WEEK\_TWO\_Programming Assignment:

# Task: Create some functions for a simplified BGP router

# Specifically, the withdraw, update, and next\_hop functions of the Router

# The class Route will be used.

#

# withdraw(rt) - rt is type Route. If a simplified BGP router gets this message, it will

#

class Route:

def \_\_init\_\_(self, neighbor, prefix, mask, path):

"""

:param neighbor: IP address of the next-hop neighbor (string)

:param prefix: Network prefix (string, e.g., "192.168.0.0")

:param mask: Subnet mask length (integer, e.g., 24 for /24)

:param path: AS path (list of integers)

"""

self.neighbor = neighbor

self.prefix = prefix

self.mask = mask

self.path = path

def pfx\_str(self):

"""

Returns the prefix in 'a.b.c.d/x' format.

"""

return f"{self.prefix}/{self.mask}"

def \_\_repr\_\_(self):

"""

String representation for debugging.

"""

return f"Route(neighbor={self.neighbor}, prefix={self.prefix}, mask={self.mask}, path={self.path})"

class Router:

def \_\_init\_\_(self):

self.rib = {} # Routing Information Base (RIB)

def print\_rib(self):

"""

Prints the RIB for debugging.

"""

for prefix, routes in self.rib.items():

print(f"{prefix}: {routes}")

def update(self, rt):

"""

Update the RIB with a given route.

:param rt: Route object

"""

prefix\_str = rt.pfx\_str() # Get prefix in "a.b.c.d/x" format

if prefix\_str not in self.rib:

self.rib[prefix\_str] = []

# Check if a route from the same neighbor exists

for i, existing\_route in enumerate(self.rib[prefix\_str]):

if existing\_route.neighbor == rt.neighbor:

self.rib[prefix\_str][i] = rt # Replace the route

return

# Add the new route

self.rib[prefix\_str].append(rt)

def withdraw(self, rt):

"""

Withdraw a route from the RIB.

:param rt: Route object

"""

prefix\_str = rt.pfx\_str()

if prefix\_str in self.rib:

# Remove routes matching the neighbor

self.rib[prefix\_str] = [

route for route in self.rib[prefix\_str]

if route.neighbor != rt.neighbor

]

# Remove prefix if no routes remain

if not self.rib[prefix\_str]:

del self.rib[prefix\_str]

def convert\_to\_binary(self, ip):

"""

Convert an IP address to a binary string.

"""

return ''.join(f"{int(octet):08b}" for octet in ip.split('.'))

def next\_hop(self, ipaddr):

"""

Find the next hop for a given IP address.

:param ipaddr: IP address (string)

:return: Next-hop IP address (string) or None if no match found

"""

binary\_ip = self.convert\_to\_binary(ipaddr)

best\_match = None

best\_prefix\_length = -1

for prefix\_str, routes in self.rib.items():

prefix, length = prefix\_str.split("/")

length = int(length)

binary\_prefix = self.convert\_to\_binary(prefix)

# Check if the IP matches the prefix

if binary\_ip[:length] == binary\_prefix[:length]:

# Prefer longer prefixes

if length > best\_prefix\_length:

best\_match = routes

best\_prefix\_length = length

if not best\_match:

return None # No matching route

# Find the best route (shortest AS path)

best\_route = min(best\_match, key=lambda route: len(route.path))

return best\_route.neighbor

def test\_router():

router = Router()

# Add routes

router.update(Route("1.1.1.1", "10.0.0.0", 24, [1, 2, 3]))

router.update(Route("2.2.2.2", "10.0.0.0", 24, [4, 5]))

router.update(Route("3.3.3.3", "10.0.0.0", 22, [6]))

router.update(Route("4.4.4.4", "12.0.0.0", 16, [7, 8, 9]))

# Print RIB

print("RIB after updates:")

router.print\_rib()

# Test next hop

print("Next hop for 10.0.0.1:", router.next\_hop("10.0.0.1"))

print("Next hop for 12.0.0.1:", router.next\_hop("12.0.0.1"))

print("Next hop for 192.168.1.1:", router.next\_hop("192.168.1.1"))

# Withdraw a route

router.withdraw(Route("1.1.1.1", "10.0.0.0", 24, [1, 2, 3]))

print("RIB after withdrawal:")

router.print\_rib()

if \_\_name\_\_ == "\_\_main\_\_":

test\_router()