############## Application #5 - Part #1 ##############

'''

Make the following configuration on each router in the network:

configure terminal

snmp-server community public RO

'''

Open a regular Linux terminal

Go to the folder containing the script, using cd /folder_path

Enter "sudo python OSPF_SNMP.py" and the password for the account

You may also need to configure the permissions on the script first! "chmod 755 script.py"

Check the console output for any errors

#Necessary Python packages (they are already installed on the Debian VM)

#https://pypi.python.org/pypi/setuptools

#https://pypi.python.org/pypi/networkx

#https://pypi.python.org/pypi/matplotlib

#https://pypi.python.org/pypi/pysnmp

#https://pypi.python.org/pypi/colorama

import pprint

import subprocess

import binascii

```
import sys

try:
    import matplotlib.pyplot as matp

except ImportError:
    print Fore.RED + Style.BRIGHT + "\n* Module matplotlib needs to be installed on your system."
    print "* Download it from: https://pypi.python.org/pypi/matplotlib\n" + Fore.WHITE + Style.BRIGHT
    sys.exit()

try:
    import networkx as nx

except ImportError:
    print Fore.RED + Style.BRIGHT + "\n* Module networkx needs to be installed on your system."
    print "* Download it from: https://pypi.python.org/pypi/networkx"
    print "* You should also install decorator: https://pypi.python.org/pypi/decorator\n" + Fore.WHITE + Style.BRIGHT
    sys.exit()

try:
    #Module for output coloring
    from colorama import init, deinit, Fore, Style

except ImportError:
    print Fore.RED + Style.BRIGHT + "\n* Module colorama needs to be installed on your system."
```

```
    print "* Download it from: https://pypi.python.org/pypi/colorama\n" + Fore.WHITE + Style.BRIGHT
    sys.exit()

try:
    #Module for SNMP
    from pysnmp.entity.rfc3413.oneliner import cmdgen

except ImportError:
    print Fore.RED + Style.BRIGHT + "\n* Module pysnmp needs to be installed on your system."
    print "* Download it from: https://pypi.python.org/pypi/pysnmp\n" + Fore.WHITE + Style.BRIGHT
    sys.exit()



#Initialize colorama
init()



#Prompting user for input
try:
    print Style.BRIGHT + "\n######################### OSPF DISCOVERY TOOL
########################"
    print "Make sure to connect to a device already running OSPF in the network!"
    print "SNMP community string should be the same on all devices running OSPF!\n"
    ip = raw_input(Fore.BLUE + Style.BRIGHT + "\n* Please enter root device IP: ")
```

```
    comm = raw_input("\n* Please enter community string: ")


except KeyboardInterrupt:
    print Fore.RED + Style.BRIGHT + "\n\n* Program aborted by user. Exiting...\n"
    sys.exit()


            ############## Application #5 - Part #2 ##############


#Checking IP address validity
def ip_is_valid():
    while True:
        #Checking octets
        a = ip.split('.')


        if (len(a) == 4) and (1 <= int(a[0]) <= 223) and (int(a[0]) != 127) and (int(a[0]) != 169 or int(a[1]) !=
254) and (0 <= int(a[1]) <= 255 and 0 <= int(a[2]) <= 255 and 0 <= int(a[3]) <= 255):
            break


        else:
            print '\n* There was an INVALID IP address! Please check and try again!\n'
            sys.exit()


    #Checking IP reachability
    print Fore.GREEN + Style.BRIGHT + "\n* Valid IP address. Checking IP reachability...\n"


    while True:
```

```
        ping_reply = subprocess.call(['ping', '-c', '3', '-w', '3', '-q', '-n', ip], stdout = subprocess.PIPE)


        if ping_reply == 0:
            print Fore.GREEN + Style.BRIGHT + "* Device is reachable. Performing SNMP extraction...\n"
            print Fore.GREEN + Style.BRIGHT + "* This may take a few moments...\n"
            break


        elif ping_reply == 2:
            print Fore.RED + Style.BRIGHT + "\n* No response from device %s." % ip
            sys.exit()


        else:
            print Fore.RED + Style.BRIGHT + "\n* Ping to the following device has FAILED:", ip
            print "\n"
            sys.exit()


#Change exception message
try:
    #Calling IP validity function
    ip_is_valid()


except KeyboardInterrupt:
    print Fore.RED + Style.BRIGHT + "\n\n* Program aborted by user. Exiting...\n"
    sys.exit()
```

```
ospf = []


#SNMP function
def snmp_get(ip):
    nbridlist = []
    nbriplist = []
    ospf_devices = {}


    #Creating command generator object
    cmdGen = cmdgen.CommandGenerator()


    #Performing SNMP GETNEXT operations on the OSPF OIDs
    #The basic syntax of nextCmd: nextCmd(authData, transportTarget, *varNames)
    #The nextCmd method returns a tuple of (errorIndication, errorStatus, errorIndex, varBindTable)


    errorIndication, errorStatus, errorIndex, varBindNbrTable =
cmdGen.nextCmd(cmdgen.CommunityData(comm),
                                                cmdgen.UdpTransportTarget((ip, 161)),
                                                '1.3.6.1.2.1.14.10.1.3')


    #print cmdGen.nextCmd(cmdgen.CommunityData(comm),cmdgen.UdpTransportTarget((ip,
161)),'1.3.6.1.2.1.14.10.1.3')
    #print varBindNbrTable
```

```
   errorIndication, errorStatus, errorIndex, varBindNbrIpTable =
cmdGen.nextCmd(cmdgen.CommunityData(comm),
                                                cmdgen.UdpTransportTarget((ip, 161)),
                                                '1.3.6.1.2.1.14.10.1.1')

   #print varBindNbrIpTable

   errorIndication, errorStatus, errorIndex, varBindHostTable =
cmdGen.nextCmd(cmdgen.CommunityData(comm),
                                               cmdgen.UdpTransportTarget((ip, 161)),
                                               '1.3.6.1.4.1.9.2.1.3')

   #print varBindHostTable

   errorIndication, errorStatus, errorIndex, varBindHostIdTable =
cmdGen.nextCmd(cmdgen.CommunityData(comm),
                                                cmdgen.UdpTransportTarget((ip, 161)),
                                                '1.3.6.1.2.1.14.1.1')

   #print varBindHostIdTable

   #Extract and print out the results
   for varBindNbrTableRow in varBindNbrTable:
      for oid, nbrid in varBindNbrTableRow:
         hex_string = binascii.hexlify(str(nbrid))
         #print hex_string
         octets = [hex_string[i:i+2] for i in range(0, len(hex_string), 2)]
```

```
        #print octets
        ip = [int(i, 16) for i in octets]
        #print ip
        nbr_r_id = '.'.join(str(i) for i in ip)
        #print nbr_r_id
        nbridlist.append(nbr_r_id)
        #print('%s = %s' % (oid, nbr_r_id))

for varBindNbrIpTableRow in varBindNbrIpTable:
    for oid, nbrip in varBindNbrIpTableRow:
        hex_string = binascii.hexlify(str(nbrip))
        octets = [hex_string[i:i+2] for i in range(0, len(hex_string), 2)]
        ip = [int(i, 16) for i in octets]
        nbr_ip = '.'.join(str(i) for i in ip)
        nbriplist.append(nbr_ip)
        #print('%s = %s' % (oid, nbr_ip))

for varBindHostTableRow in varBindHostTable:
    for oid, host in varBindHostTableRow:
        ospf_host = str(host)
        #print('%s = %s' % (oid, host))

for varBindHostIdTableRow in varBindHostIdTable:
    for oid, hostid in varBindHostIdTableRow:
        hex_string = binascii.hexlify(str(hostid))
```

```
        octets = [hex_string[i:i+2] for i in range(0, len(hex_string), 2)]
        ip = [int(i, 16) for i in octets]
        ospf_host_id = '.'.join(str(i) for i in ip)
        #print('%s = %s' % (oid, hostid))

    #Adding OSPF data by device in the ospf_device dictionary
    ospf_devices["Host"] = ospf_host
    ospf_devices["HostId"] = ospf_host_id
    ospf_devices["NbrRtrId"] = nbridlist
    ospf_devices["NbrRtrIp"] = nbriplist

    ospf.append(ospf_devices)

    return ospf

#Calling the function for the user specified IP address
ospf = snmp_get(ip)
#pprint.pprint(ospf)

            ############## Application #5 - Part #3 ##############

def find_unqueried_neighbors():

    #Host OSPF Router IDs
    all_host_ids = []
```

```
for n in range(0, len(ospf)):
    hid = ospf[n]["HostId"]
    all_host_ids.append(hid)

#print "HID"
#print all_host_ids
#print "\n"

#Neighbor OSPF Router IDs
all_nbr_ids = []

for n in range(0, len(ospf)):
    for each_nid in ospf[n]["NbrRtrId"]:

        if each_nid == "0.0.0.0":
            pass

        else:
            all_nbr_ids.append(each_nid)

#print "NBR"
#print all_nbr_ids
#print list(set(all_nbr_ids))
#print "\n"
```

```
#Determining which neighbors were not queried and adding them to a list
all_outsiders = []

for p in all_nbr_ids:

    if p not in all_host_ids:
        all_outsiders.append(p)

#print "OUT"
#print all_outsiders
#print "\n"

#Running the snmp_get() function for each unqueried neighbor
for q in all_outsiders:
    for r in range(0, len(ospf)):
        for index, s in enumerate(ospf[r]["NbrRtrId"]):
            #print index, s

            if q == s:
                new_ip = ospf[r]["NbrRtrIp"][index]
                snmp_get(new_ip)

            else:
                pass
```

```
    return all_host_ids, all_nbr_ids, ospf


            ############## Application #5 - Part #4 ##############


#Calling the function above
while True:


    if (len(list(set(find_unqueried_neighbors()[0]))) == len(list(set(find_unqueried_neighbors()[1])))):
        break


final_devices_list = find_unqueried_neighbors()[2]


#pprint.pprint(final_devices_list)


#Creating list of neighborships
neighborship_dict = {}


for each_dictionary in final_devices_list:
    for index, each_neighbor in enumerate(each_dictionary["NbrRtrId"]):


        each_tuple = (each_dictionary["HostId"], each_neighbor)
        neighborship_dict[each_tuple] = each_dictionary["NbrRtrIp"][index]


#pprint.pprint(neighborship_dict)
```

############## Application #5 - Part #5 ##############

```
while True:
    try:
        #User defined actions
        print Fore.BLUE + Style.BRIGHT + "* Please choose an action:\n\n1 - Display OSPF devices on the screen\n2 - Export OSPF devices to CSV file\n3 - Generate OSPF network topology\ne - Exit"
        user_choice = raw_input("\n* Enter your choice: ")
        print "\n"

        #Defining actions
        if user_choice == "1":

            for each_dict in final_devices_list:
                print "Hostname: " + Fore.YELLOW + Style.BRIGHT + "%s" % each_dict["Host"] + Fore.BLUE + Style.BRIGHT
                print "OSFP RID: " + Fore.YELLOW + Style.BRIGHT + "%s" % each_dict["HostId"] + Fore.BLUE + Style.BRIGHT
                print "OSPF Neighbors by ID: " + Fore.YELLOW + Style.BRIGHT + "%s" % ', '.join(each_dict["NbrRtrId"]) + Fore.BLUE + Style.BRIGHT
                print "OSPF Neighbors by IP: " + Fore.YELLOW + Style.BRIGHT + "%s" % ', '.join(each_dict["NbrRtrIp"]) + Fore.BLUE + Style.BRIGHT
                print "\n"

            continue
```

```
    #Printing devices to CSV file

    elif user_choice == "2":

        print Fore.CYAN + Style.BRIGHT + "* Generating " + Fore.YELLOW + Style.BRIGHT +
"OSPF_DEVICES" + Fore.CYAN + Style.BRIGHT + " file...\n"

        print Fore.CYAN + Style.BRIGHT + "* Check the script folder. Import the file into Excel for a better
view of the devices.\n"


        csv_file = open("OSPF_DEVICES.txt", "w")


        print >>csv_file, "Hostname" + ";" + "OSPFRouterID" + ";" + "OSPFNeighborRouterID" + ";" +
"OSPFNeighborIP"


        for each_dict in final_devices_list:

            print >>csv_file, each_dict["Host"] + ";" + each_dict["HostId"] + ";" + ',
'.join(each_dict["NbrRtrId"]) + ";" + ', '.join(each_dict["NbrRtrIp"])


             csv_file.close()


        continue


                ############## Application #5 - Part #6 ##############


    #Generating OSPF network topology

    elif user_choice == "3":

        print Fore.CYAN + Style.BRIGHT + "* Generating OSPF network topology...\n" + Fore.BLUE +
Style.BRIGHT
```

```
            #Drawing the topology using the list of neighborships
            G = nx.Graph()
            G.add_edges_from(neighborship_dict.keys())
            pos = nx.spring_layout(G, k = 0.1, iterations = 70)
            nx.draw_networkx_labels(G, pos, font_size = 9, font_family = "sans-serif", font_weight = "bold")
            nx.draw_networkx_edges(G, pos, width = 4, alpha = 0.4, edge_color = 'black')
            nx.draw_networkx_edge_labels(G, pos, neighborship_dict, label_pos = 0.3, font_size = 6)
            nx.draw(G, pos, node_size = 700, with_labels = False)
            matp.show()

            continue

        elif user_choice == "e":
            print Fore.RED + Style.BRIGHT + "* Exiting... Bye!\n"
            sys.exit()

        else:
            print Fore.RED + Style.BRIGHT + "* Invalid option. Please retry.\n"
            continue

    except KeyboardInterrupt:
        print Fore.RED + Style.BRIGHT + "\n\n* Program aborted by user. Exiting...\n"
        sys.exit()

#De-initialize colorama
```

```
deinit()


#End of program
```