Abhishek Murthy

21BDS0064

Fall Sem 2024-2025

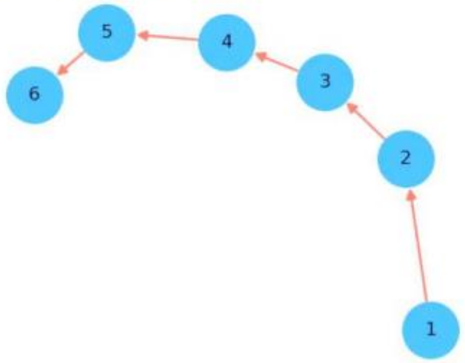
DA-6

Data Mining Lab

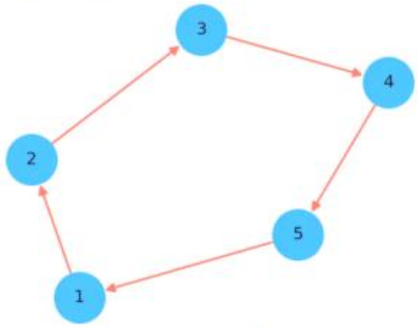
5-11-2024

Exercise

1. Perform the page rank computation for the following two graphs.
   Case 1 : No cycle graph



Case 2: Cyclic graph

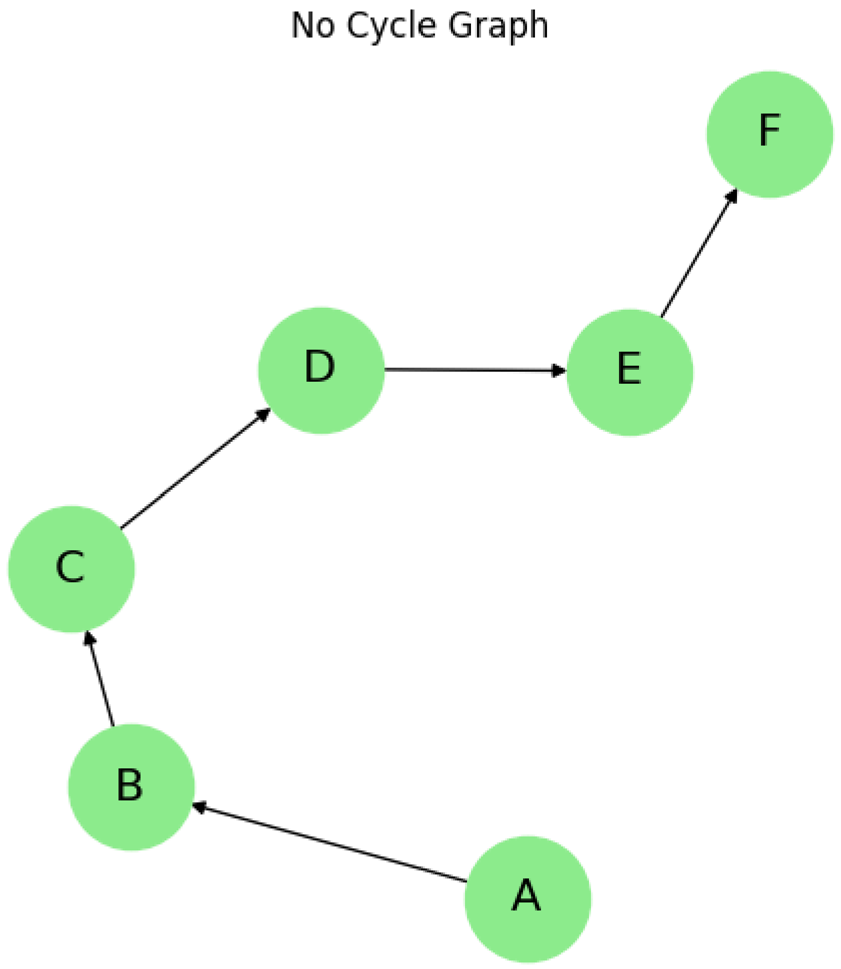


Display the graph and Print the page ranks for both the cases in descending order.

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

def compute_pagerank(matrix, damping=0.85, max_iterations=100, tolerance=1.0e-6)
    num_nodes = matrix.shape[0]
    rank_vector = np.ones(num_nodes) / num_nodes
    col_sums = matrix.sum(axis=0)
    transition_matrix = np.zeros(matrix.shape)

    for col in range(num_nodes):
        if col_sums[col] > 0:
            transition_matrix[:, col] = matrix[:, col] / col_sums[col]

    for _ in range(max_iterations):
        updated_rank = damping * np.dot(transition_matrix, rank_vector) + (1 - d
        if np.linalg.norm(updated_rank - rank_vector, 1) < tolerance:
            break
        rank_vector = updated_rank

    return rank_vector

# 1. No Cycle Graph
webpages_no_cycle = ['A', 'B', 'C', 'D', 'E', 'F']
links_no_cycle = [('A', 'B'), ('B', 'C'), ('C', 'D'), ('D', 'E'), ('E', 'F')]

# Adjacency matrix for No Cycle Graph
adj_matrix_no_cycle = np.zeros((len(webpages_no_cycle), len(webpages_no_cycle)))
for link in links_no_cycle:
    start_node = webpages_no_cycle.index(link[0])
    end_node = webpages_no_cycle.index(link[1])
    adj_matrix_no_cycle[start_node][end_node] = 1

print("Adjacency Matrix for No Cycle Graph:")
print(adj_matrix_no_cycle)

# Calculate PageRank for No Cycle Graph
pagerank_no_cycle = compute_pagerank(adj_matrix_no_cycle)
print("PageRank for No Cycle Graph:", pagerank_no_cycle)

# Plotting No Cycle Graph
graph_no_cycle = nx.DiGraph()
graph_no_cycle.add_edges_from(links_no_cycle)

plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
nx.draw(graph_no_cycle, with_labels=True, node_color='lightgreen', node_size=200
        font_size=16, font_color='black', arrows=True)
plt.title('No Cycle Graph')

# Displaying PageRank for No Cycle Graph in descending order
print("\nPageRank for No Cycle Graph (Descending Order):")
for idx in np.argsort(pagerank_no_cycle)[::-1]:
    print(f"Node {webpages_no_cycle[idx]}: {pagerank_no_cycle[idx]:.4f}")
```

```
Adjacency Matrix for No Cycle Graph:
[[0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0.]]
PageRank for No Cycle Graph: [0.10380841 0.09271578 0.07966563 0.0643125  0.04625
0.025     ]

PageRank for No Cycle Graph (Descending Order):
Node A: 0.1038
Node B: 0.0927
Node C: 0.0797
Node D: 0.0643
Node E: 0.0463
Node F: 0.0250
```

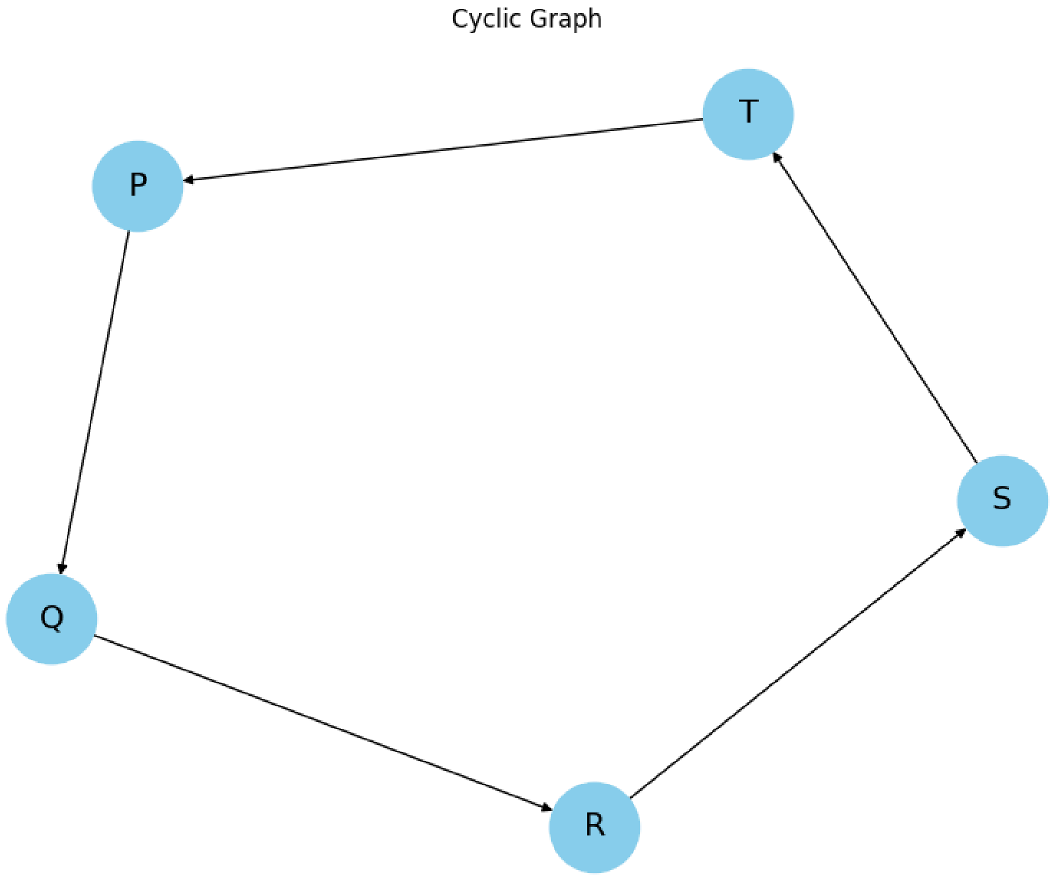

No Cycle Graph

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx

def compute_pagerank(matrix, damping=0.85, max_iterations=100, tolerance=1.0e-6)
    num_nodes = matrix.shape[0]
    rank_vector = np.ones(num_nodes) / num_nodes
    col_sums = matrix.sum(axis=0)
    transition_matrix = np.zeros(matrix.shape)

    for col in range(num_nodes):
        if col_sums[col] > 0:
```

```python
            transition_matrix[:, col] = matrix[:, col] / col_sums[col]

    for _ in range(max_iterations):
        updated_rank = damping * np.dot(transition_matrix, rank_vector) + (1 - d
        if np.linalg.norm(updated_rank - rank_vector, 1) < tolerance:
            break
        rank_vector = updated_rank

    return rank_vector

# 2. Cyclic Graph
webpages_cyclic = ['P', 'Q', 'R', 'S', 'T']
links_cyclic = [('P', 'Q'), ('Q', 'R'), ('R', 'S'), ('S', 'T'), ('T', 'P')]

# Adjacency matrix for Cyclic Graph
adj_matrix_cyclic = np.zeros((len(webpages_cyclic), len(webpages_cyclic)))
for link in links_cyclic:
    start_node = webpages_cyclic.index(link[0])
    end_node = webpages_cyclic.index(link[1])
    adj_matrix_cyclic[start_node][end_node] = 1

print("Adjacency Matrix for Cyclic Graph:")
print(adj_matrix_cyclic)

# Calculate PageRank for Cyclic Graph
pagerank_cyclic = compute_pagerank(adj_matrix_cyclic)
print("\nPageRank for Cyclic Graph:", pagerank_cyclic)

# Plotting Cyclic Graph
graph_cyclic = nx.DiGraph()
graph_cyclic.add_edges_from(links_cyclic)

plt.figure(figsize=(8, 6))
nx.draw(graph_cyclic, with_labels=True, node_color='skyblue', node_size=2000,
        font_size=16, font_color='black', arrows=True)
plt.title('Cyclic Graph')

# Displaying PageRank for Cyclic Graph in descending order
print("\nPageRank for Cyclic Graph (Descending Order):")
for idx in np.argsort(pagerank_cyclic)[::-1]:
    print(f"Node {webpages_cyclic[idx]}: {pagerank_cyclic[idx]:.4f}")

plt.tight_layout()
plt.show()
```

```
Adjacency Matrix for Cyclic Graph:
[[0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]
 [1. 0. 0. 0. 0.]]

PageRank for Cyclic Graph: [0.2 0.2 0.2 0.2 0.2]

PageRank for Cyclic Graph (Descending Order):
Node T: 0.2000
Node S: 0.2000
Node R: 0.2000
Node Q: 0.2000
Node P: 0.2000
```

```
<ipython-input-3-7a7099f62906>:55: UserWarning: This figure includes Axes that ar
e not compatible with tight_layout, so results might be incorrect.
  plt.tight_layout()
```



In [ ]: