Assignment 1

Name: Saikat Moi

Roll Number: 20CS10050

In [ ]:

```
# import all the necessary libraries here
import pandas as pd
import numpy as np
```

In [ ]:

```
edges_array = []
nodes=set()
```

In [ ]:

```
f = open("../../cora/cora.cites", "r")
for x in f:
  node1, node2 = map(int, x.split())
  edges_array.append((node1, node2))
  nodes.add(node1)
  nodes.add(node2)
```

In [ ]:

```
edges = np.array(edges_array)
```

Assumptions

For all the nodes which has outdegree 0 , considering it has edges to all the nodes from which it has incoming edges and accordingly creating the adjacency matrix for this problem

If you do not find any valid path between two nodes, you can assume a very large value (say 10^9) as the shortest distance.

In [ ]:

```
node_indices = list(nodes)

# Create an adjacency matrix initialized with zeros
num_nodes = len(node_indices)
adj_matrix = np.zeros((num_nodes, num_nodes), dtype=int)
outgoing_edges_count = np.zeros(num_nodes, dtype=int)

for edge in edges_array:
    node1, node2 = edge
    index1 = node_indices.index(node1)
    outgoing_edges_count[index1] += 1

# Populate the adjacency matrix based on the edges
for edge in edges_array:
    node1, node2 = edge
    index1, index2 = node_indices.index(node1), node_indices.index(node2)
    adj_matrix[index1, index2] = 1  # Assuming the graph is directed index1->ind
    if outgoing_edges_count[index2]==0:
        adj_matrix[index2, index1] = 1 # if an edge has outdegree 0 , then consi
        # which it has incoming edges
```

In [ ]:

```
adj = [[] for _ in range(num_nodes)]
for i in range(num_nodes):
    for j in range(num_nodes):
        if adj_matrix[i][j] == 1:
            adj[i].append(j)
```

Closeness Centrality

In [ ]:

```
from queue import Queue
dist_matrix = np.full((num_nodes, num_nodes), np.inf, dtype=float)
noofpaths = np.zeros((num_nodes, num_nodes), dtype=int)
closeness_centrality_list = []

for source_node in range(num_nodes):
    # Reset distances for each source node
    dist = [float('inf')] * num_nodes
    q = Queue()
    q.put((0, source_node))
    dist[source_node] = 0

    noofpaths[source_node][source_node] = 1

    while not q.empty():
        dis, node = q.get()

        for neighbor in adj[node]:
            if dis + 1 < dist[neighbor]:
                dist[neighbor] = dis + 1
                q.put((dist[neighbor], neighbor))
                noofpaths[neighbor][source_node] = noofpaths[source_node][node]
            elif dis + 1 == dist[neighbor]:
                noofpaths[neighbor][source_node] += noofpaths[source_node][node]

        # Calculate closeness centrality for the current source node
    dist_matrix[source_node, :] = dist
    valid_distances = [dist_i if dist_i != float('inf') else 1e9 for dist_i in d
    sum_of_distances = sum(valid_distances)
    closeness_centrality = (num_nodes - 1) / sum_of_distances if sum_of_distance
    closeness_centrality_list.append(closeness_centrality)
```

In [ ]:

```
with open('temp1.txt', 'w') as file:
    for index, score in enumerate(closeness_centrality_list):
        file.write(f"{node_indices[index]} {score:.12f}\n")

with open('temp1.txt', 'r') as file:
    lines = file.readlines()

# Sort the lines based on the score
sorted_lines = sorted(lines, key=lambda x: float(x.split(' ')[1]),reverse=True)

# Write the sorted content back to the file
with open('../../centralities/closeness.txt', 'w') as file:
    file.writelines(sorted_lines)

print("Closeness Centrality scores have been sorted and written to 'closeness.tx
```

```
Closeness Centrality scores have been sorted and written to 'closeness.txt'.
```

In [ ]:

```
len(node_indices)
```

Out[ ]:

```
2708
```

In [ ]:

```
V = list(range(num_nodes))  # List of vertices

# Create an empty adjacency list
A = {vertex: [] for vertex in V}

# Populate the adjacency list based on the adjacency matrix
for i in range(num_nodes):
    for j in range(num_nodes):
        if adj_matrix[i][j] == 1:
            A[i].append(j)
```

Betweenness Centrality

Using Brandes Algorithm for faster betweenness centrality calculation with scaling factor 1/((n-1)*(n-2))

In [ ]:

```
from collections import deque

def brandes(V, A):
    norm=((len(V)-1)*(len(V)-2)) # scaling factor 1/((n-1)*(n-2))
    C = dict((v,0) for v in V)
    for s in V:
        S = []
        P = dict((w,[]) for w in V)
        g = dict((t, 0) for t in V); g[s] = 1
        d = dict((t,-1) for t in V); d[s] = 0
        Q = deque([])
        Q.append(s)
        while Q:
            v = Q.popleft()
            S.append(v)
            for w in A[v]:
                if d[w] < 0:
                    Q.append(w)
                    d[w] = d[v] + 1
                if d[w] == d[v] + 1:
                    g[w] = g[w] + g[v]
                    P[w].append(v)
        e = dict((v, 0) for v in V)
        while S:
            # print(s)
            w = S.pop()
            for v in P[w]:
                e[v] = e[v] + ((g[v]/g[w]) * (1 + e[w]))
            if w != s:
                # print(w, e[w])
                C[w] = C[w] + e[w]

    for v in V:
        C[v] = C[v] /norm
    return C

betweenness_centrality_dic=brandes(V, A)
```

In [ ]:

```
with open('temp2.txt', 'w') as file:
    for node, score in betweenness_centrality_dic.items():
        file.write(f"{node_indices[node]} {score:.12f}\n")

with open('temp2.txt', 'r') as file:
    lines = file.readlines()

# Sort the lines based on the score
sorted_lines = sorted(lines, key=lambda x: float(x.split(' ')[1]), reverse=True)

# Write the sorted content back to the file
with open('../../centralities/betweenness.txt', 'w') as file:
    file.writelines(sorted_lines)

print("Betweenness Centrality scores have been sorted and written to 'betweennes
```

```
Betweenness Centrality scores have been sorted and written to 'betweenness.txt'.
```

Pagerank

In [ ]:

```
def pagerank_with_damping(adj_matrix, d=0.8, eps=1e-6, max_iter=100):
    N = len(adj_matrix)
    np.set_printoptions(threshold=np.inf)
    num_nodes=N
    # print(adj_matrix)
    # Create the row-normalized version of the adjacency matrix
    M = adj_matrix / adj_matrix.sum(axis=1, keepdims=True)
    #print(M)
    row_sums = np.sum(M, axis=1)
    temp_matrix = np.ones((num_nodes, num_nodes), dtype=int)/num_nodes
    M=(d)*M+(1-d)*temp_matrix
    #print(M)

    # Initialize PageRank scores
    R = np.ones(N) / N
    for _ in range(max_iter):
        R_new = np.dot(M.T, R)

        # Check for convergence
        if np.linalg.norm(R_new - R, 1) < eps:
            return R_new

        R = R_new

    return R

pagerank_scores = pagerank_with_damping(adj_matrix)
np.set_printoptions(suppress=True)
with open('temp3.txt', 'w') as file:
    for index, score in enumerate(pagerank_scores):
        file.write(f"{node_indices[index]} {score:.12f}\n")
```

In [ ]:

```
with open('temp3.txt', 'r') as file:
    lines = file.readlines()

# Sort the lines based on the score
sorted_lines = sorted(lines, key=lambda x: float(x.split(' ')[1]),reverse=True)

# Write the sorted content back to the file
with open('../../centralities/pagerank.txt', 'w') as file:
    file.writelines(sorted_lines)

print("PageRank scores have been sorted and written to 'pagerank.txt'.")
```

```
PageRank scores have been sorted and written to 'pagerank.txt'.
```