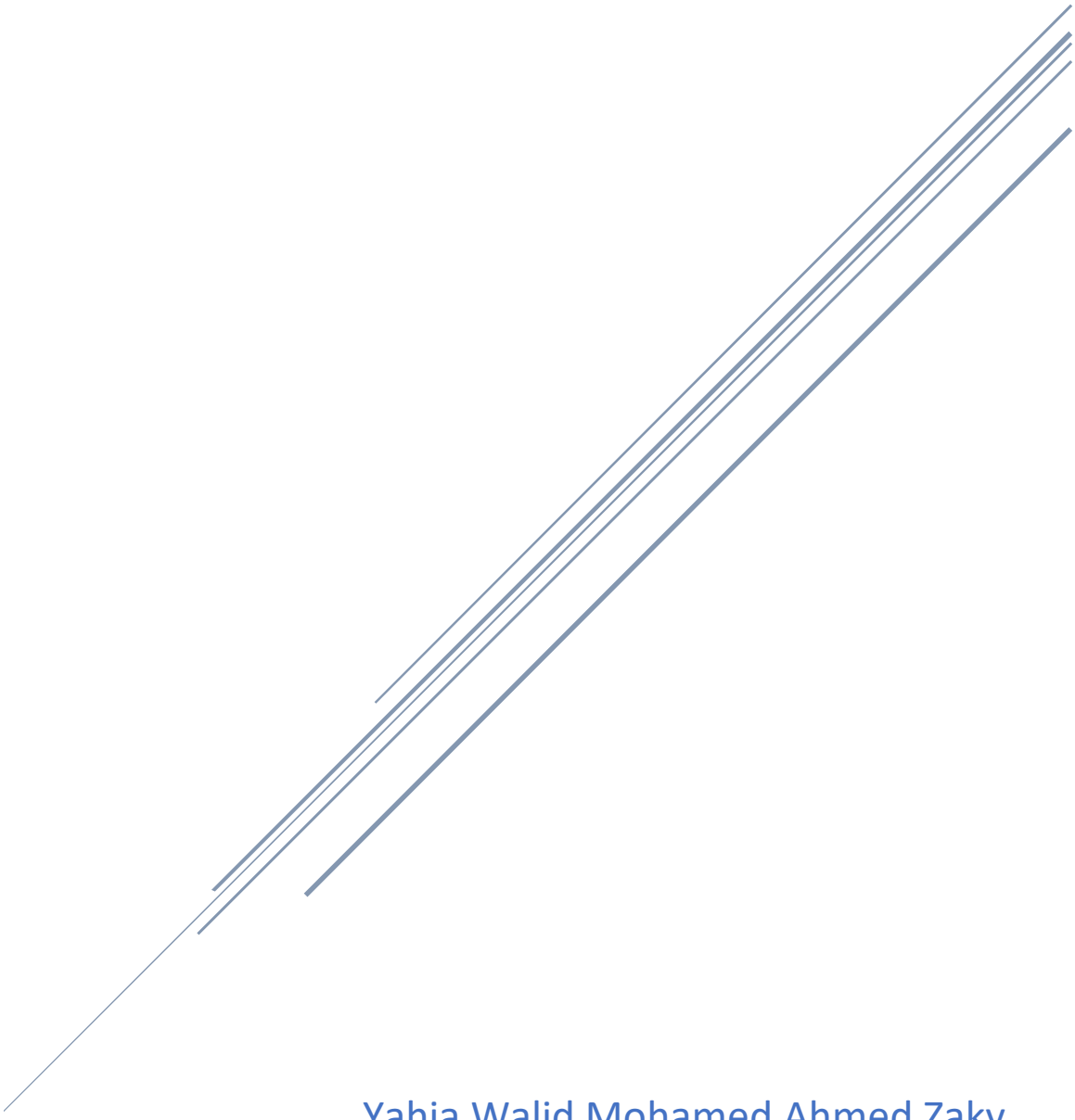
COMPUTER NETWORKS

Lab 3



Yahia Walid Mohamed Ahmed Zaky
Group-4 lab-1 ID-7137

INTRODUCTION

This report presents a network routing system implemented using Python and the NetworkX library. The system takes as input a set of edges and their associated weights and uses Dijkstra's algorithm to compute the shortest paths between each pair of nodes. The resulting forwarding tables for each node are then generated, providing a routing solution for the network. Finally, the network is visualized using the NetworkX graphing functions in Python. This report provides an overview of the implementation(including Dijkstra), highlighting the key features and performance of the system.

DISCUSSION

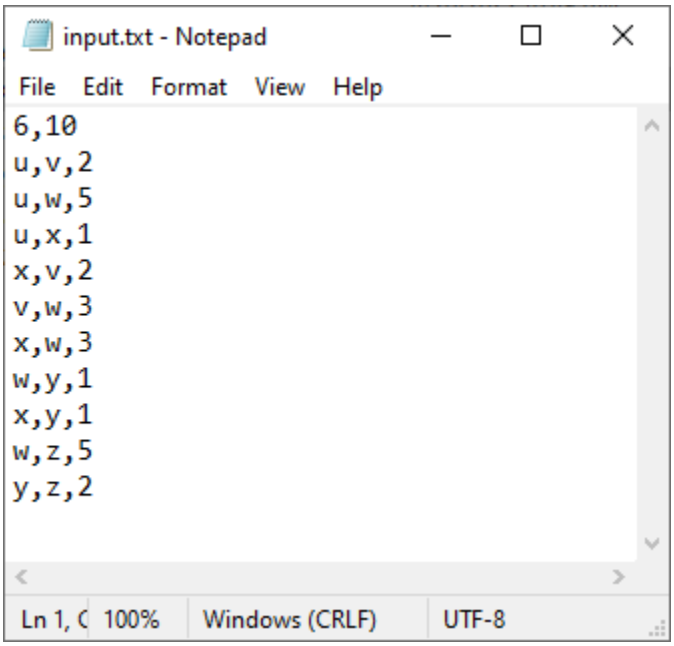
Code:

```
import networkx as nx
import matplotlib.pyplot as plt
import heapq


def dijkstra_shortest_path(G, start, end):

    dist = {node: float('inf') for node in G.nodes()}  # initializing
distances to all nodes to infinity
    dist[start] = 0  # ietting distance of start node to 0
    prev = {node: None for node in G.nodes()}  # initializing the previous
node for all nodes to none
    visited = set()  # initializing set of visited nodes
    heap = [(0, start)]  # initialize the heap with the start node and its
distance from itself which is 0

    while heap:
        (d, u) = heapq.heappop(heap)  # extract the node with the smallest
distance

        # if we reached the end node, return the shortest path
        if u == end:
            path = []
            while prev[u]:
                path.append(u)
                u = prev[u]
            path.append(start)
            return (d, path[::-1])

        # skipping nodes that have already been visited
        if u in visited:
            continue

        visited.add(u)  # mark the current node as visited

        for v in G.neighbors(u):  # For each neighbor of u

            if v in visited:  # Skip nodes that have already been visited
```

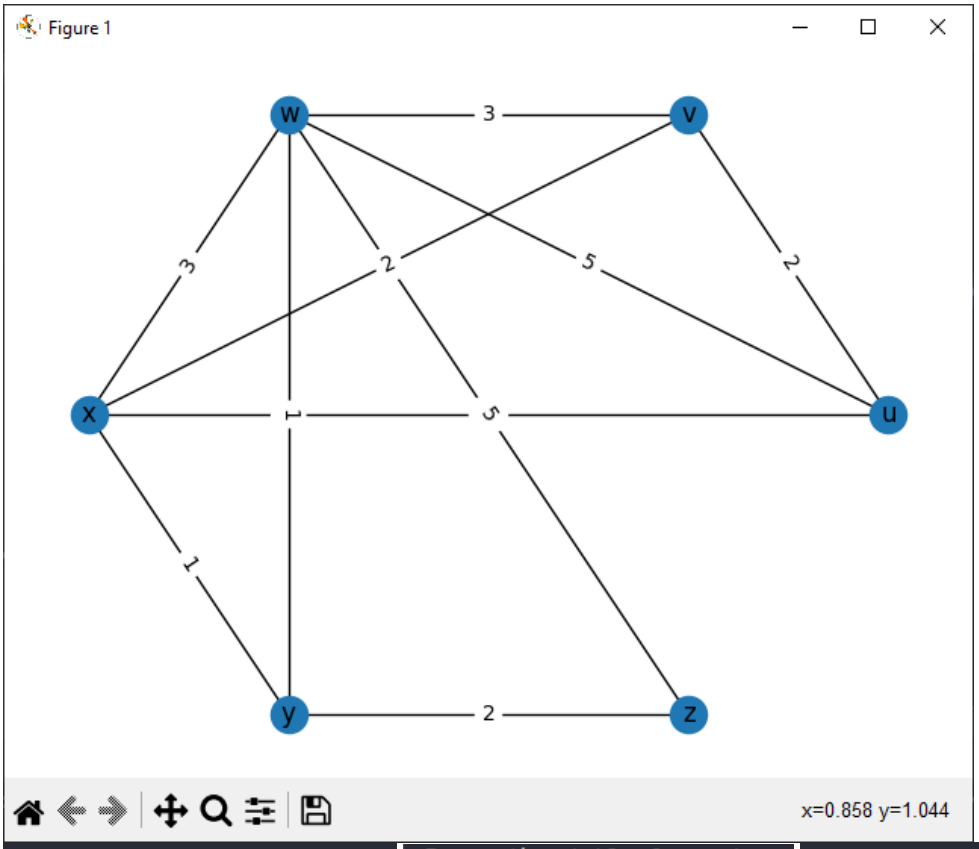
```
                continue

            alt = dist[u] + G[u][v]['weight']  # Compute the distance to the
neighbor

            if alt < dist[v]:  # If the new distance is shorter than the
current distance, update it
                dist[v] = alt
                prev[v] = u
                heapq.heappush(heap, (alt, v))  # Add the neighbor to the
heap

    return float('inf'), []  # If we haven't found a path ,return infinity
and an empty path



with open('input.txt') as f:
    # n--> number of nodes     m --> number of edges
    n, m = map(int, f.readline().strip().split(','))
    edges = [tuple(line.strip().split(',')) for line in f]

G = nx.Graph()          #simple undirected graph

for i in range(m) :
    G.add_edge(edges[i][0], edges[i][1], weight=int(edges[i][2]))

nx.draw_circular(G, with_labels=True)
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, nx.circular_layout(G),
edge_labels=edge_labels)
plt.show()

# forwarding table for each node
for node in G.nodes():
    print("Forwarding table for node", node)
    print("-------------------------")
    print("Destination\t\tLink")

    for dest in G.nodes():
        if dest == node:
            continue
        cost , path = dijkstra_shortest_path(G, node, dest)
        next_hop = path[1]
        #print(path)
        #print("Dest: %s, Next Hop: %s, Cost: %d" % (dest, next_hop, cost))
        print("%s\t\t\t\t(%s,%s)" %(dest,node,next_hop))

    print()
```

Sample run

Given the following input file:



Output:




```
Forwarding table for node w
------------------------
Destination     Link
u               (w,y)
v               (w,v)
x               (w,y)
y               (w,y)
z               (w,y)

Forwarding table for node x
------------------------
Destination     Link
u               (x,u)
v               (x,v)
w               (x,y)
y               (x,y)
z               (x,y)
```

```
Forwarding table for node u
------------------------
Destination     Link
v               (u,v)
w               (u,x)
x               (u,x)
y               (u,x)
z               (u,x)

Forwarding table for node v
------------------------
Destination     Link
u               (v,u)
w               (v,w)
x               (v,x)
y               (v,x)
z               (v,x)
```

```
Forwarding table for node y
------------------------
Destination     Link
u               (y,x)
v               (y,x)
w               (y,w)
x               (y,x)
z               (y,z)

Forwarding table for node z
------------------------
Destination     Link
u               (z,y)
v               (z,y)
w               (z,y)
x               (z,y)
y               (z,y)
```

CONCLUSION

In this project, we have implemented a routing algorithm using Dijkstra's shortest path algorithm in Python. We have used the NetworkX library to create a graph from an input file, and then used the Dijkstra function to compute the shortest path from each node to all other nodes in the graph. Finally, we printed the forwarding table for each node in the graph, which shows the next hop for each destination node.

Overall, this project demonstrates how routing algorithms can be implemented using Python and NetworkX, which can be useful in various networking applications. By understanding the fundamentals of Dijkstra's algorithm and applying it to network routing, we can optimize the communication process and ensure that data is transmitted efficiently across the network.

Python file drive link: https://drive.google.com/drive/u/0/folders/1QQYW3COyfmE23xwiWTsGnkQYRhlPuKUC