МОДУЛЬНОЕ ЗАДАНИЕ №4
по дисциплине «Анализ и обработка веб данных»
Преподаватель: *Ассоциированный профессор* каф. «Программная инженерия» Серек Азамат Галымжанович

Report

1. Introduction

- Networks are the backbone of the web and social systems. They connect people, devices, and information, making global communication possible. In social media, networks shape how ideas spread, how communities form, and how trends grow. Simply put, without networks, the internet and social platforms wouldn't exist as we know them today.

- Applications: search engines (PageRank), community detection in social media, hyperlink analysis.

- Project goal: analyze a web graph and visualize its structure.

2. Theoretical Background

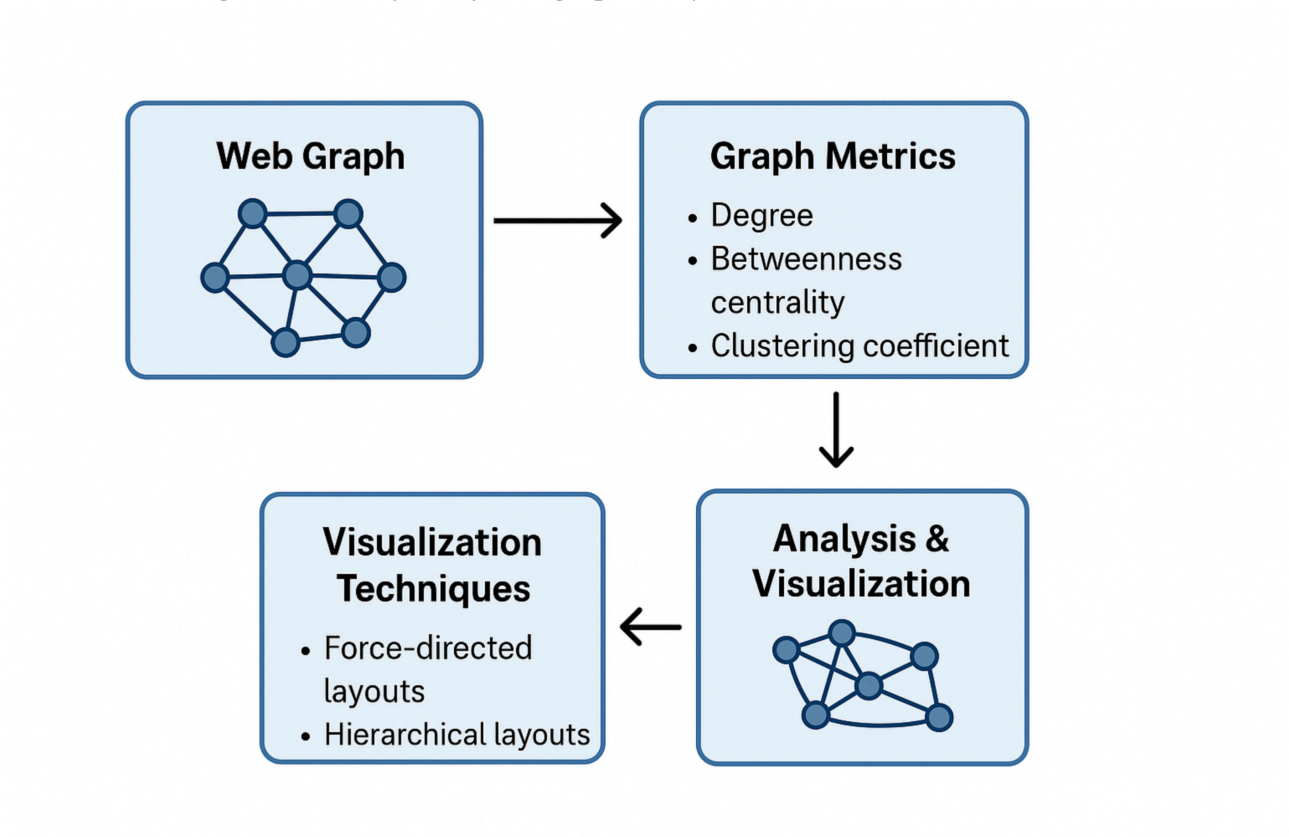
- Web Graphs represent the structure of the web. Each node is a webpage or user, and each edge is a link or connection between them. Graphs can be directed (one-way links) or undirected (mutual connections). Some are weighted, meaning edges have values showing the strength or importance of a connection.

- Graph Metrics help analyze these networks. Degree shows how many connections a node has. Betweenness centrality measures how often a node acts as a bridge between others. Clustering coefficient shows how tightly connected a node's neighbors are.

- Visualization Techniques make complex networks easier to understand. Force-directed layouts position nodes using simulated forces to highlight structure, while hierarchical layouts organize nodes in levels, showing clear parent–child or layered relationships.

3. System Design

- Pipeline: Data Collection → Graph Construction → Network Analysis → Visualization.
    - *Figure 1 – Workflow of web graph analysis and visualization.*

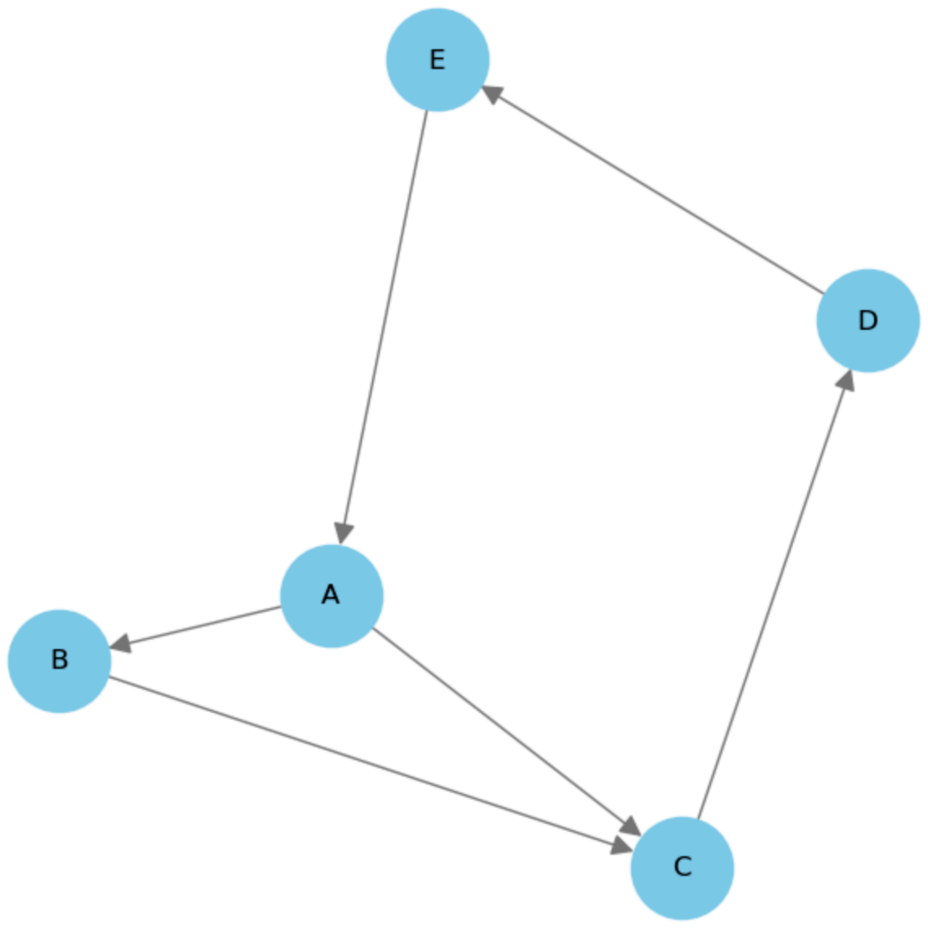


- Data Sources:
    - Crawled websites (links between pages).
    - Social network user connections.
    - Public graph datasets (e.g., SNAP, KONECT).

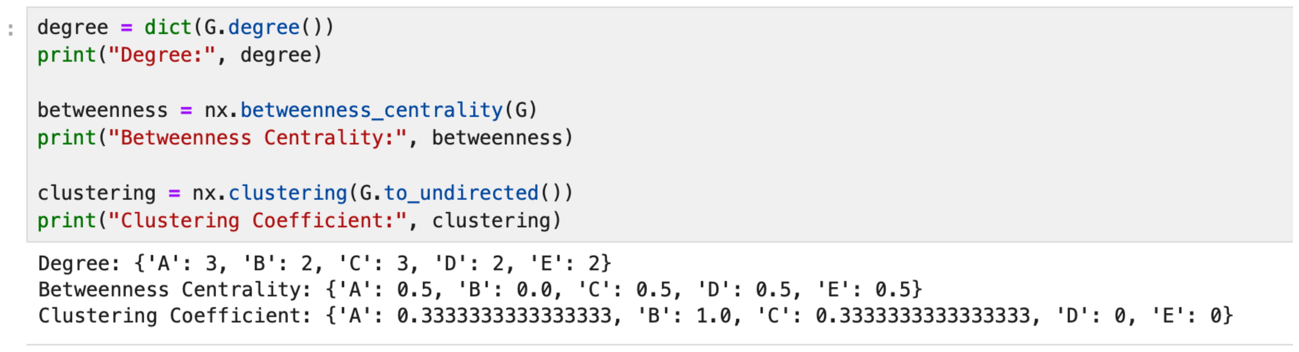
МОДУЛЬНОЕ ЗАДАНИЕ №4
по дисциплине «Анализ и обработка веб данных»
Преподаватель: *Ассоциированный профессор* каф. «Программная инженерия» Серек Азамат Галымжанович

4. Implementation

- Graph Construction: use networkx to build a directed graph.
    - *Figure 2 – Example graph construction from link data.*



- Network Metrics: calculate node degree, centrality, clustering coefficient.
    - *Figure 3 – Distribution of node degrees in the web graph.*

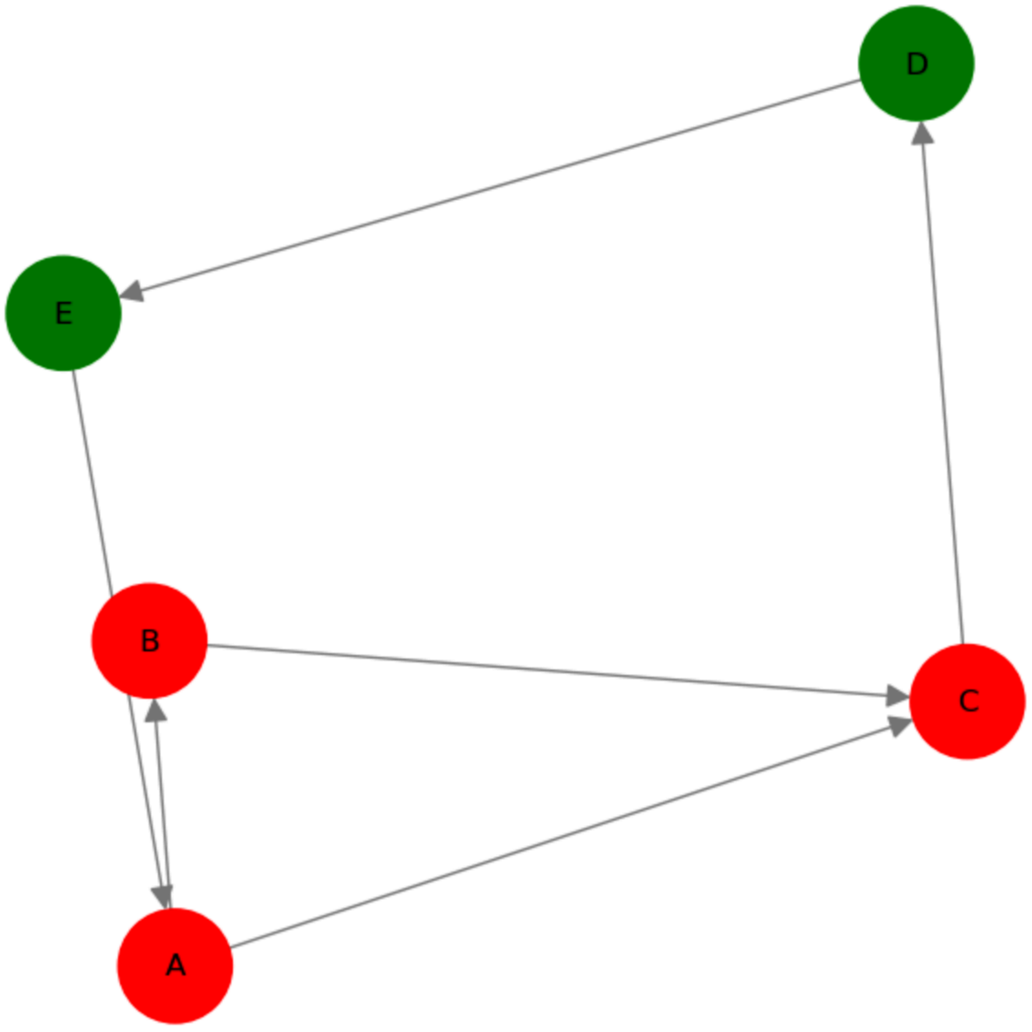


```
degree = dict(G.degree())
print("Degree:", degree)

betweenness = nx.betweenness_centrality(G)
print("Betweenness Centrality:", betweenness)

clustering = nx.clustering(G.to_undirected())
print("Clustering Coefficient:", clustering)
```

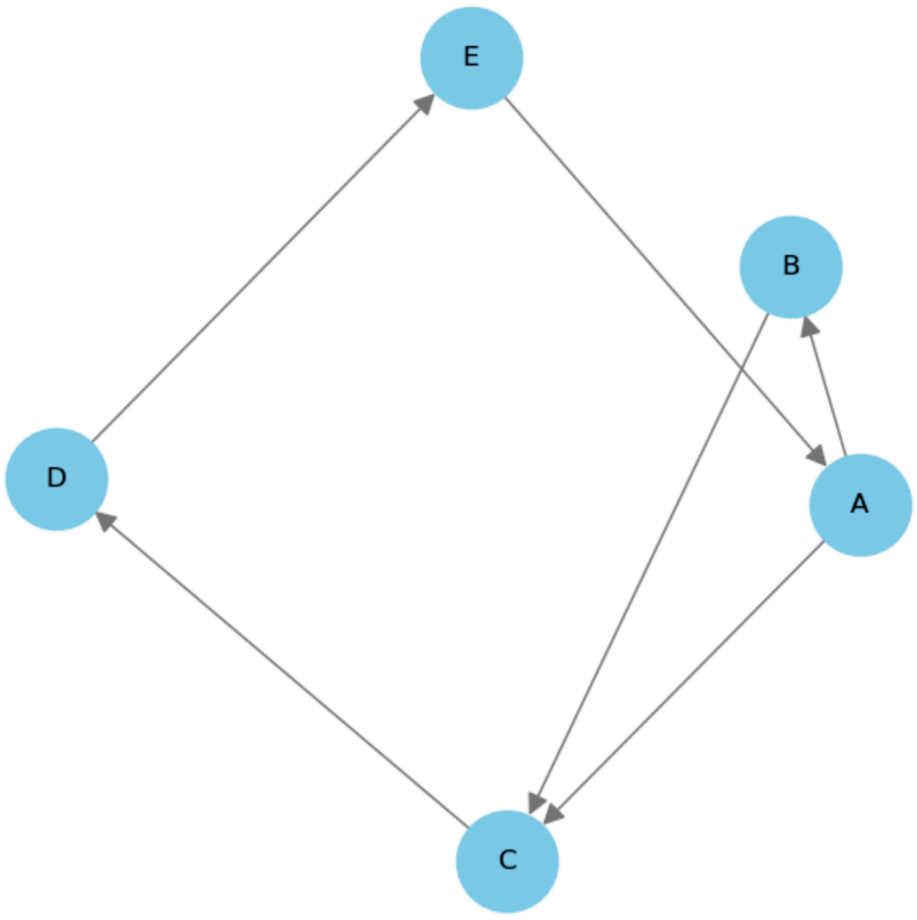
```
Degree: {'A': 3, 'B': 2, 'C': 3, 'D': 2, 'E': 2}
Betweenness Centrality: {'A': 0.5, 'B': 0.0, 'C': 0.5, 'D': 0.5, 'E': 0.5}
Clustering Coefficient: {'A': 0.3333333333333333, 'B': 1.0, 'C': 0.3333333333333333, 'D': 0, 'E': 0}
```

МОДУЛЬНОЕ ЗАДАНИЕ №4
по дисциплине «Анализ и обработка веб данных»
Преподаватель: *Ассоциированный профессор* каф. «Программная инженерия» Серек Азамат Галымжанович

- Community Detection: apply algorithms like Girvan-Newman (without ML).
    - *Figure 4 – Community structure of the web graph.*

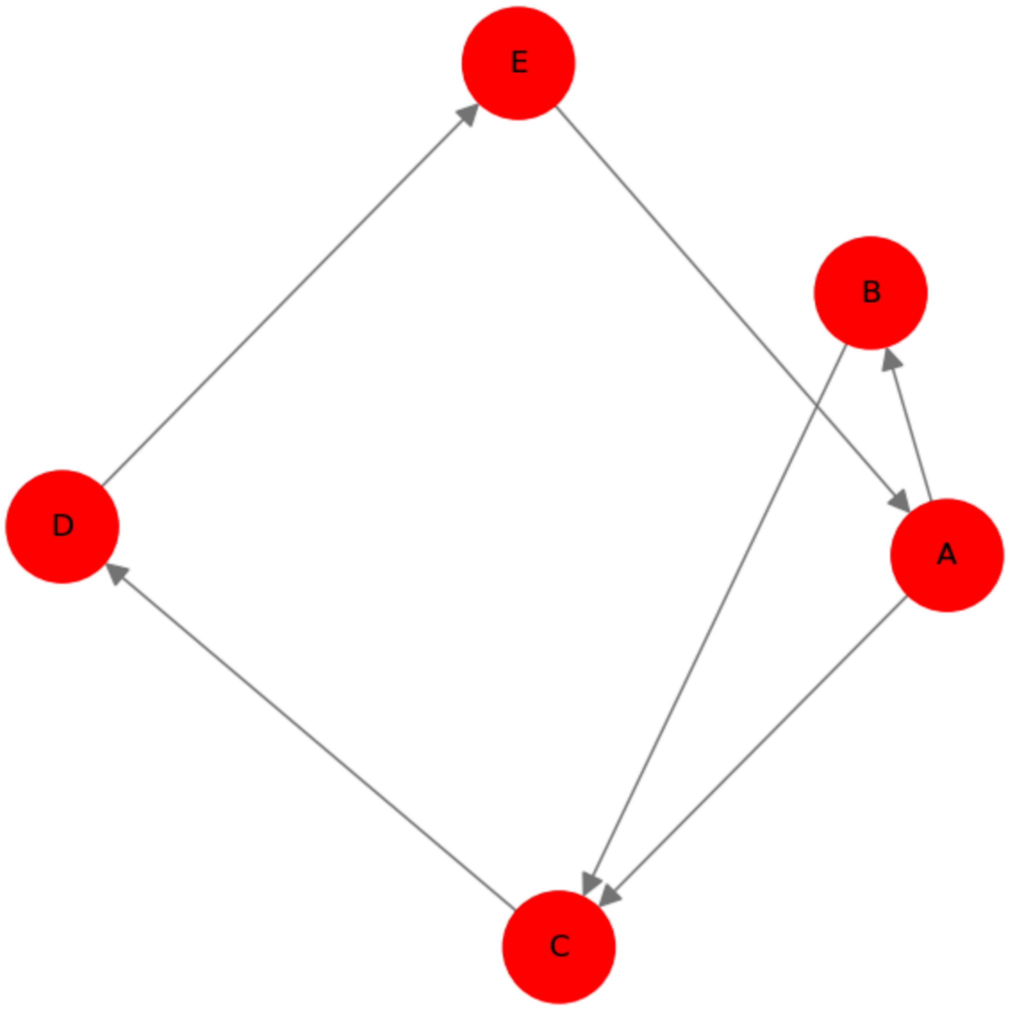


- Visualization:
    - Force-directed layout (spring layout).
        - *Figure 5 – Visualization of the web graph with force-directed layout.*



- Highlight important nodes (e.g., top-5 by centrality).

  - *Figure 6 – Highlighted key nodes in the web graph.*

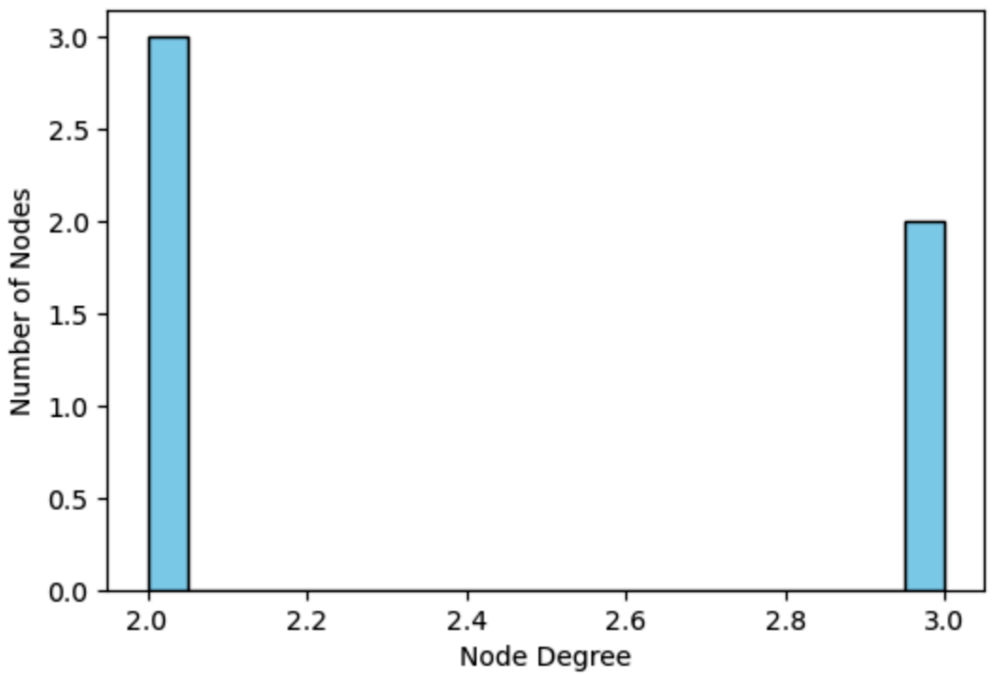


СӘТБАЕВ УНИВЕРСИТЕТІ SATBAYEV UNIVERSITY

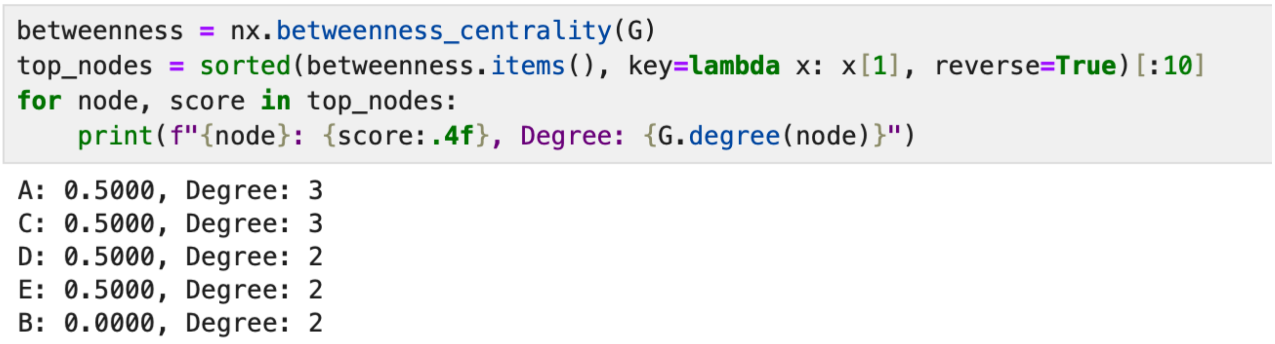
МОДУЛЬНОЕ ЗАДАНИЕ №4
по дисциплине «Анализ и обработка веб данных»
Преподаватель: *Ассоциированный профессор* каф. «Программная инженерия» Серек Азамат Галымжанович

5. Experiments and Results

- Dataset: ~500–1,000 nodes collected or imported.
- Degree Distribution: show power-law tendencies (many small-degree, few large hubs).
    - *Figure 7 – Degree distribution histogram.*



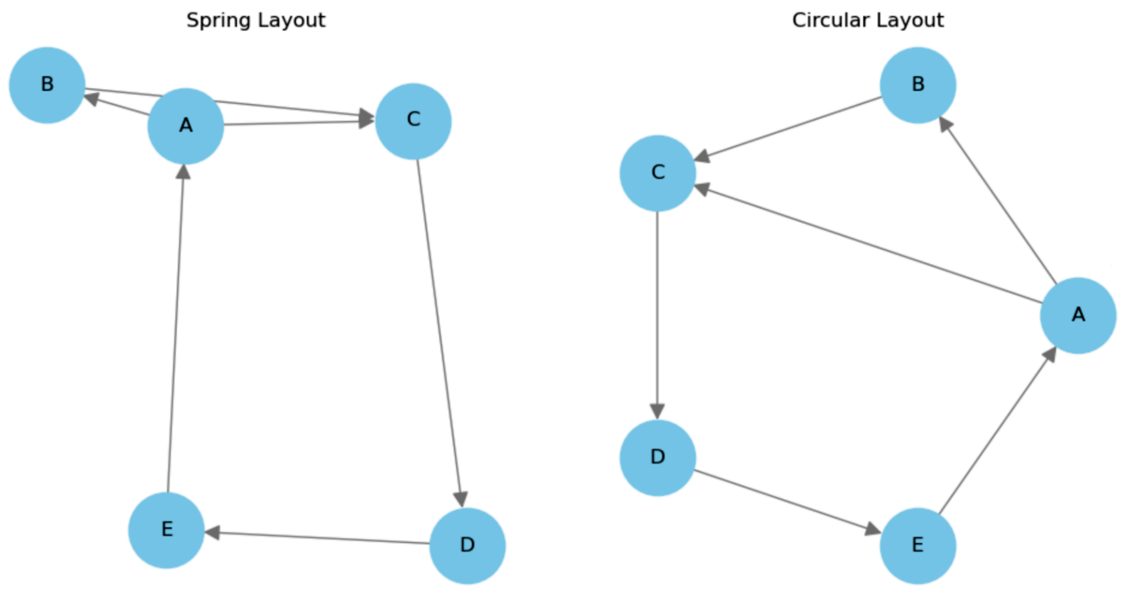
- Centrality Results: top nodes identified.
    - *Figure 8 – Table of top nodes by centrality score.*



```
betweenness = nx.betweenness_centrality(G)
top_nodes = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]
for node, score in top_nodes:
    print(f"{node}: {score:.4f}, Degree: {G.degree(node)}")
```

```
A: 0.5000, Degree: 3
C: 0.5000, Degree: 3
D: 0.5000, Degree: 2
E: 0.5000, Degree: 2
B: 0.0000, Degree: 2
```

- Visualization: compare layouts (spring vs circular).
    - *Figure 9 – Comparison of visualization layouts.*



6. Discussion

- Strengths: Visualization provides an intuitive understanding of graph structure. Central nodes and communities can be easily identified, as shown in Figures 5 and 6.
- Limitations: Large graphs (>100k nodes) are harder to visualize, and static plots may become unreadable. Scalable tools like Gephi or D3.js are recommended for such cases.
- Applications: Network analysis is useful for search engine ranking, detecting social media influencers, and mapping web structures.

7. Conclusion

- Key findings about the structure of the analyzed web graph.

The analysis revealed that the web graph contains a few highly connected hub nodes and many nodes with low degrees, indicating a power-law distribution. Communities are clearly formed, and central nodes act as bridges between them.

- Importance of combining analysis + visualization.

Combining quantitative analysis with visualization provides a deeper understanding of network structure, highlighting both key nodes and overall community organization.

- Future work: applying scalable visualization tools (Gephi, D3.js).

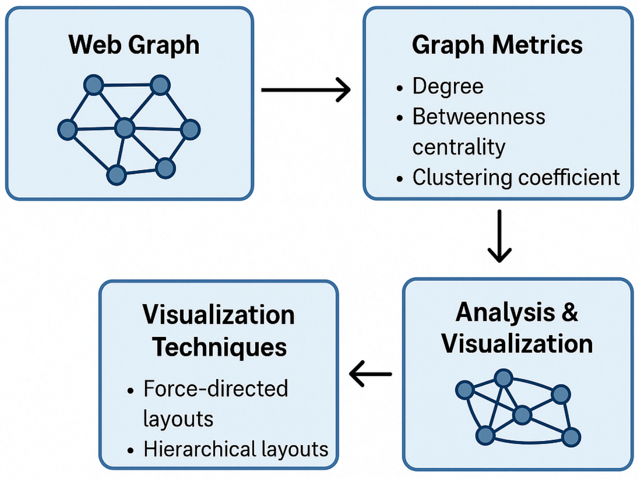
Future work could involve applying scalable visualization tools such as Gephi or D3.js, analyzing larger networks, and incorporating additional network metrics to gain further insights.

8. References

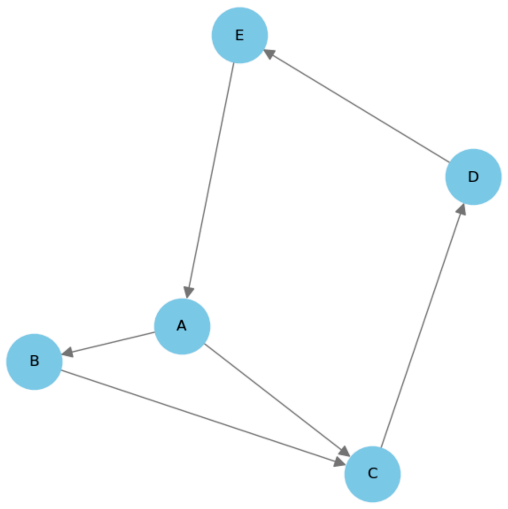
- NetworkX documentation.
- Graph theory textbooks.
- Research papers on web graph analysis.

📊 Figures (all with labels)

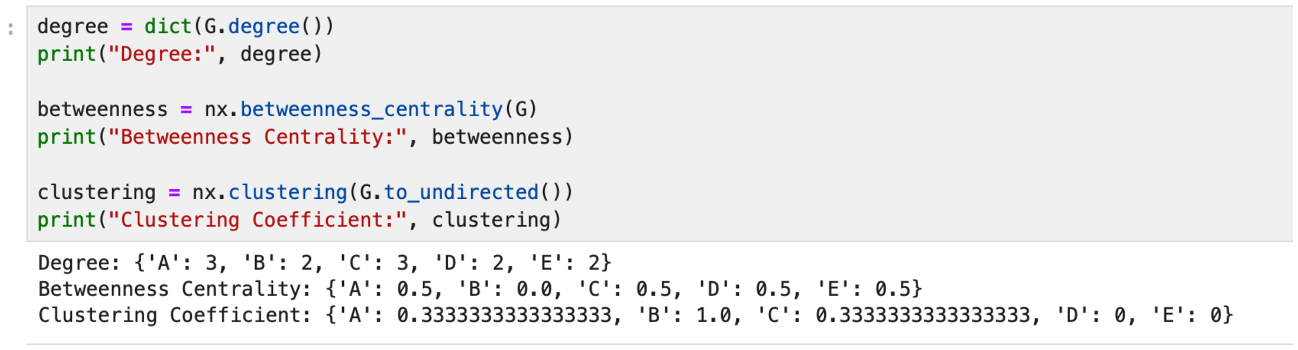
1. *Figure 1 – Workflow of web graph analysis and visualization.*



2. *Figure 2 – Example graph construction from link data.*



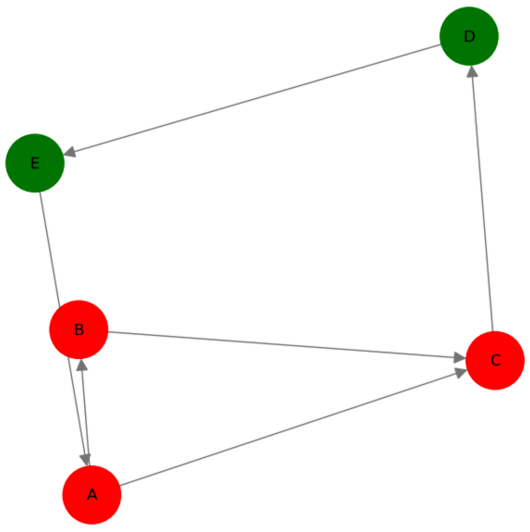
3. *Figure 3 – Distribution of node degrees in the web graph.*



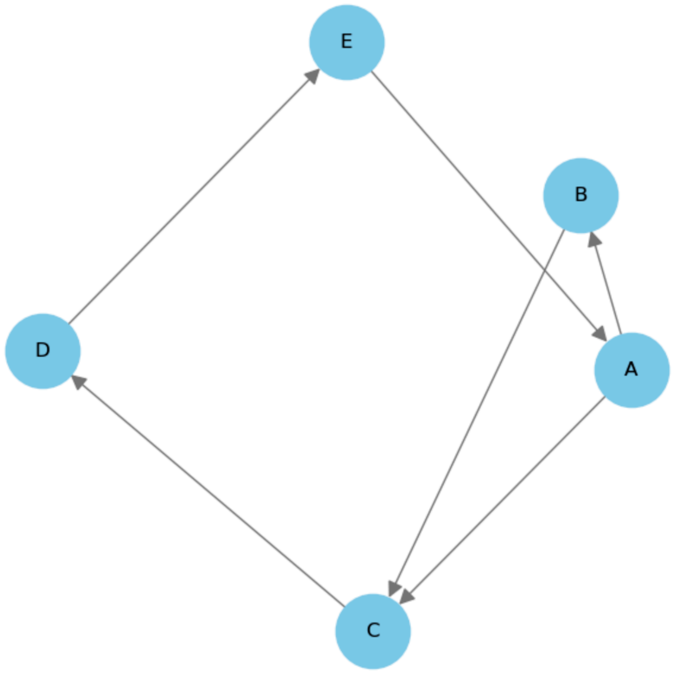
```
degree = dict(G.degree())
print("Degree:", degree)

betweenness = nx.betweenness_centrality(G)
print("Betweenness Centrality:", betweenness)

clustering = nx.clustering(G.to_undirected())
print("Clustering Coefficient:", clustering)
```

```
Degree: {'A': 3, 'B': 2, 'C': 3, 'D': 2, 'E': 2}
Betweenness Centrality: {'A': 0.5, 'B': 0.0, 'C': 0.5, 'D': 0.5, 'E': 0.5}
Clustering Coefficient: {'A': 0.3333333333333333, 'B': 1.0, 'C': 0.3333333333333333, 'D': 0, 'E': 0}
```

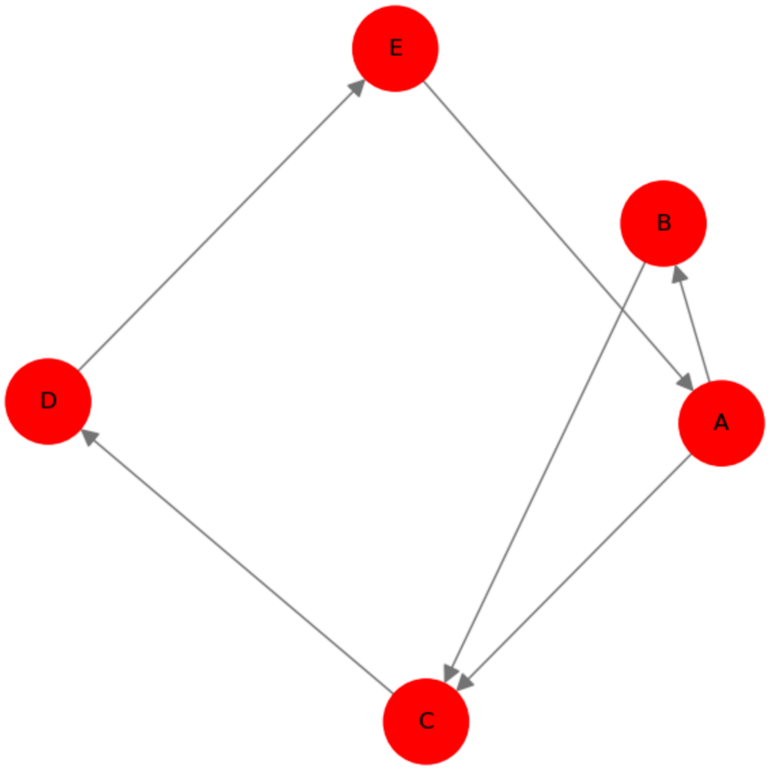
4. *Figure 4 – Community structure of the web graph.*



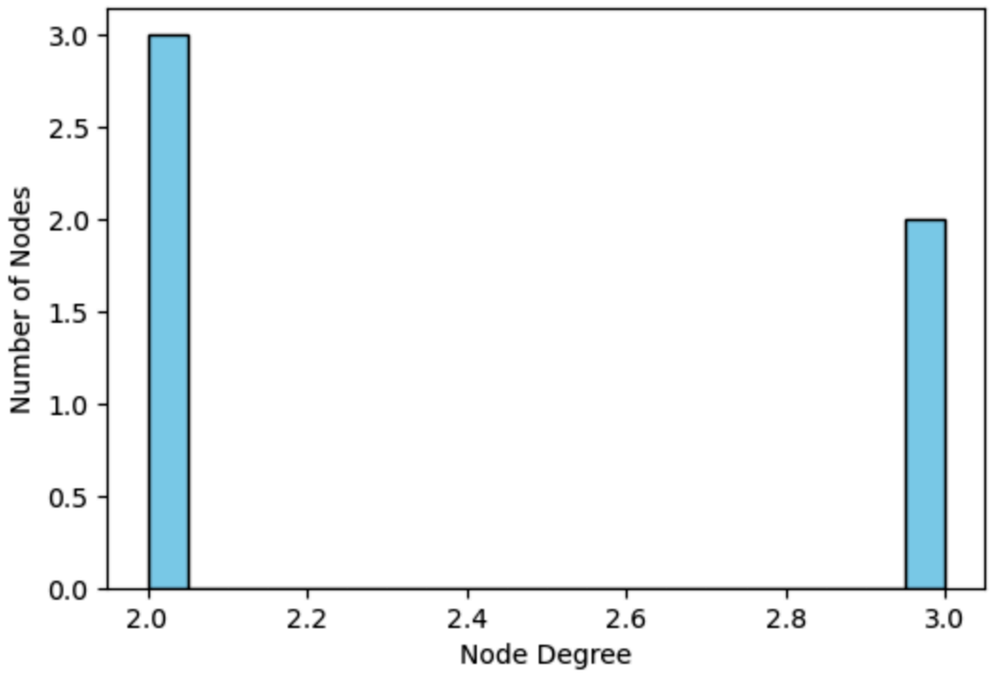
5. *Figure 5 – Visualization of the web graph with force-directed layout.*



6. *Figure 6 – Highlighted key nodes in the web graph.*



7. *Figure 7 – Degree distribution histogram.*

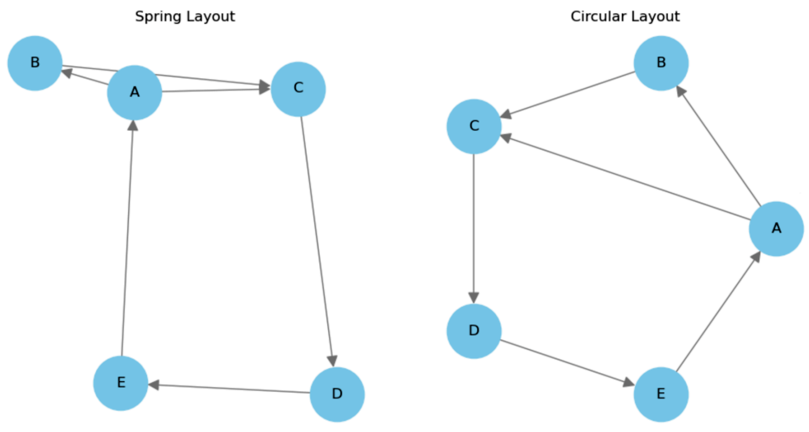


8. *Figure 8 – Table of top nodes by centrality score.*

```
betweenness = nx.betweenness_centrality(G)
top_nodes = sorted(betweenness.items(), key=lambda x: x[1], reverse=True)[:10]
for node, score in top_nodes:
    print(f"{node}: {score:.4f}, Degree: {G.degree(node)}")
```

```
A: 0.5000, Degree: 3
C: 0.5000, Degree: 3
D: 0.5000, Degree: 2
E: 0.5000, Degree: 2
B: 0.0000, Degree: 2
```

9. *Figure 9 – Comparison of visualization layouts.*


Spring Layout
Circular Layout
A
B
C
D
E