_louvain
from urllib.request import urlopen
import csv
from io import StringIO


def analyze_karate_club():
    """
    Load and return Zachary's Karate Club graph with string
labels on nodes.
    """
    print("[INFO] Loading Zachary's Karate Club dataset...")
    G = nx.karate_club_graph()
    # Assign string labels
    nx.set_node_attributes(G, {n: str(n) for n in G.nodes()},
'label')
    return G
```

```
def load_from_csv(file_path, source_col='source',
target_col='target'):
    """
    Load an undirected graph from a CSV with edge list columns.
    If source/target columns not found, use first two columns.
    """
    print(f"[INFO] Loading graph edges from CSV: {file_path}")
    df = pd.read_csv(file_path)
    if source_col in df.columns and target_col in df.columns:
        edges = list(zip(df[source_col], df[target_col]))
    else:
        edges = list(zip(df.iloc[:, 0], df.iloc[:, 1]))
    G = nx.Graph()
    G.add_edges_from(edges)
    return G
```

```
def load_hyperlink_network(url):
    """
    Load a directed network of hyperlinks from a CSV at a URL.
    Assumes header (source,target).
    """
    print(f"[INFO] Fetching hyperlink network from URL: {url}")
    data = urlopen(url).read().decode('utf-8')
    reader = csv.reader(StringIO(data))
    headers = next(reader, None)
```

```
    G = nx.DiGraph()
    for row in reader:
        if len(row) < 2:
            continue
        G.add_edge(row[0], row[1])
    return G
```

```
def calculate_metrics(G):
    """
    Compute centrality metrics: degree, betweenness, closeness,
    eigenvector, and PageRank.
    Returns a dict of metrics mapping metric name to
dict(node->value).
    """
    print("[INFO] Calculating centrality metrics...")
    metrics = {
        'degree': nx.degree_centrality(G),
```

```python
        'betweenness': nx.betweenness_centrality(G),
        'closeness': nx.closeness_centrality(G),
        'eigenvector': nx.eigenvector_centrality(G,
max_iter=1000),
        'pagerank': nx.pagerank(G)
    }
    return metrics
```

```python
def identify_communities(G):
    """
    Detect communities using Louvain algorithm on graph G.
    Annotates G nodes with 'community' attribute and
    returns the partition mapping.
    """
    print("[INFO] Running Louvain community detection...")
    partition = community_louvain.best_partition(G)
    nx.set_node_attributes(G, partition, 'community')
    ncom = len(set(partition.values()))
    print(f"[INFO] Detected {ncom} communities.")
    return partition
```

```python
def visualize_network(G, partition=None, figsize=(10, 7),
savepath=None):
    """
    Draw network G with optional partition coloring.
    If savepath provided, saves image to file.
    """
    print("[INFO] Visualizing network...")
    plt.figure(figsize=figsize)
    pos = nx.spring_layout(G, seed=42)
```

```python
    if partition:
        communities = [partition[n] for n in G.nodes()]
        cmap = plt.cm.tab20
        nx.draw_networkx_nodes(G, pos, node_size=100,
node_color=communities, cmap=cmap, alpha=0.8)
    else:
        nx.draw_networkx_nodes(G, pos, node_size=100,
alpha=0.8)
```

```
    nx.draw_networkx_edges(G, pos, alpha=0.5)
    labels = nx.get_node_attributes(G, 'label')
    nx.draw_networkx_labels(G, pos, labels, font_size=8)
```

```
    plt.title('Network Visualization')
    plt.axis('off')
    if savepath:
        plt.savefig(savepath, dpi=300)
        print(f"[INFO] Visualization saved to {savepath}")
    plt.show()
```

```
def display_top_nodes(metrics, top_n=5):
    """
    Print the top_n nodes for each metric in metrics dict.
    """
    print(f"[INFO] Displaying top {top_n} nodes by metric...")
    for name, vals in metrics.items():
        sorted_nodes = sorted(vals.items(), key=lambda x: x[1],
reverse=True)[:top_n]
        print(f"\nTop {top_n} by {name.capitalize()}: ")
        for rank, (node, score) in enumerate(sorted_nodes, 1):
            print(f"  {rank}. Node {node}: {score:.4f}")
```

```
def print_basic_stats(G):
    """
    Print basic network statistics like number of nodes, edges,
density,
    connectivity, and component sizes if disconnected.
    """
    print("[INFO] Network basic statistics:")
    print(f" Nodes: {G.number_of_nodes()}")
    print(f" Edges: {G.number_of_edges()}")
    print(f" Density: {nx.density(G):.4f}")
    if nx.is_connected(G.to_undirected()):
        print(" Graph is connected.")
    else:
        comps =
list(nx.connected_components(G.to_undirected()))
        print(f" Graph disconnected with {len(comps)}
components.")
        print(f" Largest component size: {len(max(comps,
key=len))}")
```

```
def main(source_csv=None, hyperlink_url=None,
output_image="network.png"):
    # Load network
    if source_csv:
        G = load_from_csv(source_csv)
    elif hyperlink_url:
        G = load_hyperlink_network(hyperlink_url)
    else:
        G = analyze_karate_club()
```

```
    # Compute metrics and stats
    metrics = calculate_metrics(G)
    display_top_nodes(metrics)
    print_basic_stats(G)
```

```
    # Community detection and visualization
    partition = identify_communities(G)
    visualize_network(G, partition, savepath=output_image)
```

```
if __name__ == '__main__':
    main()
```

<u>OUTPUT</u>

- Top 5 Nodes by Centrality Metrics:

  Degree Centrality: Nodes 33, 0, 32, 2, 1

  Betweenness Centrality: Nodes 0, 33, 32, 2, 31

  Closeness Centrality: Nodes 0, 2, 33, 31, 8

  Eigenvector Centrality: Nodes 33, 0, 2, 32, 1

  PageRank: Nodes 33, 0, 32, 2, 1
- *Basic Graph Statistics:*

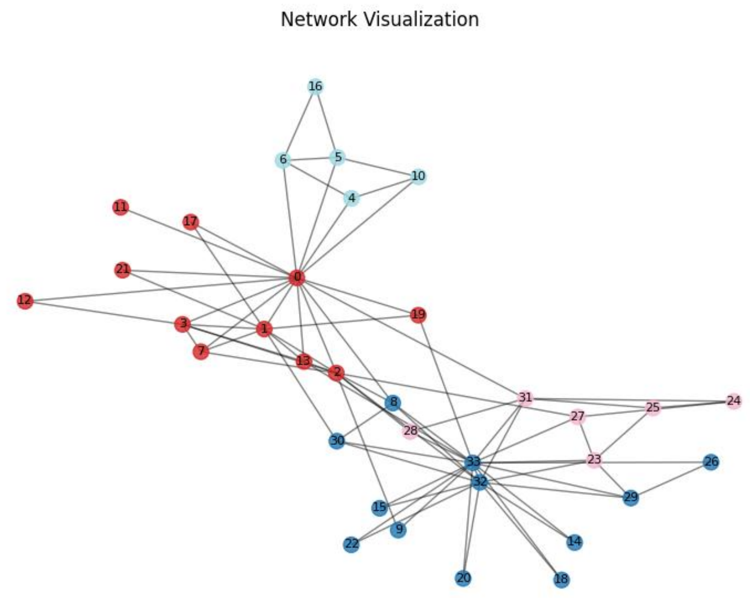
  Nodes: 34

  Edges: 78

  Density: 0.1390

  Connectivity: Connected
- *Communities Detected:*

  4 distinct communities detected using the Louvain algorithm.



VISUALIZATION

The graph visualization illustrates the structure of friendships and detected communities. Nodes are colored by community, and the layout is generated using the spring layout algorithm.

CONCLUSSION

This analysis of the Karate Club dataset shows how centrality measures and community detection provide insights into network structure. Node 33 and Node 0 are influential across multiple metrics, suggesting leadership roles. The visualization and community detection align with the real-world split that occurred in the club