ENSF 462 Lab 05 Report **Completed by: Dominic Choi**
UCID: 30109955

router.py

```
import socket
import threading
import time
import sys
import json

def load_config(filename):
    with open(filename, "r") as file:
        lines = file.readlines()
        node_count = int(lines[0].strip())
        neighbors = []

        for line in lines[1:]:
            parts = line.strip().split()
            if len(parts) == 4:
                neighbors.append({
                        "label": parts[0],
                        "id": int(parts[1]),
                        "cost": int(parts[2]),
                        "port": int(parts[3]),
                    })

            else:
                if line.strip():  # if line is empty or spaces
                    print(f"Ignoring line: {line.strip()}")

        return node_count, neighbors

def dijkstra(graph, source):
    distance = {node: float('inf') for node in range(len(graph))}
    previous = {node: None for node in range(len(graph))}
    distance[source] = 0
    unvisited = set(range(len(graph)))

    while unvisited:
        current_node = min(unvisited, key=lambda node: distance[node])
        unvisited.remove(current_node)

        for neighbor_info in graph[current_node]:
            neighbor, cost = neighbor_info["id"], neighbor_info["cost"]
            temp_value = distance[current_node] + cost
```

```
            if temp_value < distance[neighbor]:
                distance[neighbor] = temp_value
                previous[neighbor] = current_node

    return distance, previous

def send_udp(message, host, port):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.sendto(message.encode(), (host, port))

def receive_udp(port):
    with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
        s.bind(("", port))
        message, _ = s.recvfrom(1024)

    return message.decode()

def send_link_state(router_id, neighbors):
    while True:
        message = json.dumps({"id": router_id, "neighbors": neighbors})
        for neighbor in neighbors:
            send_udp(message, "localhost", neighbor["port"])

        time.sleep(1)

def receive_and_broadcast_link_state(port, neighbors, link_state):
    while True:
        message = receive_udp(port)
        data = json.loads(message)
        link_state[data["id"]] = data["neighbors"]
        for neighbor in neighbors:
            send_udp(message, "localhost", neighbor["port"])

def print_routing_table(router_id, distance, previous, node_count):
    router_label = chr(router_id + ord("A"))

    # Print Dijkstra
    print("-----------------------------------")
    print("DestID Dist PrevID")

    for destination in range(node_count):
        if destination == router_id: #if self, distance = 0
            distance[destination] = 0
            prev_node_id = router_id
```

```
        else:
            prev_node_id = (previous[destination]
                            if previous[destination] is not None
                            else "-")

        print(f"{destination}      {distance[destination]}    {prev_node_id}")

    # Forwarding table
    print(f"\nThe forwarding table in {router_label} is printed as follows:")
    print("DestID NextLabel")

    for destination in range(node_count):
        if destination != router_id:
            next_hop = previous[destination]

            if next_hop == router_id: # if direct connection
                next_hop_label = chr(destination + ord("A"))

            else:
                while (previous[next_hop] is not None
                       and previous[next_hop] != router_id):

                    next_hop = previous[next_hop]

                next_hop_label = (chr(next_hop + ord("A"))
                                  if next_hop is not None else "None")

            print(f"{destination}      {next_hop_label}")

    print("-----------------------------------")

def main(router_id, router_port, config_file):
    node_count, neighbors = load_config(config_file)
    link_state = {router_id: neighbors}  # Initialize with self neighbors

    # Threads for each component
    send_thread = threading.Thread(
        target=send_link_state, args=(router_id, neighbors))

    receive_thread = threading.Thread(
        target=receive_and_broadcast_link_state,
        args=(router_port, neighbors, link_state))

    send_thread.start()
```

```
    receive_thread.start()

    while True:
        if len(link_state) == node_count:
            distance, previous = dijkstra(link_state, router_id)
            print_routing_table(router_id, distance, previous, node_count)
            return

        time.sleep(1)

if __name__ == "__main__":
    if len(sys.argv) != 4:
        print("Usage: python Router.py <router_id> <router_port>
<config_file>")
        sys.exit(1)
    router_id = int(sys.argv[1])
    router_port = int(sys.argv[2])
    config_file = sys.argv[3]

    main(router_id, router_port, config_file
```

Output: