**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:

A screenshot of a computer

Description automatically generated