```
# Mix Network
# By Amanda Flote & Olivia Mattsson
# !/usr/bin/env python
from pcapfile import savefile

def main():
    # Opens pcapfile and txt file with input
    testcap = open('HA2/cia.log.5.pcap', 'rb')
    capfile = savefile.load_savefile(testcap, layers=2, verbose=True)
    with open('HA2/quizB2.txt') as f:
        lines = f.read().splitlines()

        # The learning phase begins here:
        NazirIP = lines[0]
        MixIP = lines[1]
        NoParners = int(lines[2])
        NazirSent = False
        foundMix = False
        receivers = []
        for pkt in capfile.packets:
            ip_src = pkt.packet.payload.src.decode('UTF8')
            ip_dst = pkt.packet.payload.dst.decode('UTF8')
            # Found his IP, and next batch should be saved:
            if (NazirIP == ip_src):
                newreceivers = set()
                NazirSent = True
            # Found Mix IP after Nazir IP was found:
            if NazirSent and ip_src == MixIP:
                foundMix = True
            # We have looped through the entire batch, ads the batch to the total receivers and resets everything.
            if foundMix and ip_src != MixIP:
                receivers.append(newreceivers)
                foundMix = False
                NazirSent = False
            # Adds the destination to the batch
            elif foundMix:
                newreceivers.add(ip_dst)
        # Find the disjunct pairs of the receiver batches:
        disjunct = disReceivers(receivers , NoParners)
        # Finds the partners through the exlusion phase:
        partners = exclude(disjunct, receivers)
        # Sums the IP addresses in int
        ipSum = sumIP(partners)
        print(ipSum)

def disReceivers(receivers, noPartners):
    rec = receivers
    disjunct = []
    # As long as the disjunct list has less than noPartners:
    while len(disjunct) < noPartners:
        # Take each index in receivers and checks if it is disjunct with the disjunct list that is
        # already saved:
        for iList in range(len(rec)-1):
            if is_disjunct(disjunct, rec[iList]):
                # If it is, append the batch to the disjunct set:
                disjunct.append(rec[iList])
    return disjunct

# Checks if all of the list sets are disjoint with the newSet:
def is_disjunct(disList, newSet):
    for disSet in disList:
        if not disSet.isdisjoint(newSet):
            return False
    return True


# Excluding phase:
def exclude(dis, receivers):
    # For each receiver:
    for receiver in receivers:
        foundIP = -1
        sameIP = False
        # Loop over each disjoint set:
        for indexDis in range(len(dis)):
            # If the receiver is not disjoint, that means we found a match:
            if not receiver.isdisjoint(dis[indexDis]):
                # If we find a matching IP in the receiver set:
                if foundIP == -1:
                    foundIP = indexDis
                # If we already found IP, we set sameIP to true:
                else:
                    sameIP = True
                    break
        # If we have found an IP in a set and it's not same:
        if not sameIP and not foundIP == -1:
            # Replace the disjoint set with the intersection of the set with the same IP:
            dis[foundIP] = dis[foundIP].intersection(receiver)

        # Checks if all disjoint sets have just one IP left:
        singleIpLeft = True
        for s in dis:
            if len(s) != 1:
                singleIpLeft = False
        # If that's true, we stop the exclusion phase:
        if singleIpLeft:
            break

    return dis


# Summarizes the IP:
def sumIP(partners):
    ipSum = 0
    for s in partners:
        for ip in s:
            # Takes each ip in each set and splits it by the dots:
            split = ip.split(".")
            # Converts the ints to hexa and saves them in a string:
            for i in range(len(split)):
                split[i] = int_to_hexa(int(split[i]))
```

```
                while len(split[i]) < 2:
                    split[i] = '0' + split[i]
            hexstring = ''.join(split)
            ipSum += hexa_to_int(hexstring)

    return ipSum


# Hexadecimal string to int
def hexa_to_int(inputVal):
    return int(inputVal, 16)


# Int to hexadecimal string
def int_to_hexa(inputVal):
    return hex(inputVal).lstrip('0x')

if __name__ == '__main__':
    main()
```