Name: Sancheet Kumar Baidya

Reg Num: 22MCB0029

Assignment 1

Social Network Analytics Lab

1. Reading a csv file and creating a data frame of source, destination and weights.

Ans.

Reading the csv file

```
In [2]: df = pd.read_csv('edges.csv')
```

Printing the dataframe

```
In [3]: df
```

Out[3]:

	Source	Destination	Weight
0	A	B	10
1	A	C	12
2	B	C	14
3	B	E	9
4	B	D	8
5	C	E	10
6	C	F	12
7	D	H	8
8	E	D	12
9	E	G	10
10	E	F	9
11	F	G	6
12	F	I	9
13	G	D	12
14	G	H	6
15	G	J	7
16	G	I	12
17	H	J	5
18	I	J	9

2. Function to create a directed and undirected graph

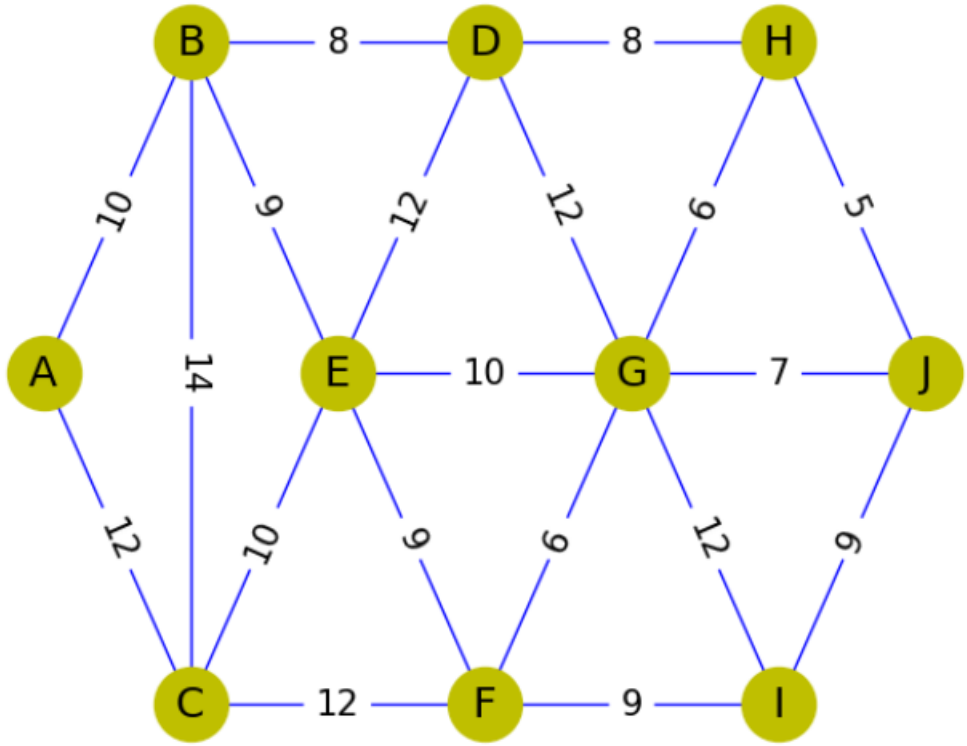
Ans.

```
In [4]: list_edges_weights = list(df.itertuples(index=False, name=None))
```

```
In [5]: def create_graph(graph):
            graph.add_weighted_edges_from(list_edges_weights)
            pos = {'A':[2,4], 'B':[4,7], 'C':[4, 1], 'D':[8,7], 'E':[6,4],
                   'F':[8,1], 'G':[10,4], 'H':[12,7] , 'I':[12,1], 'J':[14,4]}
            weight = nx.get_edge_attributes(G, 'weight')
            nx.draw(graph, pos=pos, with_labels=True,node_size= 1000, node_color='y',
                    edge_color='b', arrowsize=35, font_size=18)
            nx.draw_networkx_edge_labels(graph, pos, edge_labels=weight , font_size =15)
```

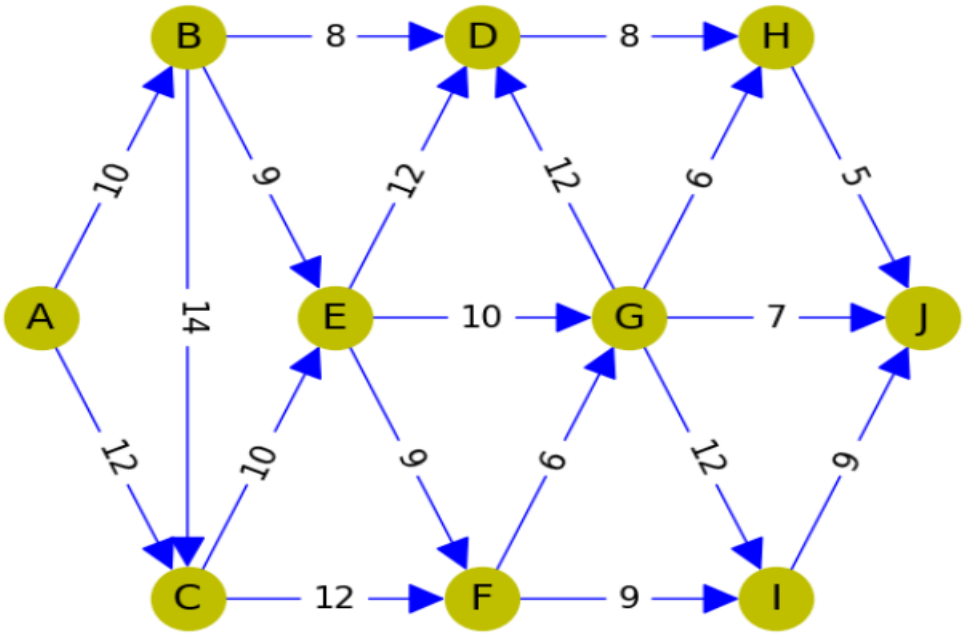
i) Undirected Graph

```
In [6]: # Create Undirected graph
        G = nx.Graph()
        create_graph(G)
```



ii) Directed Graph

```
In [7]: # Create directetd graph
        DG = nx.DiGraph()
        create_graph(DG)
```



3. <u>Finding the following parameters for the undirected graph</u>

Ans.

Q2) Find the following for the undirected graph

- Number of nodes
- Number of edges
- Node with maximum degree
- Node with minimum degree

```
In [8]: #number of nodes
        G.number_of_nodes()
```

```
Out[8]: 10
```

```
In [9]: #number of edges
        G.number_of_edges()
```

```
Out[9]: 19
```

```
In [10]: #Node with maximum degree
         max_degree = sorted(G.degree, key=lambda x: x[1], reverse=True)
         max_degree[0]
```

```
Out[10]: ('G', 6)
```

```
In [11]: #Node with minimum degree
         min_degree = sorted(G.degree, key=lambda x: x[1])
         min_degree[0]
```

```
Out[11]: ('A', 2)
```

4. <u>Find the following parameters for directed graph</u>

Ans.

Q3) Find the following for the directed graph

- Number of nodes
- Number of edges
- Nodes with maximum outdegree
- Nodes with minimum outdegree
- Nodes with maximum indegree
- Nodes with minimum indegree

In [12]:
```
#number of nodes
DG.number_of_nodes()
```

Out[12]: 10

In [13]:
```
#number of edges
DG.number_of_edges()
```

Out[13]: 19

In [15]:
```
#Node with maximum indegree
max_indegree = sorted(DG.in_degree, key=lambda x: x[1], reverse=True)
max_indegree[0]
```

Out[15]: ('D', 3)

In [16]:
```
#Node with minimum indegree
min_indegree = sorted(DG.in_degree, key=lambda x: x[1])
min_indegree[0]
```

Out[16]: ('A', 0)

In [17]:
```
#Node with maximum outdegree
max_outdegree = sorted(DG.out_degree, key=lambda x: x[1], reverse=True)
max_outdegree[0]
```

Out[17]: ('G', 4)

In [18]:
```
#Node with minimum outdegree
min_outdegree = sorted(DG.out_degree, key=lambda x: x[1])
min_outdegree[0]
```

Out[18]: ('J', 0)

5. Create adjacency matrix for directed and undirected graph. Also calculate the row and column sum.

Ans.

Q4) Create adjacency matrix for both undirected and directed graph

- Row sum for all nodes
- Row sum and column sum for all nodes

In [33]:
```
def adj_matrix(graph):
    return nx.adjacency_matrix(graph).todense()
```

In [29]:
```
def row_sum(graph, matrix):
    x = np.array(matrix)
    nodes = list(graph)
    k=0
    for i in x:
        row_sum = 0
        for j in i:
            row_sum += j

        print(f"Node {nodes[k]}: {row_sum}")
        k += 1
```

In [41]:
```
def col_sum(graph, matrix):
    x = np.array(matrix)
    nodes = list(graph)
    k=0
    for i in range(9):
        col_sum = 0
        for j in range(9):
            col_sum += x[j][i]

        print(f"Node {nodes[k]}: {col_sum}")
        k += 1
```

i) Undirected Graph Adjacency Matrix

In [34]:
```
# Adjacency matrix of undirected graph
adj_undir_matrix = adj_matrix(G)
adj_undir_matrix
```

Out[34]:
```
matrix([[ 0, 10, 12,  0,  0,  0,  0,  0,  0,  0],
        [10,  0, 14,  9,  8,  0,  0,  0,  0,  0],
        [12, 14,  0, 10,  0, 12,  0,  0,  0,  0],
        [ 0,  9, 10,  0, 12,  9,  0, 10,  0,  0],
        [ 0,  8,  0, 12,  0,  0,  8, 12,  0,  0],
        [ 0,  0, 12,  9,  0,  0,  0,  6,  9,  0],
        [ 0,  0,  0,  0,  8,  0,  0,  6,  0,  5],
        [ 0,  0,  0, 10, 12,  6,  6,  0, 12,  7],
        [ 0,  0,  0,  0,  0,  9,  0, 12,  0,  9],
        [ 0,  0,  0,  0,  0,  0,  5,  7,  9,  0]], dtype=int32)
```

ii) Row sum for undirected graph

In [31]:
```
#row sum
row_sum(G, adj_undir_matrix)
```

```
Node A: 22
Node B: 41
Node C: 48
Node E: 50
Node D: 40
Node F: 36
Node H: 19
Node G: 53
Node I: 30
Node J: 21
```

iii) Adjacency matrix for directed graph

In [35]:
```
# Adjacency matrix for directed graph
adj_dir_matrix = adj_matrix(DG)
adj_dir_matrix
```

Out[35]:
```
matrix([[ 0, 10, 12,  0,  0,  0,  0,  0,  0,  0],
        [ 0,  0, 14,  9,  8,  0,  0,  0,  0,  0],
        [ 0,  0,  0, 10,  0, 12,  0,  0,  0,  0],
        [ 0,  0,  0,  0, 12,  9,  0, 10,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  8,  0,  0,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  6,  9,  0],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  5],
        [ 0,  0,  0,  0, 12,  0,  6,  0, 12,  7],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  9],
        [ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0]], dtype=int32)
```

iv) Row sum of Nodes for Directed graph

In [42]:
```
# Row sum of adjacency matrix of directed graph
row_sum(DG, adj_dir_matrix)
```

```
Node A: 22
Node B: 31
Node C: 22
Node E: 31
Node D: 8
Node F: 15
Node H: 5
Node G: 37
Node I: 9
Node J: 0
```

v) Column sum of Nodes for Directed graph

In [43]:
```
# Column sum of adjacency matrix of directed graph
col_sum(DG, adj_dir_matrix)
```

```
Node A: 0
Node B: 10
Node C: 26
Node E: 19
Node D: 32
Node F: 21
Node H: 14
Node G: 16
Node I: 21
```

6. Calculating Centrality measures

Centrality measures are a vital tool for understanding networks, often also known as graphs.

These algorithms use graph theory to calculate the importance of any given node in a network. They cut through noisy data, revealing parts of the network that need attention – but they all work differently. Each measure has its own definition of 'importance', so you need to understand how they work to find the best one for your graph visualization applications.

i) Degree Centrality

Definition: Degree centrality assigns an importance score based simply on the number of links held by each node.

What it tells us: How many direct, 'one hop' connections each node has to other nodes in the network.

When to use it: For finding very connected individuals, popular individuals, individuals who are likely to hold most information or individuals who can quickly connect with the wider network.

A bit more detail: Degree centrality is the simplest measure of node connectivity. Sometimes it's useful to look at in-degree (number of inbound links) and out-degree (number of outbound links) as distinct measures, for example when looking at transactional data or account activity.

ii) Betweenness Centrality

Definition: Betweenness centrality measures the number of times a node lies on the shortest path between other nodes.

What it tells us: This measure shows which nodes are 'bridges' between nodes in a network. It does this by identifying all the shortest paths and then counting how many times each node falls on one.

When to use it: For finding the individuals who influence the flow around a system.

A bit more detail: Betweenness is useful for analysing communication dynamics, but should be used with care. A high betweenness count could indicate someone holds authority over disparate clusters in a network, or just that they are on the periphery of both clusters.

iii) Closeness Centrality

Definition: Closeness centrality scores each node based on their 'closeness' to all other nodes in the network.

What it tells us: This measure calculates the shortest paths between all nodes, then assigns each node a score based on its sum of shortest paths.

When to use it: For finding the individuals who are best placed to influence the entire network most quickly.

A bit more detail: Closeness centrality can help find good 'broadcasters', but in a highly-connected network, you will often find all nodes have a similar score. What may be more useful is using Closeness to find influencers in a single cluster.

iv) Page Rank

Definition: PageRank is a variant of Eigen Centrality, also assigning nodes a score based on their connections, and their connections' connections. The difference is that PageRank also takes link direction and weight into account – so links can only pass influence in one direction, and pass different amounts of influence.

What it tells us: This measure uncovers nodes whose influence extends beyond their direct connections into the wider network.

When to use it: Because it considers direction and connection weight, PageRank can be helpful for understanding citations and authority.

A bit more detail: PageRank is famously one of the ranking algorithms behind the original Google search engine (the 'Page' part of its name comes from creator and Google founder, Larry Page).

v) Eigen Centrality

Definition: Like degree centrality, Eigen Centrality measures a node's influence based on the number of links it has to other nodes in the network. Eigen Centrality then goes a step further by also considering how well connected a node is, and how many links their connections have, and so on through the network.

What it tells us: By calculating the extended connections of a node, Eigen Centrality can identify nodes with influence over the whole network, not just those directly connected to it.

When to use it: Eigen Centrality is a good 'all-round' SNA score, handy for understanding human social networks, but also for understanding networks like malware propagation.

vii) Centrality measures for Directed Graph

	Nodes	Degree_Centrality	Closeness_Centrality	Betweeness_Centrality	Page_Rank
0	A	0.22	0.00	0.00	0.04
1	B	0.44	0.11	0.04	0.05
2	C	0.44	0.22	0.06	0.07
3	E	0.56	0.25	0.09	0.08
4	D	0.44	0.44	0.05	0.10
5	F	0.44	0.30	0.08	0.09
6	H	0.33	0.39	0.02	0.13
7	G	0.67	0.31	0.10	0.09
8	I	0.33	0.33	0.01	0.11
9	J	0.33	0.47	0.00	0.25

viii) Centrality measures for Undirected Graph

	Nodes	Degree_Centrality	Closeness_Centrality	Betweeness_Centrality	Eigen_Vector_Centrality	Page_Rank
0	A	0.22	0.43	0.00	0.14	0.06
1	B	0.44	0.56	0.12	0.29	0.11
2	C	0.44	0.56	0.12	0.29	0.13
3	E	0.56	0.69	0.14	0.42	0.13
4	D	0.44	0.64	0.14	0.35	0.11
5	F	0.44	0.64	0.14	0.35	0.10
6	H	0.33	0.53	0.03	0.25	0.06
7	G	0.67	0.69	0.25	0.45	0.14
8	I	0.33	0.53	0.03	0.25	0.09
9	J	0.33	0.47	0.01	0.23	0.07

7. Comparison of Centrality Measures

i) For undirected graph

	Centrality	Min_Node	Min_score	Max_Node	Max_Score
0	Degree_Centrality	0	0.22	7	0.67
1	Closeness_Centrality	0	0.43	3	0.69
2	Betweeness_Centrality	0	0.00	7	0.25
3	Eigen_Vector_Centrality	0	0.14	7	0.45
4	Page_Rank	0	0.06	7	0.14

ii) For directed graph

	Centrality	Min_Node	Min_score	Max_Node	Max_Score
0	Degree_Centrality	0	0.22	7	0.67
1	Closeness_Centrality	0	0.00	9	0.47
2	Betweeness_Centrality	0	0.00	7	0.10
3	Page_Rank	0	0.04	9	0.25