Educate Elevate Enlighten

JSS Mahavidyapeetha

JSS Science And Technology University

(Established Under JSS Science and Technology University Act No. 43 of 2013)

(Formerly Known as SJCE)

JSS

SCIENCE AND TECHNOLOGY UNIVERSITY

MYSURU

Report on Event 2 And 4 of Lab Component

Computer Networks (EC620L)

Title: Computer Networks

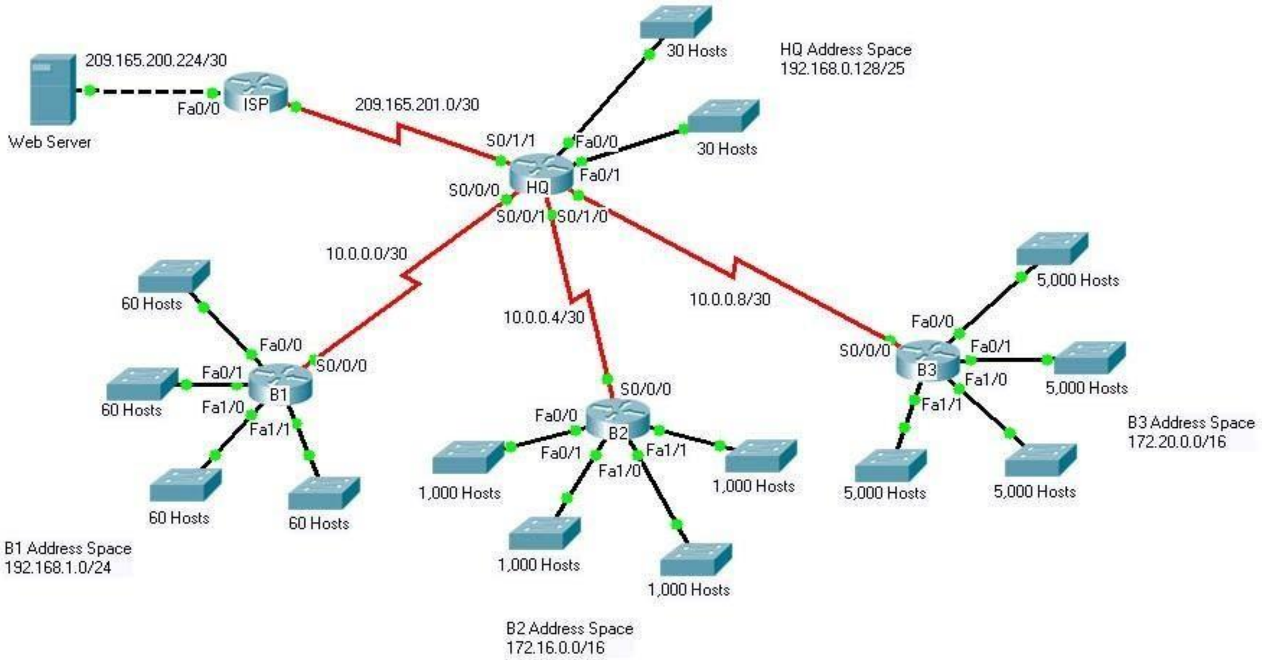
Name:	Vivan Sanjay Athreya
Roll No:	58
USN:	01JST18EC120
Division:	A

Submitted to

Prof. Kavyashree M K
Assistant Professor, Dept. of ECE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION
ENGINEERING
2021

1. SUBNETTING



Topology Diagram

Objectives

1. Design and document an addressing scheme based on requirements.
2. Select appropriate equipment and cable the devices.
3. Apply a basic configuration to the devices.
4. Configure static and default routing.
5. Verify full connectivity between all devices in the topology.

This activity focuses on Subnetting skills, basic device configurations and static routing. Once you have configured all devices, you will test for end to end connectivity and examine your configuration.

Task 1: Design and document an addressing scheme.

Step 1: Design an addressing scheme.

Based on the network requirements shown in the topology, design an appropriate addressing scheme.

- The HQ, B1, B2, and B3 routers each have an address space. Subnet the address space basedon the hostrequirements.
- For each address space, assign subnet zero to the Fa0/0 LAN, subnet 1 to the Fa0/1, and soon.

Step 2: Document the addressing scheme.

- Document the IP addresses and subnet masks. Assign the first IP address to the routerinterface.
- For the WAN links, assign the first IP address toHQ.

Task 2: Apply a basic configuration.

Using your documentation, configure the routers with basic configurations including addressing and hostnames. Use 64000 as the clock rate. ISP is the DCE in its WAN link to HQ. HQ is the DCE for all other WAN links.

Task 3: Configure static and default routing

Configure static and default routing using the exit interface argument.

- HQ should have three static routes and one default route.
- B1, B2, and B3 should have one default route.
- ISP should have seven static routes. This will include the three WAN links between HQ and the branch routers B1, B2, and B3.

Task 5: Test connectivity and examine the configuration.

Step 1: Test connectivity.

- You should now have end-to-end connectivity. Use ping to test connectivity across the network. Each router should be able to ping all other router interfaces and the Web Server.
- Use extended ping to test LAN connectivity to the Web Server. For example, the test the Fa0/0 interface on B1, you would do the following:

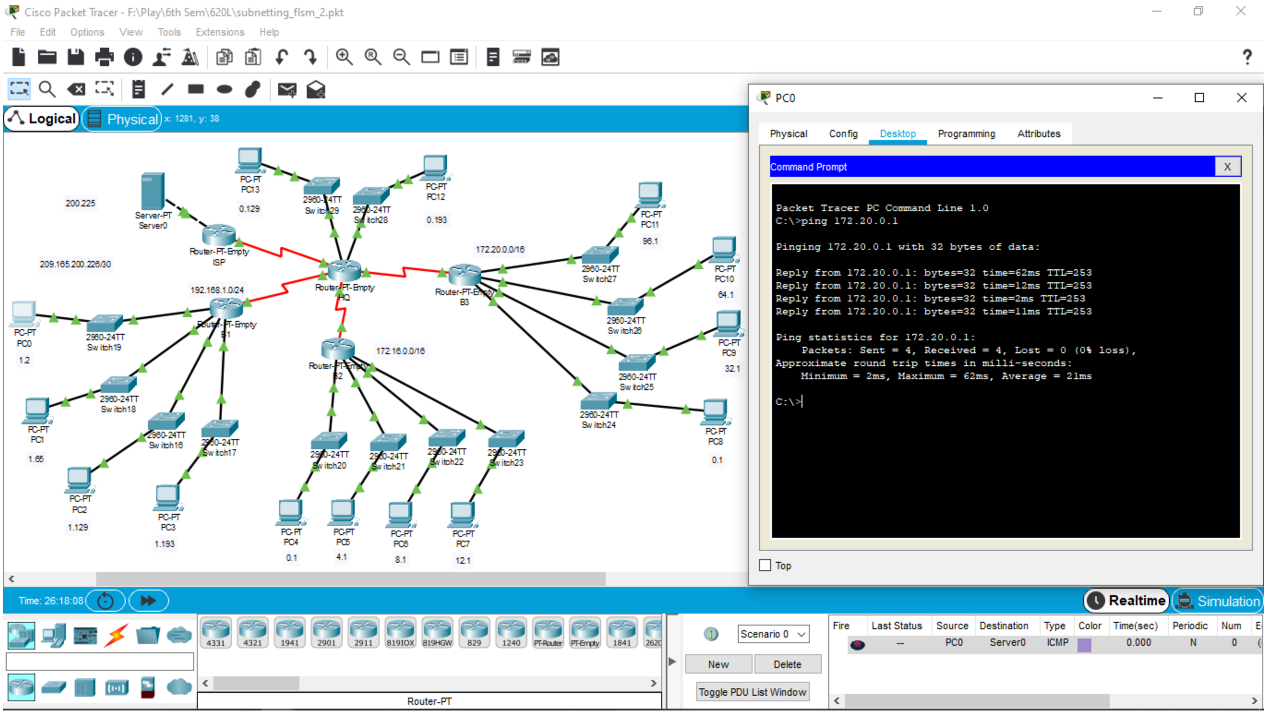
- Troubleshoot until pings are successful.

Step 2: Examine the configuration.

Use verification commands to make sure your configurations are complete.

Addressing Table

Device	Interface	IP Address	Subnet Mask
HQ	Fa0/0	192.168.0.129	255.255.255.192
	Fa0/1	192.168.0.129	255.255.255.192
	S0/0/0	10.0.0.1	255.255.255.252
	S0/0/1	10.0.0.5	255.255.255.252
	S0/1/0	10.0.0.9	255.255.255.252
	S0/1/1	209.165.201.2	255.255.255.252
B1	Fa0/0	192.168.1.1	255.255.255.192
	Fa0/1	192.168.1.65	255.255.255.192
	Fa1/0	192.168.1.129	255.255.255.192
	Fa1/1	192.168.1.193	255.255.255.192
	S0/0/0	10.0.0.2	255.255.255.252
B2	Fa0/0	172.16.0.1	255.255.252.0
	Fa0/1	172.16.4.1	255.255.252.0
	Fa1/0	172.16.8.1	255.255.252.0
	Fa1/1	172.16.12.1	255.255.252.0
	S0/0/0	10.0.0.6	255.255.255.252
B3	Fa0/0	172.20.0.1	255.255.224.0
	Fa0/1	172.20.32.1	255.255.224.0
	Fa1/0	172.20.64.1	255.255.224.0
	Fa1/1	172.20.128.1	255.255.224.0
	S0/0/0	10.0.0.10	255.255.255.252
ISP	S0/0/0	209.165.201.1	255.255.255.252
	Fa0/0	209.165.200.225	255.255.255.252
Web Server	NIC	209.165.200.226	255.255.255.252

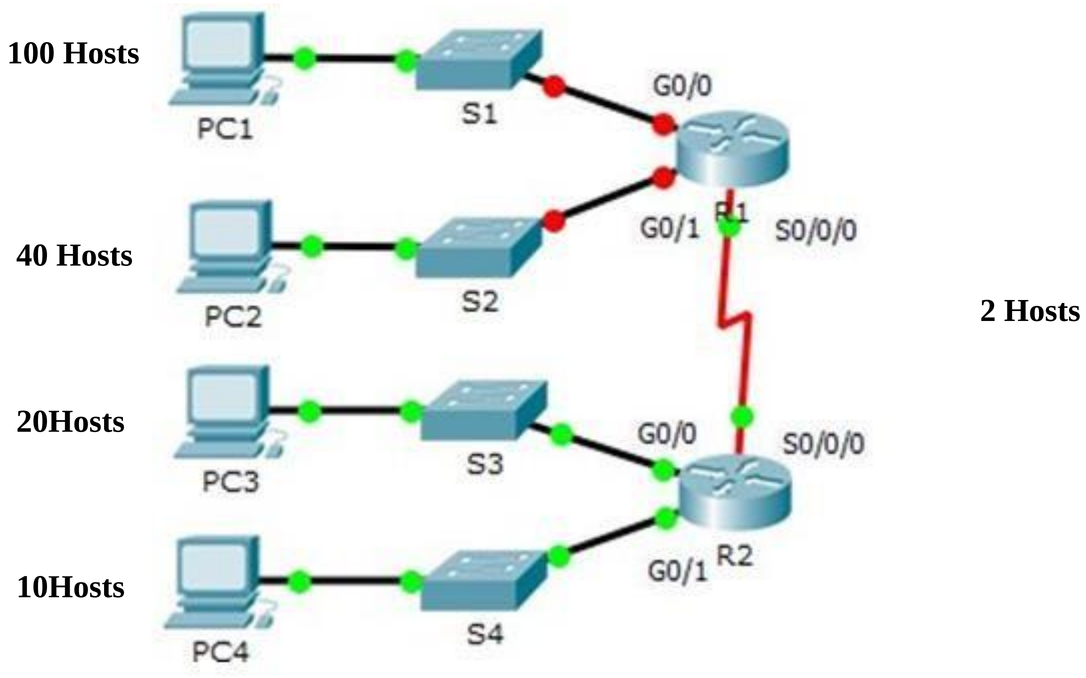


```
Packet Tracer PC Command Line 1.0
C:\>ping 172.20.0.1

Pinging 172.20.0.1 with 32 bytes of data:

Reply from 172.20.0.1: bytes=32 time=62ms TTL=253
Reply from 172.20.0.1: bytes=32 time=12ms TTL=253
Reply from 172.20.0.1: bytes=32 time=2ms TTL=253
Reply from 172.20.0.1: bytes=32 time=11ms TTL=253

Ping statistics for 172.20.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 62ms, Average = 21ms

C:\>
```

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	E
●	--	PC0	Server0	ICMP	▨	0.000	N	0	

2. Subnetting Scenario - VLSM

Topology



Objectives

1. Design an IP Addressing Scheme(VLSM)
2. Assign IP Addresses to Network Devices and Verify Connectivity

Scenario

In this activity, you are given the network address of 195.152.92.0/24 to subnet and provide the IP addressing for the network shown in the topology. Each LAN in the network requires enough space for, atleast, 25 addresses for end devices, the switch and the router. The connection between R1 to R2 will require an IP address for each end of thelink.

Design an IP Addressing Scheme

Step 1: Subnet the 195.152.92.0/24 network into the appropriate number of subnets.

a) Based on the topology, how many subnets areneeded?

b) How many bits must be borrowed to support the number of subnets in the topologytable?

c) How many subnets does thiscreate?

d) How many usable hosts does this create persubnet?

Note: If your answer is less than the 100 hosts required, then you borrowed too many bits.

a) Calculate the binary value for the first fivesubnets.

Net 0: 195. 152. 92 .0 0 0 0 0 0 0 0

Net 1: 195. 152. 92. 1 0 0 0 0 0 0 0

Net 2: 195. 152. 92. 1 1 0 0 0 0 0 0

Net 3: 195. 152. 92. 1 1 1 0 0 0 0 0

Net 4: 195. 152. 92. 1 1 1 1 0 0 0 0

b) Calculate the binary and decimal value of the new subnetmask.

Subnet 0Mask:**255.255.255.128** Subnet 3 Mask: **255.255.255.240**

Subnet 1Mask:**255.255.255.192** subnet 4 Mask: **255.255.255.252**

Subnet 2 Mask:**255.255.255.224**

c) Fill in the Subnet Table, listing the decimal value of all available subnets, the first and lastusable host address, and the broadcast address. Repeat until all addresses arelisted.

Note: You may not need to use all rows.

Subnet Table

Subnet Number	Subnet Address	First Usable Host Address	Last Usable Host Address	Broadcast Address
0	195.152.92.0	195.152.92.1	195.152.92.126	195.152.92.127
1	195.152.92.128	195.152.92.129	195.152.92.190	195.152.92.191
2	195.152.92.192	195.152.92.193	195.152.92.221	195.152.92.223
3	195.152.92.224	195.152.92.225	195.152.92.238	195.152.92.239
4	195.152.92.240	195.152.92.241	195.152.92.242	195.152.92.243

Step 02: Assign the subnets to the network shown in the topology.

a. Assign Subnet 0 to the LAN connected to the GigabitEthernet 0/0 interface ofR1.

b. Assign Subnet 1 to the LAN connected to the GigabitEthernet 0/1 interface ofR1.

c. Assign Subnet 2 to the LAN connected to the GigabitEthernet 0/0 interface ofR2

d. Assign Subnet 3 to the LAN connected to the GigabitEthernet 0/1 interface ofR2

e. Assign Subnet 4 to the WAN link between R1 toR2

Step 3: Document the addressing scheme.

Fill in the Addressing Table using the following guidelines:

a) Assign the first usable IP addresses to R1 for the two LAN links and the WANlink.

b) Assign the first usable IP addresses to R2 for the LANs links. Assign the last usable IP address for the WANlink.

c) Assign the second usable IP addresses to thehosts.

d) Assign the last usable IP addresses to thehosts.

Addressing Table

Device	Interface	IP Address	Subnet Mask	Default Gateway
R1	G0/0	195.152.92.1	255.255.255.128	N/A
	G0/1	195.152.92.129	255.255.255.192	N/A
	S0/0/0	195.152.92.241	255.255.255.252	N/A
R2	G0/0	195.152.92.193	255.255.255.224	N/A
	G0/1	195.152.92.225	255.255.255.240	N/A
	S0/0/0	195.152.92.242	255.255.255.252	N/A
PC1	NIC	195.152.92.2	255.255.255.128	195.152.92.1
PC2	NIC	195.152.92.130	255.255.255.192	195.152.92.129
PC3	NIC	195.152.92.194	255.255.255.224	195.152.92.193
PC4	NIC	195.152.92.226	255.255.255.240	195.152.92.225

Assign IP Addresses to Network Devices and Verify Connectivity

Most of the IP addressing is already configured on this network. Implement the following steps to complete the addressing configuration.

Step 1: Configure IP addressing on R1 LAN interfaces.

Step 2: Configure IP addressing on S3, including the default gateway

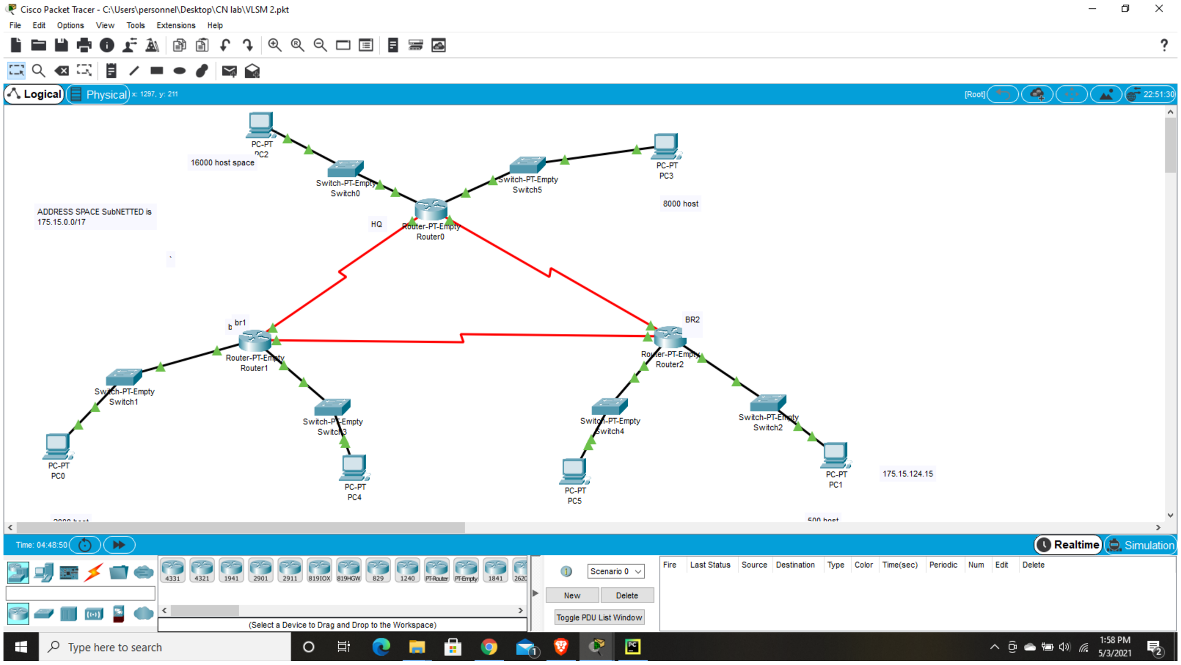
Step 3: Configure IP addressing on PC4, including the default gateway.

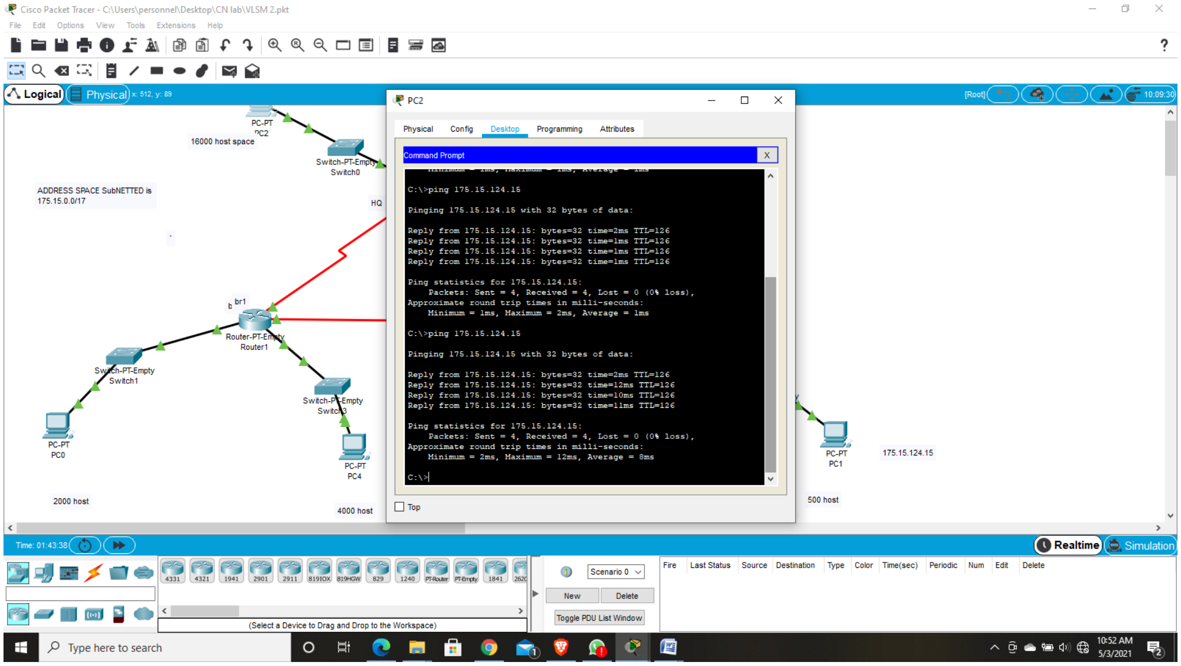
Step 4: Verify connectivity.

Verify connectivity from LAN to every other LAN. However, you should be able to ping every IP address listed in the Addressing Table.

Note:

1. Take the screen shot of the results which is verified in Simulation Mode /Real mode and write the inference.
2. After the completion of the experiment, answer all the viva question given below in the labitself.





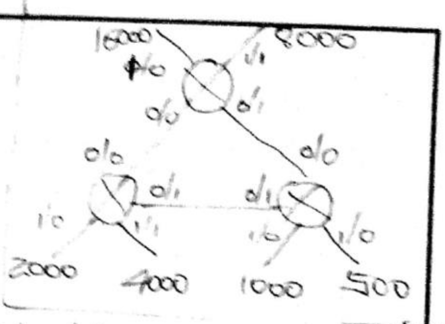
```
C:\>ping 175.15.124.15

Pinging 175.15.124.15 with 32 bytes of data:

Reply from 175.15.124.15: bytes=32 time=2ms TTL=126
Reply from 175.15.124.15: bytes=32 time=1ms TTL=126
Reply from 175.15.124.15: bytes=32 time=1ms TTL=126
Reply from 175.15.124.15: bytes=32 time=1ms TTL=126

Ping statistics for 175.15.124.15:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\>ping 175.15.124.15

Pinging 175.15.124.15 with 32 bytes of data:

Reply from 175.15.124.15: bytes=32 time=2ms TTL=126
Reply from 175.15.124.15: bytes=32 time=12ms TTL=126
Reply from 175.15.124.15: bytes=32 time=10ms TTL=126
Reply from 175.15.124.15: bytes=32 time=11ms TTL=126

Ping statistics for 175.15.124.15:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 12ms, Average = 8ms

C:\>
```

175.15.125.0/17

i.e. 175.15.0|1111101.00000000

SM 255.255.1|0000000.00000000

N/w | Host



$\Rightarrow$ Maj. N/w id = 175.15.0.0/17

with 15 host bits $\rightarrow 2^{15} = 32760$ hosts

required hosts = 31500

-"- n/w = 6

i. $2^h - 2 \geq 16000$

$\Rightarrow h = 14$

$\frac{2^{14}}{256} = 64$

ii. $2^h - 2 \geq 8000$

$\Rightarrow h = 13$

$\frac{2^{13}}{256} = 32 \left\langle \begin{matrix} 32+64 \\ =96 \end{matrix} \right\rangle$

iii. $2^h - 2 \geq 4000$

$\Rightarrow h = 12$

$\frac{2^{12}}{256} = 16 \left\langle \begin{matrix} 16+96 \\ =112 \end{matrix} \right\rangle$

IV. $2^h - 2 \geq 2000$

$\Rightarrow h = 11$

$\frac{2^{11}}{256} = 8 \left\langle \begin{matrix} 8+112 \\ =120 \end{matrix} \right\rangle$

V. $2^h - 2 \geq 1000$

$\Rightarrow h = 10$

$\frac{2^{10}}{256} = 4 \left\langle \begin{matrix} 4+120 \\ =124 \end{matrix} \right\rangle$

VI. $2^h - 2 \geq 500$

$\Rightarrow h = 9$

$\frac{2^{9}}{256} = 2 \left\langle \begin{matrix} 2+124 \\ =126 \end{matrix} \right\rangle$

i. 175.15.0.0 to 175.15.63.255/18
ii. 175.15.64.0 to 175.15.95.255/19
iii. 175.15.96.0 to 175.15.111.255/20
IV. 175.15.112.0 to 175.15.119.255/21
V. 175.15.120.0 to 175.15.123.255/22
VI. 175.15.124.0 to 175.15.125.255/23
175.15.126.0 to 175.15.

SM : 255.255

i. .11000000.0 → /18 → 192
ii. .11100000.0 → /19 → 224
iii. .11110000.0 → /20 → 240
IV. .11111000.0 → /21 → 248
V. .11111100.0 → /22 → 252
VI. 11111110.0 → /23 → 254

I. 175.15.126.0 to 175.15.126.

$2^h - 2 \geq 2 \Rightarrow h = 2$

Viva

1. $2^5 - 2 = 30$

2. Subnetting is dividing of larger networks into smaller networks.

3. Fixed Length Subnet Mask : the number of mask in the different LAN will be same.

4. 222.1.0.20/~~27~~ 26

5. 197.1.2.67 ~~27~~ N/w & 197.1.2.67 → host. 126

6. 128 & 192

$2^n - 2 > 48$
$2^6 - 2 > 48$
$62 > 48$
$H = 6$, $N = 2$
i.e. 255.255.255.192

$2^n > 2$
$n = 1$
i.e. 255.255.255.128

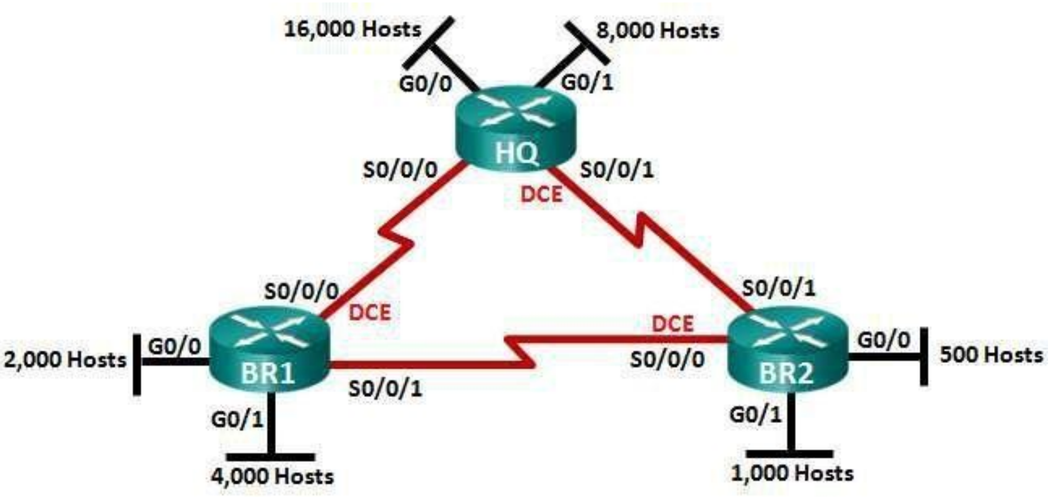
7. a. $2^5 = 32$: 255.255.255.248
   b. $2^3 = 8$
   c. ~~192.163~~ 205.11.2.11

8. 211.1.1.0
   255.255.255.0
   $2^8 - 2 = 254$.

Mev
03.4.2021.

2.1. Designing and Implementing a VLSM Addressing Scheme

Topology



Objectives

1. **Examine Network Requirements**
2. **Design the VLSM Address Scheme**
3. **Cable and Configure the IPv4 Network**

Background / Scenario

Variable Length Subnet Mask (VLSM) was designed to avoid wasting IP addresses. With VLSM, a network is subnetted and then re-subnetted. This process can be repeated multiple times to create subnets of various sizes based on the number of hosts required in each subnet. Effective use of VLSM requires address planning.

In this lab, use the 175.15.125.0/17 network address to develop an address scheme for the network displayed in the topology diagram. VLSM is used to meet the IPv4 addressing requirements. After you have designed the VLSM address scheme, you will configure the interfaces on the routers with the appropriate IP address information.

Part 1: Examine NetworkRequirements

In Part 1, you will examine the network requirements to develop a VLSM address scheme for the network displayed in the topology diagram using the 175.15.125.0/17 network address.

Step 1: Determine how many host addresses and subnets are available.

How many host addresses are available in a /17 network? 32760

What is the total number of host addresses needed in the topology diagram? 31500

How many subnets are needed in the network topology? 6

Step 2: Determine the largest subnet.

What is the subnet description (e.g. BR1 G0/1 LAN or BR1-HQ WAN link)? HQ G0/0

How many IP addresses are required in the largest subnet? 16000

What subnet mask can support that many host addresses? 255.255.63.0 (/18)

How many total host addresses can that subnet mask support? 16384

Part 2: Design the VLSM Address Scheme

Step 1: Calculate the subnet information.

Use the information that you obtained in Part 1 to fill in the following table.

Subnet Description	Number of Hosts Needed	Network Address /CIDR	First Host Address	Broadcast Address
HQ G0/0	16,000	175.15.0.0	175.15.0.1	175.15.63.255
HQ G0/1	8,000	175.15.64.0	175.15.64.1	175.15.95.255
BR1 G0/1	4,000	175.15.96.0	175.15.96.1	175.15.111.255
BR1 G0/0	2,000	175.15.112.0	175.15.112.1	175.15.119.255
BR2 G0/1	1,000	175.15.120.0	175.15.120.1	175.15.123.255
BR2 G0/0	500	175.15.124.0	175.15.124.1	175.15.125.255
HQ S0/0/0 – BR1 S0/0/0	2	195.152.92.0	195.152.92.1	195.152.92.255
HQ S0/0/1 – BR2 S0/0/1	2	195.152.93.0	195.152.93.1	195.152.93.255
BR1 S0/0/1 – BR2 S0/0/0	2	195.152.94.0	195.152.94.1	195.152.94.255

Step 2: Complete the device interface address table.

Assign the first host address in the subnet to the Ethernet interfaces. HQ should be given the first host address on the Serial links to BR1 and BR2. BR1 should be given the first host address for the serial link to BR2.

Device	Interface	IP Address	Subnet Mask	Device Interface
HQ	G0/0	175.15.0.1	255.255.192.0	16,000 Host LAN
	G0/1	175.15.64.0	255.255.224	8,000 Host LAN
	S0/0/0	195.152.92.0	255.255.255.0	BR1 S0/0/0
	S0/0/1	195.152.93.0	255.255.255.0	BR2 S0/0/1
BR1	G0/1	175.15.96.0	255.255.240.0	2,000 Host LAN
	G0/0	175.15.112.0	255.255.248	4,000 Host LAN
	S0/0/0	195.152.92.1	255.255.255.0	HQ S0/0/0
	S0/0/1	195.152.94.0	255.255.255.0	BR2 S0/0/0

BR2	G0/0	175.15.124.0	255.255.254	500 Host LAN
	G0/1	175.15.120.0	255.255.252	1,000 Host LAN
	S0/0/0	195.152.94.1	255.255.255.0	BR1 S0/0/1
	S0/0/1	195.152.93.1	255.255.255.0	HQ S0/0/1

Step 1: Cable the network as shown in the topology.

Step 2: Configure basic settings on each router.

a. Assign the device name to the router.

Step 3: Configure the interfaces on each router.

a. Assign an IP address and subnet mask to each interface using the table that you completed in Part 2.

b. Configure an interface description for each interface.

c. Set the clocking rate on all DCE serial interfaces to 128000.

```
HQ(config-if)# clock rate 128000
```

d. Activate the interfaces.

Step 4: Save the configuration on all devices. Step 5:

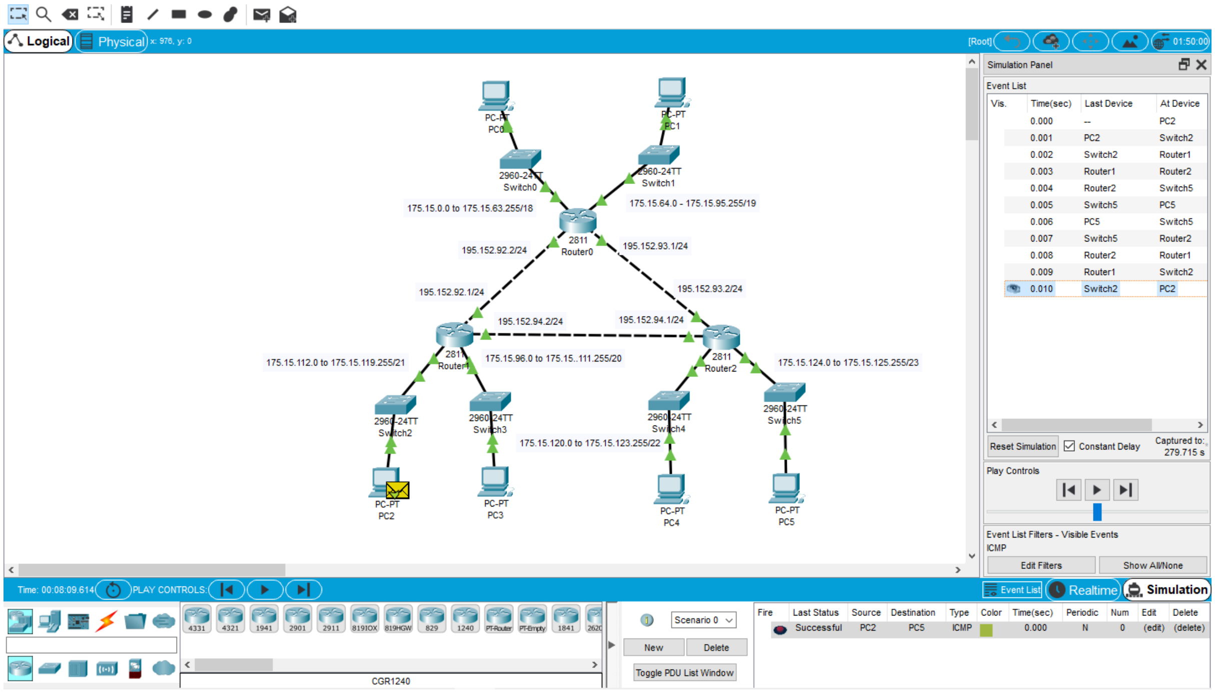
Test Connectivity.

a. From HQ, ping BR1's S0/0/0 interface address.

b. From HQ, ping BR2's S0/0/1 interface address.

c. From BR1, ping BR2's S0/0/0 interface address.

d. Troubleshoot connectivity issues if pings were not successful..



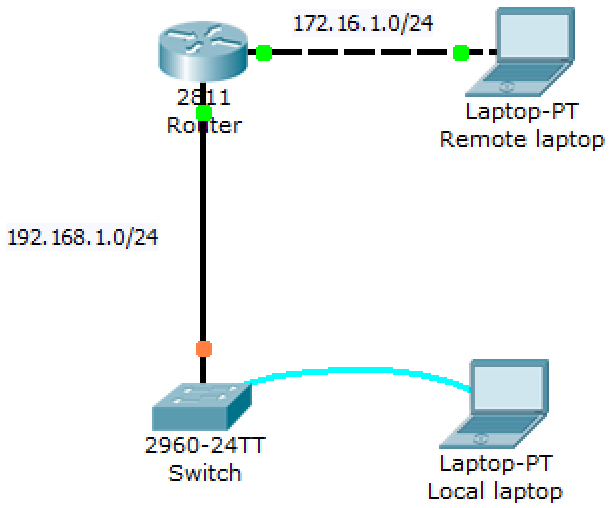
Event List

Vis.	Time(sec)	Last Device	At Device
	0.000	--	PC2
	0.001	PC2	Switch2
	0.002	Switch2	Router1
	0.003	Router1	Router2
	0.004	Router2	Switch5
	0.005	Switch5	PC5
	0.006	PC5	Switch5
	0.007	Switch5	Router2
	0.008	Router2	Router1
	0.009	Router1	Switch2
👁	0.010	Switch2	PC2

Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num	Edit	Delete
●	Successful	PC2	PC5	ICMP	🟩	0.000	N	0	(edit)	(delete)

3. Basic Router Configuration:

Network diagram



Lab instructions

This lab will test your ability to configure basic settings such as hostname, motd banner, encrypted passwords, and terminal options on a Cisco Catalyst 2960 switch simulated in Packet Tracer 7 & above version.

1. Use the local laptop connect to the switch console and configure the laptop with the right parameters for console access to the Cisco 2960 Catalyst switch.Configure Switch hostname as LOCAL-Router
2. Configure the message of the day as "Unauthorized access is forbidden"
3. Configure the password for privileged mode access as "cisco". The password must be md5 encrypted
4. Configure password encryption on the switch using the global configuration command
5. Configure CONSOLE access with the following settings :

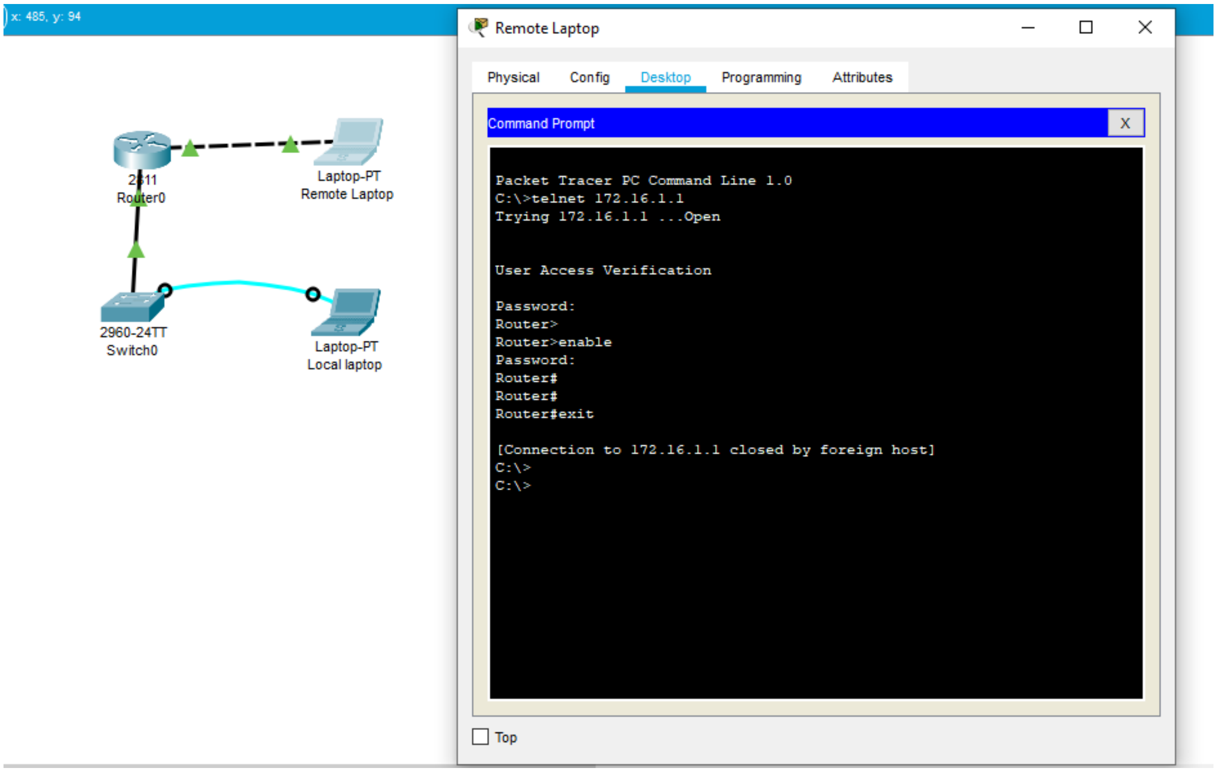
   - Login enabled

   - Password : ciscoconsole

- History size : 15 commands
- Timeout : 6'45"
- Synchronous logging

6. Configure TELNET access with the following settings :
   - Login enabled
   - Password : ciscotelnet
   - History size : 15 commands
   - Timeout : 8'20"
   - Synchronous logging

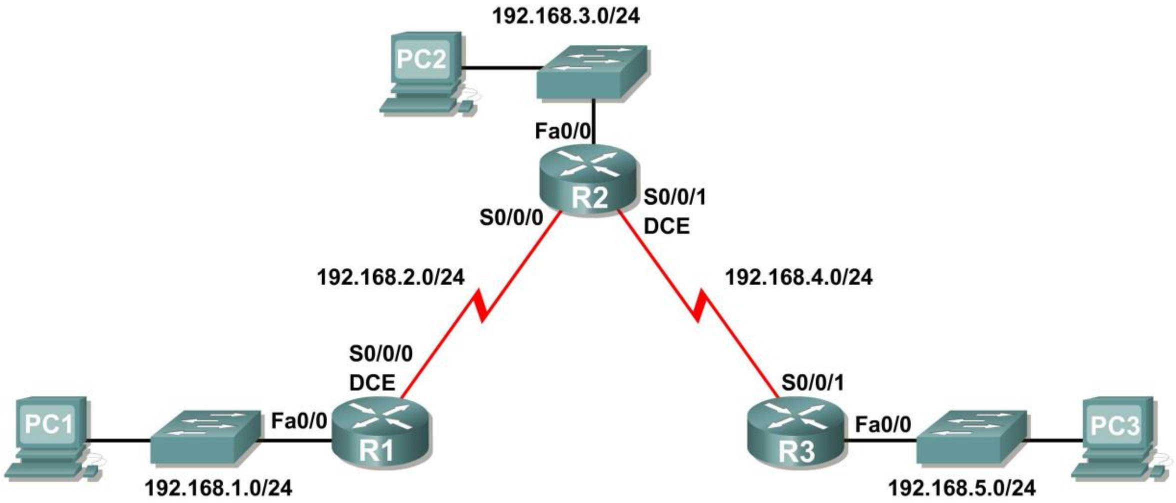
7. Configure the IP address of the switch as 192.168.1.2/24 and its default gateway IP (192.168.1.1).
8. Test telnet connectivity from the Remote Laptop using the telnet client.



```
Packet Tracer PC Command Line 1.0
C:\>telnet 172.16.1.1
Trying 172.16.1.1 ...Open


User Access Verification

Password:
Router>
Router>enable
Password:
Router#
Router#
Router#exit

[Connection to 172.16.1.1 closed by foreign host]
C:\>
C:\>
```

4. RIP Configuration:

Topology Diagram



Learning Objectives

Upon completion of this lab, you will be able to:

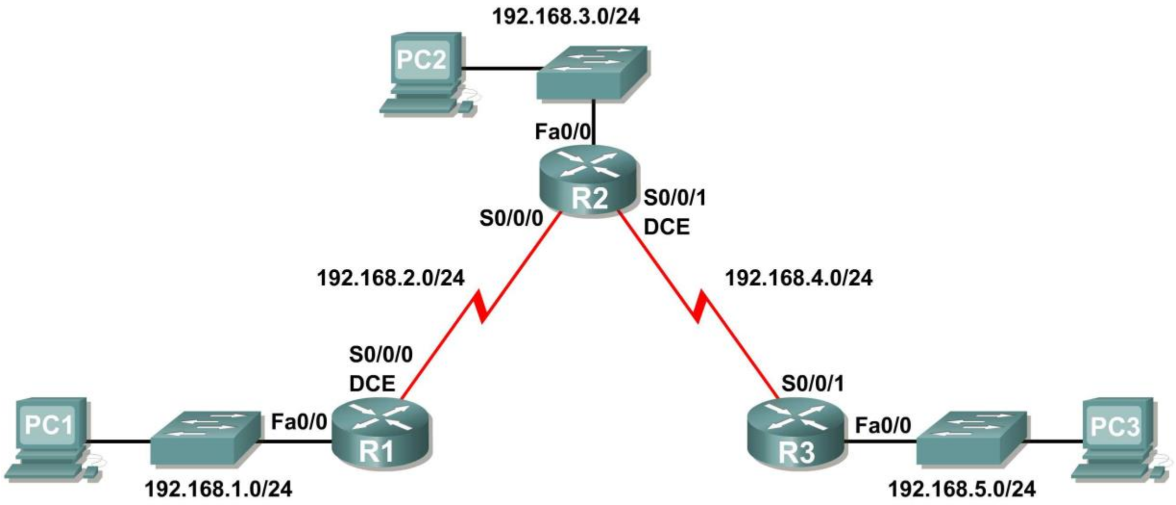
- Cable a network according to the Topology Diagram.
- Erase the startup configuration and reload a router to the default state.
- Perform basic configuration tasks on a router.
- Configure and activate interfaces.
- Configure RIP routing on all routers.
- Verify RIP routing using **`show`** and **`debug`** commands.
- Reconfigure the network to make it contiguous.
- Observe automatic summarization at boundary router.
- Gather information about RIP processing using the **`debug ip rip`** command.
- Configure a static default route.
- Propagate default routes to RIP neighbors.
- Document the RIP configuration.

Scenarios

- Scenario A: Running RIPv1 on Classful Networks
- Scenario B: Running RIPv1 with Subnets and Between Classful Networks
- Scenario C: Running RIPv1 on a Stub Network.

4.1. Scenario A: Running RIPv1 on Classful Networks

Topology Diagram



Addressing Table

Device	Interface	IP Address	Subnet Mask	Default Gateway
R1	**Fa0/0**	192.168.1.1	255.255.255.0	N/A
	S0/0/0	192.168.2.1	255.255.255.0	N/A
R2	**Fa0/0**	192.168.3.1	255.255.255.0	N/A
	S0/0/0	192.168.2.2	255.255.255.0	N/A
	S0/0/1	192.168.4.2	255.255.255.0	N/A
R3	**Fa0/0**	192.168.5.1	255.255.255.0	N/A
	S0/0/1	192.168.4.1	255.255.255.0	N/A
PC1	**NIC**	192.168.1.10	255.255.255.0	192.168.1.1
PC2	**NIC**	192.168.3.10	255.255.255.0	192.168.3.1
PC3	**NIC**	192.168.5.10	255.255.255.0	192.168.5.1

Task 1: Prepare the Network.

Step 1: Cable a network that is similar to the one in the Topology Diagram.

You can use any current router in your lab as long as it has the required interfaces shown in the topology.
Note: If you use 1700, 2500, or 2600 routers, the router outputs and interface descriptions will appear different.

Step 2: Clear any existing configurations on the routers.

Task 2: Perform Basic Router Configurations.

Perform basic configuration of the R1, R2, and R3 routers according to the following guidelines:

1. Configure the router hostname.
2. Disable DNS lookup.
3. Configure an EXEC mode password.
4. Configure a message-of-the-day banner.
5. Configure a password for console connections.
6. Configure a password for VTY connections.

Task 3: Configure and Activate Serial and Ethernet Addresses.

Step 1: Configure interfaces on R1, R2, and R3.

Configure the interfaces on the R1, R2, and R3 routers with the IP addresses from the table under the Topology Diagram.

Step 2: Verify IP addressing and interfaces.

Use the **show ip interface brief** command to verify that the IP addressing is correct and that the interfaces are active.

When you have finished, be sure to save the running configuration to the NVRAM of the router.

Step 3: Configure Ethernet interfaces of PC1, PC2, and PC3.

Configure the Ethernet interfaces of PC1, PC2, and PC3 with the IP addresses and default gateways from the table under the Topology Diagram.

Step 4: Test the PC configuration by pinging the default gateway from the PC.

Task 4: Configure RIP.

Step 1: Enable dynamic routing.

To enable a dynamic routing protocol, enter global configuration mode and use the **router** command.

Enter **router ?** at the global configuration prompt to a see a list of available routing protocols on your router.

To enable RIP, enter the command **router rip** in global configuration mode.

```
R1(config)#router rip
R1(config-router)#
```

Step 2: Enter classful network addresses.

Once you are in routing configuration mode, enter the classful network address for each directly connected network, using the **network** command.

```
R1(config-router)#network 192.168.1.0
R1(config-router)#network 192.168.2.0
```

```
R1(config-router)#
```

The **network** command:

- Enables RIP on all interfaces that belong to this network. These interfaces will now both send and receive RIP updates.
- Advertises this network in RIP routing updates sent to other routers every 30 seconds.

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

```
R1(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R1#copy run start
```

Step 3: Configure RIP on the R2 router using the `router rip` and `network` commands.

```
R2(config)#router rip
R2(config-router)#network 192.168.2.0
R2(config-router)#network 192.168.3.0
R2(config-router)#network 192.168.4.0
R2(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R2#copy run start
```

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

Step 4: Configure RIP on the R3 router using the `router rip` and `network` commands.

```
R3(config)#router rip
R3(config-router)#network 192.168.4.0
R3(config-router)#network 192.168.5.0
R3(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R3# copy run start
```

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

Task 5: Verify RIP Routing.

Step 1: Use the `show ip route` command to verify that each router has all of the networks in the topology entered in the routing table.

Routes learned through RIP are coded with an **R** in the routing table. If the tables are not converged as shown here, troubleshoot your configuration. Did you verify that the configured interfaces are active? Did you configure RIP correctly? Return to Task 3 and Task 4 to review the steps necessary to achieve convergence.

```
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter
area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/0/0
R    192.168.3.0/24 [120/1] via 192.168.2.2, 00:00:04, Serial0/0/0
R    192.168.4.0/24 [120/1] via 192.168.2.2, 00:00:04, Serial0/0/0
R    192.168.5.0/24 [120/2] via 192.168.2.2, 00:00:04, Serial0/0/0
R1#


R2#show ip route

<Output omitted>

R    192.168.1.0/24 [120/1] via 192.168.2.1, 00:00:22, Serial0/0/0
C    192.168.2.0/24 is directly connected, Serial0/0/0
C    192.168.3.0/24 is directly connected, FastEthernet0/0
C    192.168.4.0/24 is directly connected, Serial0/0/1
R    192.168.5.0/24 [120/1] via 192.168.4.1, 00:00:23, Serial0/0/1
R2#

R3#show ip route

<Output omitted>

R    192.168.1.0/24 [120/2] via 192.168.4.2, 00:00:18, Serial0/0/1
R    192.168.2.0/24 [120/1] via 192.168.4.2, 00:00:18, Serial0/0/1
R    192.168.3.0/24 [120/1] via 192.168.4.2, 00:00:18, Serial0/0/1
C    192.168.4.0/24 is directly connected, Serial0/0/1
C    192.168.5.0/24 is directly connected, FastEthernet0/0
R3#
```

Step 2: Use the `show ip protocols` command to view information about the routing processes.

The **`show ip protocols`** command can be used to view information about the routing processes that are occurring on the router. This output can be used to verify most RIP parameters to confirm that:

- RIP routing is configured
- The correct interfaces send and receive RIP updates
- The router advertises the correct networks
- RIP neighbors are sending updates

```
R1#show ip protocols
Routing Protocol is "rip"
Sending updates every 30 seconds, next due in 16 seconds
Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
Redistributing: rip
Default version control: send version 1, receive any version
  Interface             Send  Recv  Triggered RIP  Key-chain
  FastEthernet0/0       1     2 1
  Serial0/0/0           1     2 1
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
      192.168.1.0
      192.168.2.0
Passive Interface(s):
Routing Information Sources:
      Gateway         Distance      Last Update
      192.168.2.2          120
Distance: (default is 120)
R1#
```

R1 is indeed configured with RIP. R1 is sending and receiving RIP updates on FastEthernet0/0 and Serial0/0/0. R1 is advertising networks 192.168.1.0 and 192.168.2.0. R1 has one routing information source. R2 is sending R1 updates.

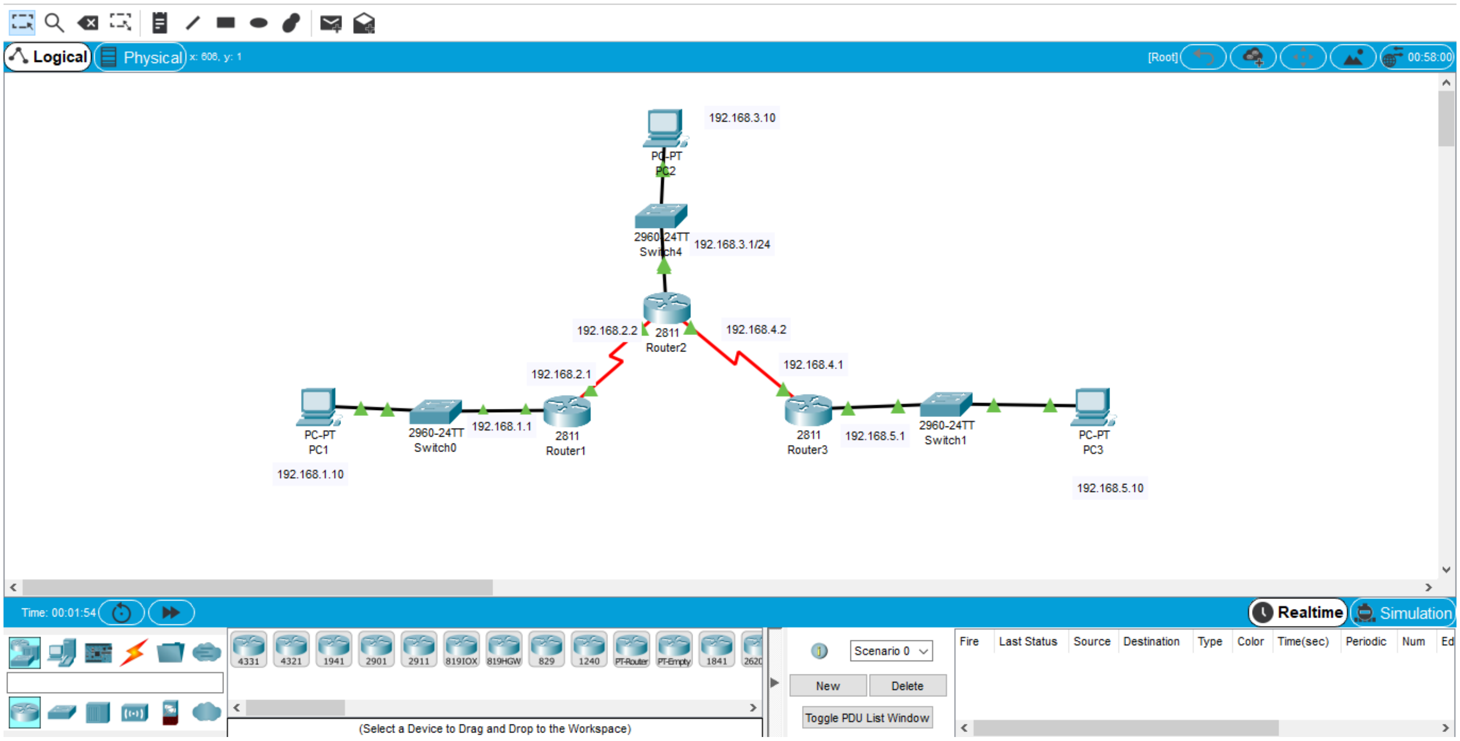
Step 3: Use the `debug ip rip` command to view the RIP messages being sent and received.

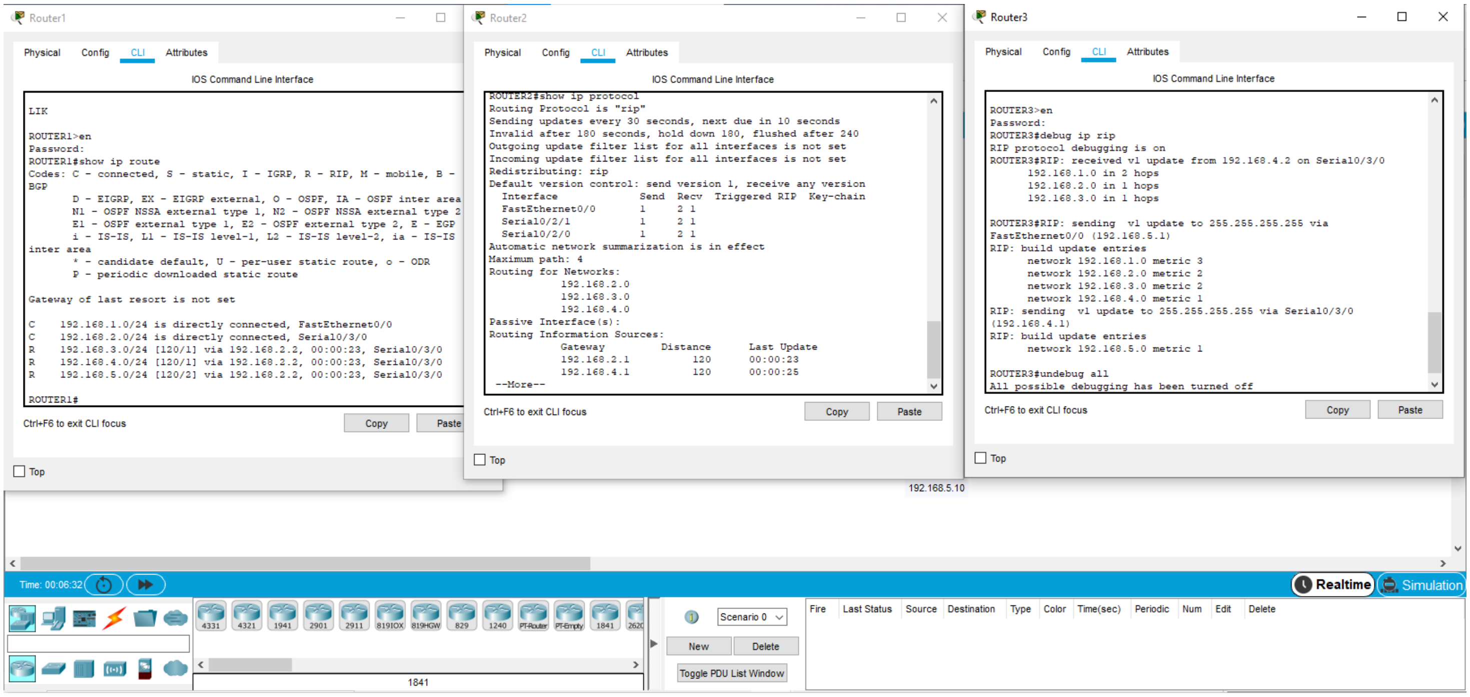
Rip updates are sent every 30 seconds so you may have to wait for debug information to be displayed.

```
R1#debug ip rip
R1#RIP: received v1 update from 192.168.2.2 on Serial0/0/0
      192.168.3.0 in 1 hops
      192.168.4.0 in 1 hops
      192.168.5.0 in 2 hops
RIP: sending  v1 update to 255.255.255.255 via FastEthernet0/0
(192.168.1.1)
RIP: build update entries
      network 192.168.2.0 metric 1
      network 192.168.3.0 metric 2
      network 192.168.4.0 metric 2
      network 192.168.5.0 metric 3
RIP: sending  v1 update to 255.255.255.255 via Serial0/0/0 (192.168.2.1)
RIP: build update entries
      network 192.168.1.0 metric 1
```

Step 4: Discontinue the debug output with the `undebug all` command.

```
R1#undebug all
All possible debugging has been turned off
```





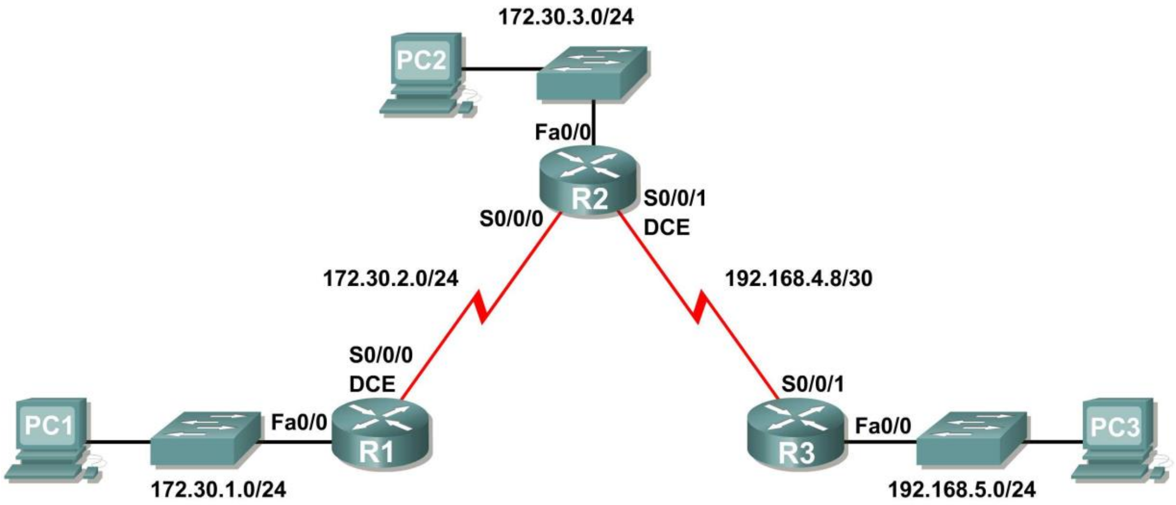
```
LIK

ROUTER1>en
Password:
ROUTER1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B -
BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS
inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/3/0
R    192.168.3.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.4.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.5.0/24 [120/2] via 192.168.2.2, 00:00:23, Serial0/3/0

ROUTER1#
```

```
ROUTER2#show ip protocol
Routing Protocol is "rip"
Sending updates every 30 seconds, next due in 10 seconds
Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
Redistributing: rip
Default version control: send version 1, receive any version
  Interface             Send  Recv  Triggered RIP  Key-chain
  FastEthernet0/0       1     2 1
  Serial0/2/1           1     2 1
  Serial0/2/0           1     2 1
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
           192.168.2.0
           192.168.3.0
           192.168.4.0
Passive Interface(s):
Routing Information Sources:
           Gateway         Distance      Last Update
           192.168.2.1          120      00:00:23
           192.168.4.1          120      00:00:25
 --More--
```

```
ROUTER3>en
Password:
ROUTER3#debug ip rip
RIP protocol debugging is on
ROUTER3#RIP: received v1 update from 192.168.4.2 on Serial0/3/0
      192.168.1.0 in 2 hops
      192.168.2.0 in 1 hops
      192.168.3.0 in 1 hops

ROUTER3#RIP: sending  v1 update to 255.255.255.255 via
FastEthernet0/0 (192.168.5.1)
RIP: build update entries
      network 192.168.1.0 metric 3
      network 192.168.2.0 metric 2
      network 192.168.3.0 metric 2
      network 192.168.4.0 metric 1
RIP: sending  v1 update to 255.255.255.255 via Serial0/3/0
(192.168.4.1)
RIP: build update entries
      network 192.168.5.0 metric 1

ROUTER3#undebug all
All possible debugging has been turned off
```

4.2. Scenario B: Running RIPv1 with Subnets between Classful Networks

Topology Diagram



Addressing Table

Device	Interface	IP Address	Subnet Mask	Default Gateway
R1	**Fa0/0**	172.30.1.1	255.255.255.0	N/A
	S0/0/0	172.30.2.1	255.255.255.0	N/A
R2	**Fa0/0**	172.30.3.1	255.255.255.0	N/A
	S0/0/0	172.30.2.2	255.255.255.0	N/A
	S0/0/1	192.168.4.9	255.255.255.252	N/A
R3	**Fa0/0**	192.168.5.1	255.255.255.0	N/A
	S0/0/1	192.168.4.10	255.255.255.252	N/A
PC1	**NIC**	172.30.1.10	255.255.255.0	172.30.1.1
PC2	**NIC**	172.30.3.10	255.255.255.0	172.30.3.1
PC3	**NIC**	192.168.5.10	255.255.255.0	192.168.5.1

Task 1: Make Changes between Scenario A and Scenario B

Step 1: Change the IP addressing on the interfaces as shown in the Topology Diagram and the Addressing Table.

Sometimes when changing the IP address on a serial interface, you may need to reset that interface by using the **`shutdown`** command, waiting for the `LINK-5-CHANGED` message, and then using the **`no shutdown`** command. This process will force the IOS to starting using the new IP address.

```
R1(config)#int s0/0/0
R1(config-if)#ip add 172.30.2.1 255.255.255.0
R1(config-if)#shutdown

%LINK-5-CHANGED: Interface Serial0/0/0, changed state to administratively
down
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed
state to down
R1(config-if)#no shutdown

%LINK-5-CHANGED: Interface Serial0/0/0, changed state to up
R1(config-if)#
%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed
state to up
```

Step 2: Verify that routers are active.

After reconfiguring all the interfaces on all three routers, verify that all necessary interfaces are active with the **`show ip interface brief`** command.

Step 3: Remove the RIP configurations from each router.

Although you can remove the old **`network`** commands with the **`no`** version of the command, it is more efficient to simply remove RIP and start over. Remove the RIP configurations from each router with the **`no router rip`** global configuration command. This will remove all the RIP configuration commands including the **`network`** commands.

```
R1(config)#no router rip

R2(config)#no router rip

R3(config)#no router rip
```

Task 2: Configure RIP

Step 1: Configure RIP routing on R1 as shown below.

```
R1(config)#router rip
R1(config-router)#network 172.30.0.0
```

Notice that only a single network statement is needed for R1. This statement includes both interfaces on different subnets of the 172.30.0.0 major network.

Step 2: Configure R1 to stop sending updates out the FastEthernet0/0 interface.

Sending updates out this interface wastes the bandwidth and processing resources of all devices on the LAN. In addition, advertising updates on a broadcast network is a security risk. RIP updates can be intercepted with packet sniffing software. Routing updates can be modified and sent back to the router, corrupting the router table with false metrics that misdirects traffic.

The **`passive-interface fastethernet 0/0`** command is used to disable sending RIPv1 updates out that interface. When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

```
R1(config-router)#passive-interface fastethernet 0/0
R1(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R1#copy run start
```

Step 3: Configure RIP routing on R2 as shown below.

```
R2(config)#router rip
R2(config-router)#network 172.30.0.0
R2(config-router)#network 192.168.4.0
R2(config-router)#passive-interface fastethernet 0/0
R2(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R2#copy run start
```

Again notice that only a single network statement is needed for the two subnets of 172.30.0.0. This statement includes both interfaces, on different subnets, of the 172.30.0.0 major network. The network for the WAN link between R2 and R3 is also configured. When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

Step 4: Configure RIP routing on R3 as shown below.

```
R3(config)#router rip
R3(config-router)#network 192.168.4.0
R3(config-router)#network 192.168.5.0
R3(config-router)#passive-interface fastethernet 0/0
R3(config-router)#end
%SYS-5-CONFIG_I: Configured from console by console
R3#copy run start
```

When you are finished with the RIP configuration, return to privileged EXEC mode and save the current configuration to NVRAM.

Task 3: Verify RIP Routing

Step 1: Use the `show ip route` command to verify that each router has all of the networks in the topology in the routing table.

```
R1#show ip route

<Output omitted>

     172.30.0.0/24 is subnetted, 3 subnets
C       172.30.1.0 is directly connected, FastEthernet0/0
C       172.30.2.0 is directly connected, Serial0/0/0
R       172.30.3.0 [120/1] via 172.30.2.2, 00:00:22, Serial0/0/0
R    192.168.4.0/24 [120/1] via 172.30.2.2, 00:00:22, Serial0/0/0
R    192.168.5.0/24 [120/2] via 172.30.2.2, 00:00:22, Serial0/0/0
R1#
```

```
R2#show ip route

<Output omitted>

     172.30.0.0/24 is subnetted, 3 subnets
R       172.30.1.0 [120/1] via 172.30.2.1, 00:00:04, Serial0/0/0
C       172.30.2.0 is directly connected, Serial0/0/0
C       172.30.3.0 is directly connected, FastEthernet0/0
     192.168.4.0/30 is subnetted, 1 subnets
C       192.168.4.8 is directly connected, Serial0/0/1
R    192.168.5.0/24 [120/1] via 192.168.4.10, 00:00:19, Serial0/0/1
R2#
```

```
R3#show ip route

R    172.30.0.0/16 [120/1] via 192.168.4.9, 00:00:22, Serial0/0/1
     192.168.4.0/30 is subnetted, 1 subnets
C       192.168.4.8 is directly connected, Serial0/0/1
C    192.168.5.0/24 is directly connected, FastEthernet0/0
```

Step 2: Verify that all necessary interfaces are active.

If one or more routing tables does not have a converged routing table, first make sure that all necessary interfaces are active with **`show ip interface brief`**.
Then use **`show ip protocols`** to verify the RIP configuration. Notice in the output from this command that the FastEthernet0/0 interface is no longer listed under **Interface** but is now listed under a new section of the output: **Passive Interface(s)**.

```
R1#show ip protocols
Routing Protocol is "rip"
  Sending updates every 30 seconds, next due in 20 seconds
  Invalid after 180 seconds, hold down 180, flushed after 240
  Outgoing update filter list for all interfaces is not set
  Incoming update filter list for all interfaces is not set
  Redistributing: rip
  Default version control: send version 2, receive version 2
    Interface             Send  Recv  Triggered RIP  Key-chain
    Serial0/1/0           2     2
  Automatic network summarization is in effect
  Maximum path: 4
  Routing for Networks:
    172.30.0.0
    209.165.200.0
Passive Interface(s):
      FastEthernet0/0
  Routing Information Sources:
    Gateway         Distance      Last Update
    209.165.200.229      120      00:00:15
  Distance: (default is 120)
```

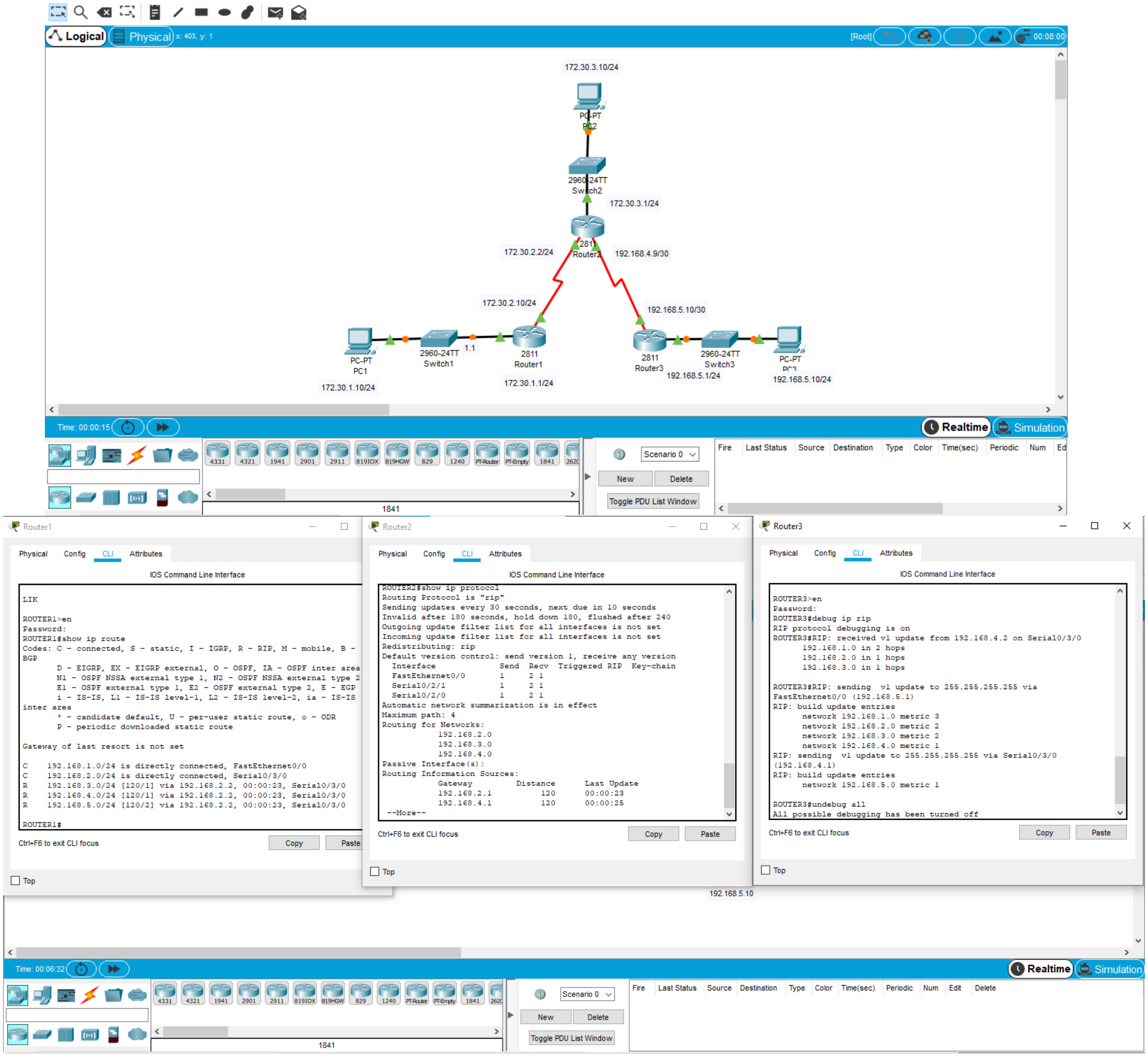
Step 3: View the RIP messages being sent and received.

To view the RIP messages being sent and received use the **`debug ip rip`** command. Notice that RIP updates are not sent out of the fa0/0 interface because of the **`passive-interface fastethernet 0/0`** command.

```
R1#debug ip rip
R1#RIP: sending  v1 update to 255.255.255.255 via Serial0/0/0
(172.30.2.1)
RIP: build update entries
      network 172.30.1.0 metric 1
RIP: received v1 update from 172.30.2.2 on Serial0/0/0
      172.30.3.0 in 1 hops
```

Step 4: Discontinue the debug output with the `undebug all` command.

```
R1#undebug all
All possible debugging has been turned off
```



Router1:

```
LIK

ROUTER1>en
Password:
ROUTER1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B -
BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS
inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/3/0
R    192.168.3.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.4.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.5.0/24 [120/2] via 192.168.2.2, 00:00:23, Serial0/3/0

ROUTER1#
```

Router2:

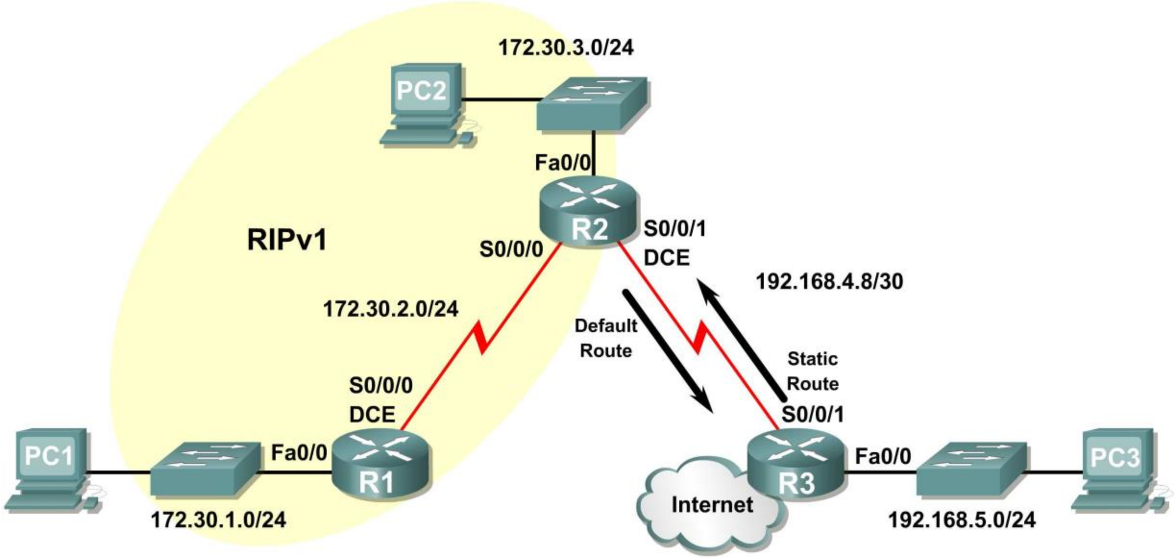
```
ROUTER2#show ip protocol
Routing Protocol is "rip"
Sending updates every 30 seconds, next due in 10 seconds
Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
Redistributing: rip
Default version control: send version 1, receive any version
  Interface             Send  Recv  Triggered RIP  Key-chain
  FastEthernet0/0       1     2 1
  Serial0/2/1           1     2 1
  Serial0/2/0           1     2 1
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
           192.168.2.0
           192.168.3.0
           192.168.4.0
Passive Interface(s):
Routing Information Sources:
           Gateway         Distance      Last Update
           192.168.2.1          120      00:00:23
           192.168.4.1          120      00:00:25
 --More--
```

Router3:

```
ROUTER3>en
Password:
ROUTER3#debug ip rip
RIP protocol debugging is on
ROUTER3#RIP: received v1 update from 192.168.4.2 on Serial0/3/0
      192.168.1.0 in 2 hops
      192.168.2.0 in 1 hops
      192.168.3.0 in 1 hops

ROUTER3#RIP: sending  v1 update to 255.255.255.255 via
FastEthernet0/0 (192.168.5.1)
RIP: build update entries
      network 192.168.1.0 metric 3
      network 192.168.2.0 metric 2
      network 192.168.3.0 metric 2
      network 192.168.4.0 metric 1
RIP: sending  v1 update to 255.255.255.255 via Serial0/3/0
(192.168.4.1)
RIP: build update entries
      network 192.168.5.0 metric 1

ROUTER3#undebug all
All possible debugging has been turned off
```

4.3. Scenario C: Running RIPv1 on a Stub Network

Topology Diagram



Background

In this scenario we will modify Scenario B to only run RIP between R1 and R2. Scenario C is a typical configuration for most companies connecting a stub network to a central headquarters router or an ISP. Typically, a company runs a dynamic routing protocol (RIPv1 in our case) within the local network but finds it unnecessary to run a dynamic routing protocol between the company's gateway router and the ISP. For example, colleges with multiple campuses often run a dynamic routing protocol between campuses but use default routing to the ISP for access to the Internet. In some cases, remote campuses may even use default routing to the main campus, choosing to use dynamic routing only locally.

To keep our example simple, for Scenario C, we left the addressing intact from Scenario B. Let's assume that R3 is the ISP for our Company XYZ, which consists of the R1 and R2 routers using the 172.30.0.0/16 major network, subnetted with a /24 mask. Company XYZ is a stub network, meaning that there is only one way in and one way out of the 172.30.0.0/16 network—in via R2 (the gateway router) and out via R3 (the ISP). It doesn't make sense for R2 to send R3 RIP updates for the 172.30.0.0 network every 30 seconds, because R3 has no other way to get to 172.30.0.0 except through R2. It makes more sense for R3 to have a static route configured for the 172.30.0.0/16 network pointing to R2.

How about traffic from Company XYZ toward the Internet? It makes no sense for R3 to send over 120,000 summarized Internet routes to R2. All R2 needs to know is that if a packet is not destined for a host on the 172.30.0.0 network, then it should send the packet to the ISP, R3. This is the same for all other Company XYZ routers (only R1 in our case). They should send all traffic not destined for the 172.30.0.0 network to R2. R2 would then forward the traffic to R3.

Task 1: Make Changes between Scenario B and Scenario C.

Step 1: Remove network 192.168.4.0 from the RIP configuration for R2.

Remove network 192.168.4.0 from the RIP configuration for R2, because no updates will be sent between R2 and R3 and we don't want to advertise the 192.168.4.0 network to R1.

```
R2(config)#router rip
R2(config-router)#no network 192.168.4.0
```

Step 2: Completely remove RIP routing from R3.

```
R3(config)#no router rip
```

Task 2: Configure the Static Route on R3 for the 172.30.0.0/16 network.

Because R3 and R2 are not exchanging RIP updates, we need to configure a static route on R3 for the 172.30.0.0/16 network. This will send all 172.30.0.0/16 traffic to R2.

```
R3(config)#ip route 172.30.0.0 255.255.252.0 serial0/0/1
```

Task 3: Configure a Default Static Route on R2.

Step 1: Configure R2 to send default traffic to R3.

Configure a default static route on R2 that will send all default traffic—packets with destination IP addresses that do not match a specific route in the routing table—to R3.

```
R2(config)# ip route 0.0.0.0 0.0.0.0 serial 0/0/1
```

Step 2: Configure R2 to send default static route information to R1.

The **`default-information originate`** command is used to configure R2 to include the default static route with its RIP updates. Configure this command on R2 so that the default static route information is sent to R1.

```
R2(config)#router rip
R2(config-router)#default-information originate
R2(config-router)#
```

Note: Sometimes it is necessary to clear the RIP routing process before the **`default-information originate`** command will work. First, try the command **`clear ip route *`** on both R1 and R2. This command will cause the routers to immediately flush routes in the routing table and request updates from each other. Sometimes this does not work with RIP. If the default route information is still not sent to R1, save the configuration on R1 and R2 and then reload both routers. Doing this will reset the hardware and both routers will restart the RIP routing process.

Task 4: Verify RIP Routing.

Step 1: Use the `show ip route` command to view the routing table on R2 and R1.

```
R2#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter
area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route
```

```
Gateway of last resort is 0.0.0.0 to network 0.0.0.0

     172.30.0.0/24 is subnetted, 3 subnets
C       172.30.2.0 is directly connected, Serial0/0/0
C       172.30.3.0 is directly connected, FastEthernet0/0
R       172.30.1.0 [120/1] via 172.30.2.1, 00:00:16, Serial0/0/0
     192.168.4.0/30 is subnetted, 1 subnets
C       192.168.4.8 is directly connected, Serial0/0/1
S*   0.0.0.0/0 is directly connected, Serial0/0/1
```

Notice that R2 now has a static route tagged as a **candidate default**.

```
R1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B - BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS inter
area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is 172.30.2.2 to network 0.0.0.0

     172.30.0.0/24 is subnetted, 3 subnets
C       172.30.2.0 is directly connected, Serial0/0/0
R       172.30.3.0 [120/1] via 172.30.2.2, 00:00:05, Serial0/0/0
C       172.30.1.0 is directly connected, FastEthernet0/0
R*   0.0.0.0/0 [120/1] via 172.30.2.2, 00:00:19, Serial0/0/0
```

Notice that R1 now has a RIP route tagged as a **candidate default** route. The route is the "quad-zero" default route sent by R2. R1 will now send default traffic to the **Gateway of last resort** at 172.30.2.2, which is the IP address of R2.

Step 2: View the RIP updates that are sent and received on R1 with the `debug ip rip` command.

```
R1#debug ip rip
RIP protocol debugging is on
R1#RIP: sending  v1 update to 255.255.255.255 via Serial0/0/0
(172.30.2.1)
RIP: build update entries
      network 172.30.1.0 metric 1
RIP: received v1 update from 172.30.2.2 on Serial0/0/0
      0.0.0.0 in 1 hops
      172.30.3.0 in 1 hops
```

Notice that R1 is receiving the default route from R2.

Step 3: Discontinue the debug output with the `undebug all` command.

```
R1#undebug all
All possible debugging has been turned off
```

Step 4: Use the `show ip route` command to view the routing table on R3.

```
R3#show ip route
```

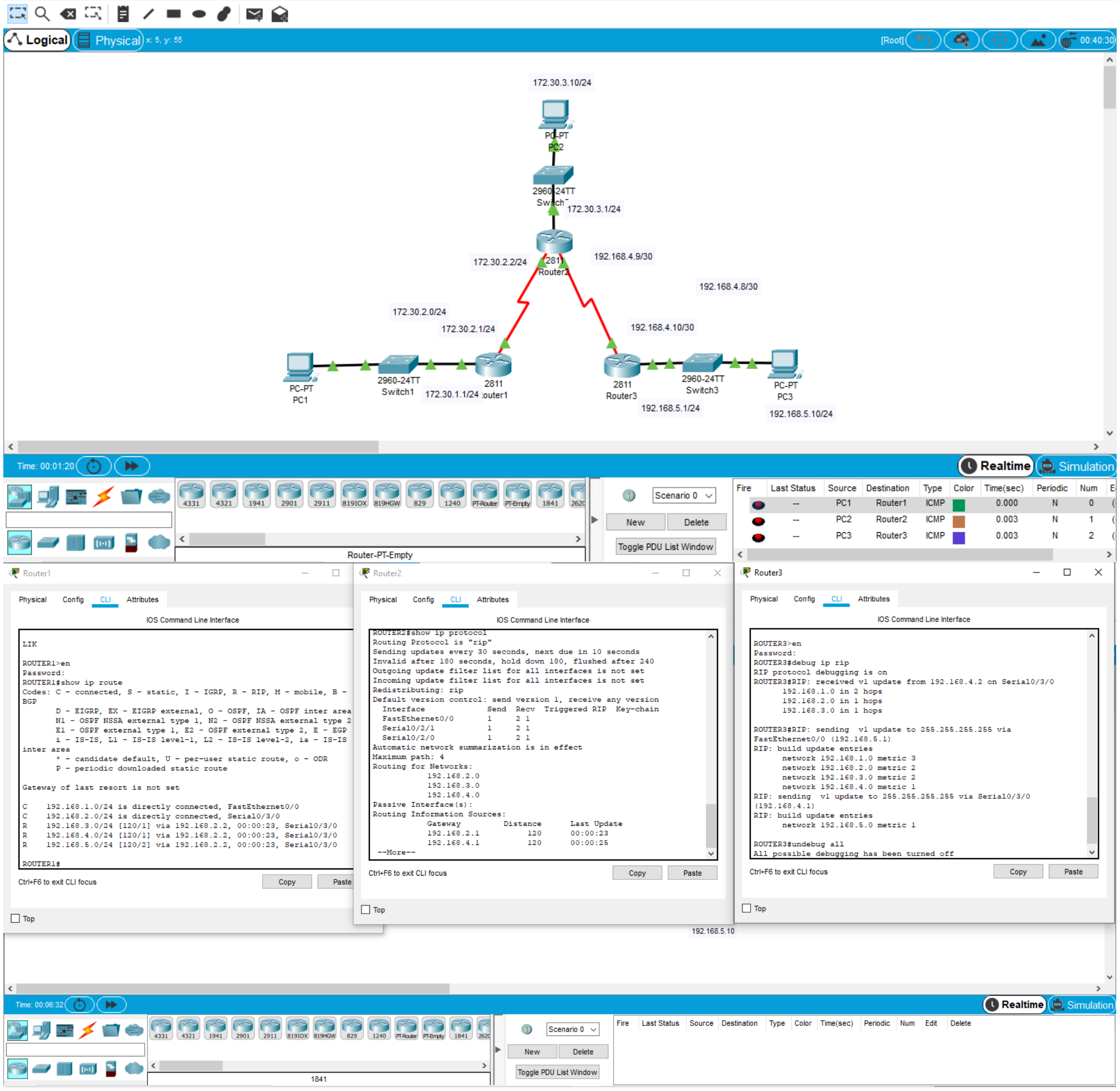
```
S    172.30.0.0/16 is directly connected, Serial0/0/1
     192.168.4.0/30 is subnetted, 1 subnets
C       192.168.4.8 is directly connected, Serial0/0/1
C    192.168.5.0/24 is directly connected, FastEthernet0/0
```

Notice that RIP is not being used on R3. The only route that is not directly connected is the static route.

Task 5: Document the Router Configurations

On each router, capture the following command output to a text file and save for future reference:

- Running configuration
- Routing table
- Interface summarization
- Output from **`show ip protocols`**



Fire	Last Status	Source	Destination	Type	Color	Time(sec)	Periodic	Num
●	--	PC1	Router1	ICMP	🟩	0.000	N	0
●	--	PC2	Router2	ICMP	🟧	0.003	N	1
●	--	PC3	Router3	ICMP	🟪	0.003	N	2

Router1

```
LIK

ROUTER1>en
Password:
ROUTER1#show ip route
Codes: C - connected, S - static, I - IGRP, R - RIP, M - mobile, B -
BGP
       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
       E1 - OSPF external type 1, E2 - OSPF external type 2, E - EGP
       i - IS-IS, L1 - IS-IS level-1, L2 - IS-IS level-2, ia - IS-IS
inter area
       * - candidate default, U - per-user static route, o - ODR
       P - periodic downloaded static route

Gateway of last resort is not set

C    192.168.1.0/24 is directly connected, FastEthernet0/0
C    192.168.2.0/24 is directly connected, Serial0/3/0
R    192.168.3.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.4.0/24 [120/1] via 192.168.2.2, 00:00:23, Serial0/3/0
R    192.168.5.0/24 [120/2] via 192.168.2.2, 00:00:23, Serial0/3/0

ROUTER1#
```

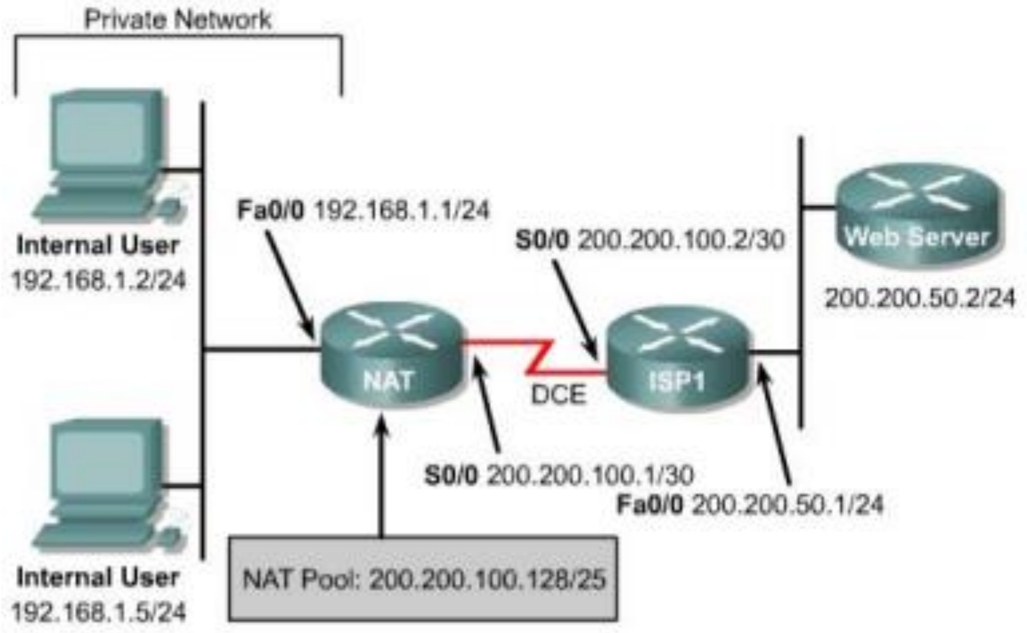
Router2

```
ROUTER2#show ip protocol
Routing Protocol is "rip"
Sending updates every 30 seconds, next due in 10 seconds
Invalid after 180 seconds, hold down 180, flushed after 240
Outgoing update filter list for all interfaces is not set
Incoming update filter list for all interfaces is not set
Redistributing: rip
Default version control: send version 1, receive any version
  Interface             Send  Recv  Triggered RIP  Key-chain
  FastEthernet0/0       1     2 1
  Serial0/2/1           1     2 1
  Serial0/2/0           1     2 1
Automatic network summarization is in effect
Maximum path: 4
Routing for Networks:
          192.168.2.0
          192.168.3.0
          192.168.4.0
Passive Interface(s):
Routing Information Sources:
          Gateway         Distance      Last Update
          192.168.2.1          120      00:00:23
          192.168.4.1          120      00:00:25
 --More--
```

Router3

```
ROUTER3>en
Password:
ROUTER3#debug ip rip
RIP protocol debugging is on
ROUTER3#RIP: received v1 update from 192.168.4.2 on Serial0/3/0
      192.168.1.0 in 2 hops
      192.168.2.0 in 1 hops
      192.168.3.0 in 1 hops

ROUTER3#RIP: sending  v1 update to 255.255.255.255 via
FastEthernet0/0 (192.168.5.1)
RIP: build update entries
      network 192.168.1.0 metric 3
      network 192.168.2.0 metric 2
      network 192.168.3.0 metric 2
      network 192.168.4.0 metric 1
RIP: sending  v1 update to 255.255.255.255 via Serial0/3/0
(192.168.4.1)
RIP: build update entries
      network 192.168.5.0 metric 1

ROUTER3#undebug all
All possible debugging has been turned off
```

5.1. Network Address Translation (NAT)



Objective

In this lab, static Network Address Translation (NAT) and dynamic NAT are configured.

Scenario

The network needs approximately 100 private IP addresses translated in a one-to-one fashion with a pool of public IP addresses. To do this, network will use NAT translation with a portion of its class C address space allocated by ISP1.

Step 1

Build and configure the network according to the diagram.

Use **ping** to test connectivity between the NAT and ISP1 routers, between the workstations and the default gateway, and between WebServer and ISP1.

Step 2

Since no routing protocol will be enabled, configure a default route to the Internet on the

NAT router:
```
NAT(config)#ip route 0.0.0.0 0.0.0.0 200.200.100.2
```

ISP1 needs to be able to reach hosts on the 192.168.0/24 network. But these hosts will have their IP addresses translated to public IP addresses in the 200.200.100.128/25 network, so a static route to the 200.200.100.128/25 network is required:

```
ISP1(config)#ip route 200.200.100.128 255.255.255.128 200.200.100.1
```

Step 3

Create a standard Access Control List that defines all Internal Users:

```
NAT(config)#access-list 1 permit 192.168.1.0 0.0.0.255
```

Step 4

In this step, configure private and public address spaces to be used for NAT and configure the translation:

The public address space 200.200.100.128/25 will be used as a pool to provide NAT translation for the private IP addresses. To statically map the Internal User with IP address 192.168.1.2 pictured in the diagram, enter the following command

```
NAT(config)#ip nat inside source static 192.168.1.2 200.200.100.252
```

This static mapping has the advantage of allowing “external” users to always access the host 192.168.1.2 by way of the fixed IP address 200.200.100.252 (in addition to letting the 192.168.1.2 Internal User access the Internet). On the down side, this external accessibility is also viewed as a security vulnerability. To allow the other hosts on the internal (private) network to reach the Internet, translations will need to be made for those hosts as well. A list of static translations could be made one by one, but a simpler alternative is to configure a pool of addresses and let the router make one-to-one dynamic NAT translations for these hosts. For example, to map the non-statically mapped hosts in the 192.168.1.0/24 network to public IP addresses in the range 200.200.100.129 to200.200.100.250, proceed as follows:

```
NAT(config)#ip nat pool public 200.200.100.129 200.200.100.250
netmask 255.255.255.128
NAT(config)#ip nat inside source list 1 pool public
```

This provides a dynamic one-to-one NAT translation between public IP addresses in the “public” pool and private IP addresses specified by access list 1. The Internal Users IP addresses are configured independently of the NAT translation. Dynamic NAT translations are made for any internal hosts for which no static translation has been defined. The configuration above reserves IP addresses 200.200.100.251 to 200.200.100.254 for use in further static NAT mappings. Static translations are often used with an internal server to enable external access to it by way of a fixed external IP.

Note: If there are more than 128 active hosts on the private network, static NAT translation and/or dynamic one-to-one NAT translations will prevent more than 128 hosts from accessing the Internet. For these additional hosts to get on the Internet, “NAT overloading” must be configured (see Lab 2.10.4b).

Step 5

Now, designate the inside NAT interface and the outside NAT interface. In more complex topologies, it is possible to have more than one inside NAT interface.

```
NAT(config)#interface fastethernet 0/0
NAT(config-if)#ip nat inside

NAT(config-if)#interface serial 0/0
NAT(config-if)#ip nat outside
```

There are several **show** commands that can be used to see if NAT is working: **show ip nat translations**, **show ip nat statistics**, and **show ip nat translations verbose.**

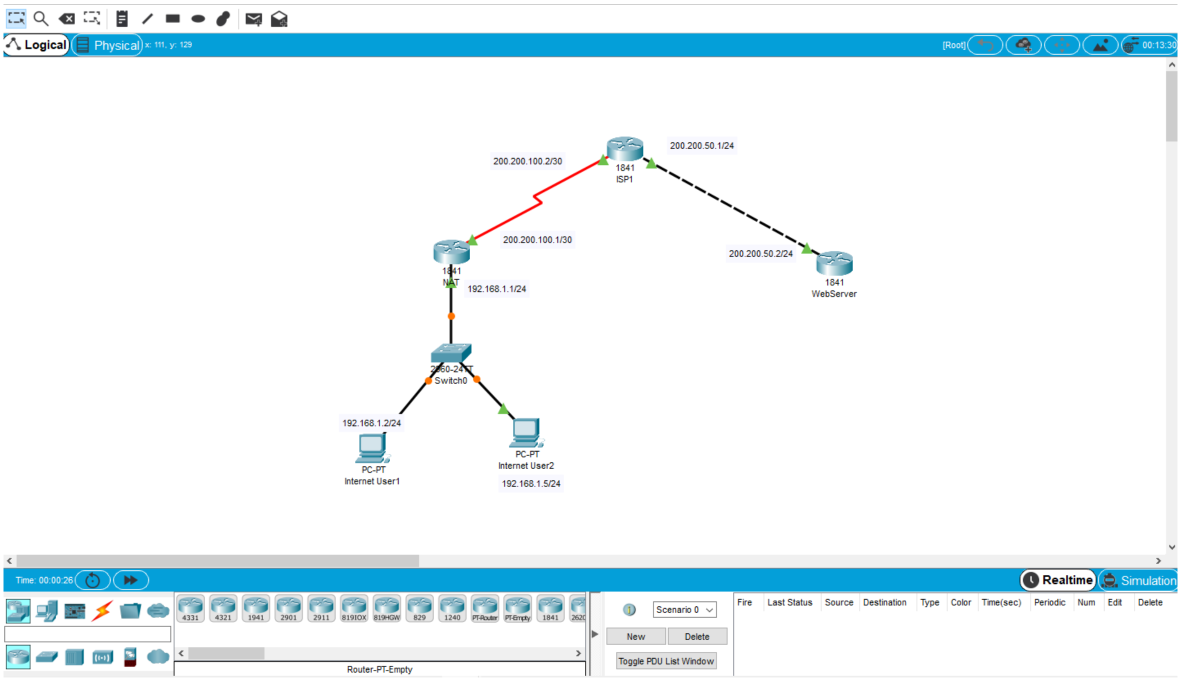
From the two Internal User workstations, ping WebServer (200.200.50.2). Then check that WebServer is accessible by connecting from an Internal User workstation using a browser with the WebServer IP address, 200.200.50.2. Issue the three NAT show commands listed above on the NAT router. Sample outputs are shown below.

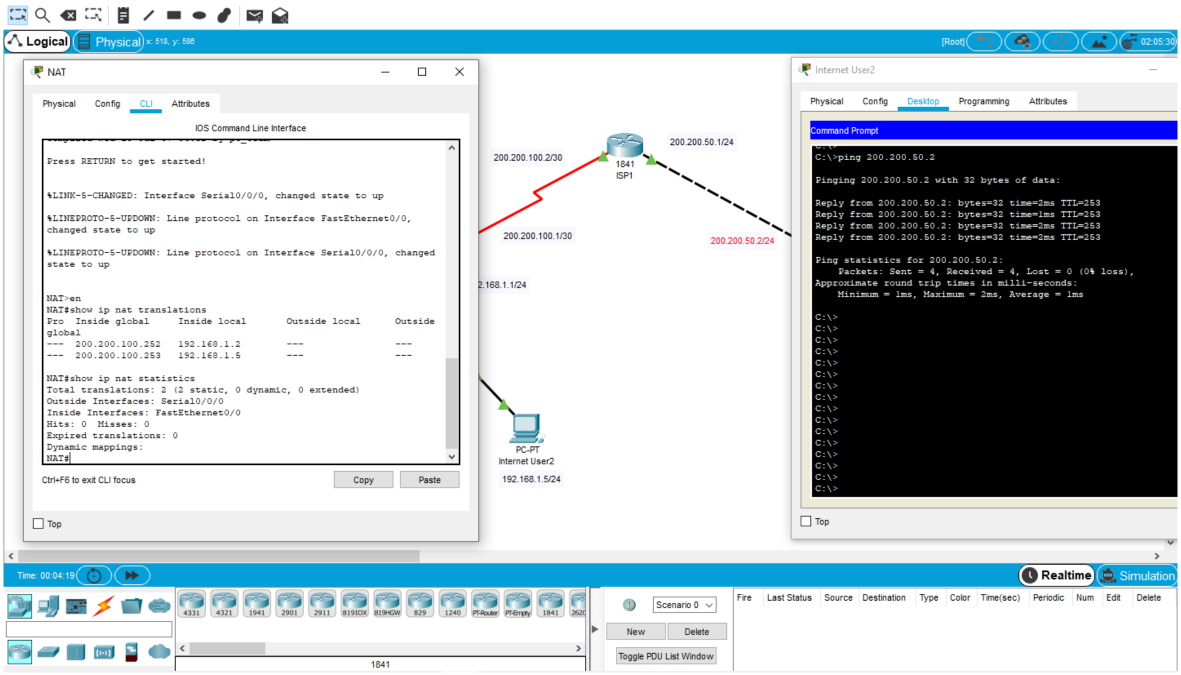
```
NAT#show ip nat translations
Pro Inside global Inside local Outside local Outside global ---
200.200.100.129 192.168.1.5 --- ---
--- 200.200.100.252 192.168.1.2 --- ---


NAT#show ip nat statistics
Total active translations: 2 (1 static, 1 dynamic; 0 extended)
Outside interfaces:
  Serial0/0
Inside interfaces:
  FastEthernet0/0
Hits: 131 Misses: 9
Expired translations: 0
Dynamic mappings:
-- Inside Source
 [Id: 2] access-list 1 pool public refcount 1
 pool public: netmask 255.255.255.128
        start 200.200.100.129 end 200.200.100.250
type generic, total addresses 122, allocated 1 (0%), misses 0


NAT#show ip nat translations verbose
Pro Inside global Inside local Outside local Outside global ---
200.200.100.129 192.168.1.5 --- ---
    create 00:02:55, use 00:02:55, left 23:57:04, Map-Id(In): 2,
    flags:
none, use_count: 0
--- 200.200.100.252 192.168.1.2 --- ---
    create 00:40:36, use 00:02:59,
    flags:
static, use_count: 0
```

Notice that the Internal User with IP address 192.168.1.5 had its address dynamically translated to 200.200.100.129, the first available address in the “public” pool. The command **clear ip nat translation *** can be used to clear all dynamic NAT translations:

```
NAT#clear ip nat translation *

NAT#show ip nat translations

Pro Inside global Inside local Outside local Outside global ---

200.200.100.252 192.168.1.2 --- ---
```

Save the configurations for NAT and ISP1.

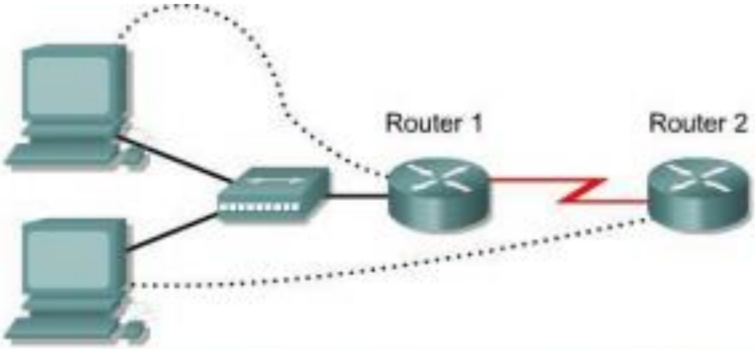


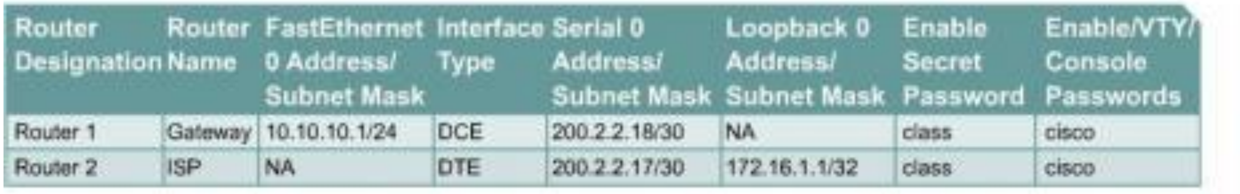


```
Press RETURN to get started!


%LINK-5-CHANGED: Interface Serial0/0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0,
changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed
state to up


NAT>en
NAT#show ip nat translations
Pro  Inside global     Inside local       Outside local      Outside
global
---  200.200.100.252   192.168.1.2        ---                ---
---  200.200.100.253   192.168.1.5        ---                ---

NAT#show ip nat statistics
Total translations: 2 (2 static, 0 dynamic, 0 extended)
Outside Interfaces: Serial0/0/0
Inside Interfaces: FastEthernet0/0
Hits: 0  Misses: 0
Expired translations: 0
Dynamic mappings:
NAT#
```

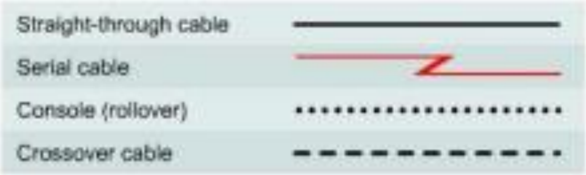
```
C:\>ping 200.200.50.2

Pinging 200.200.50.2 with 32 bytes of data:

Reply from 200.200.50.2: bytes=32 time=2ms TTL=253
Reply from 200.200.50.2: bytes=32 time=1ms TTL=253
Reply from 200.200.50.2: bytes=32 time=2ms TTL=253
Reply from 200.200.50.2: bytes=32 time=2ms TTL=253

Ping statistics for 200.200.50.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
```

5.2. Verifying NAT and PAT Configuration





Router Designation	Router Name	FastEthernet 0 Address/ Subnet Mask	Interface Type	Serial 0 Address/ Subnet Mask	Loopback 0 Address/ Subnet Mask	Enable Secret Password	Enable/VTY/ Console Passwords
Router 1	Gateway	10.10.10.1/24	DCE	200.2.2.18/30	NA	class	cisco
Router 2	ISP	NA	DTE	200.2.2.17/30	172.16.1.1/32	class	cisco



Objective

• Configure a router for Network Address Translation (NAT) and Port Address Translation (PAT) • Test the configuration and verify NAT/PAT statistics

Background/Preparation

The ISP has allocated a company the public CIDR IP address 199.99.9.32/30. This is equivalent to four public IP addresses. Since the company has an internal requirement for more than 30 addresses, the IT manager has decided to use NAT with PAT. Routing between the ISP and the gateway router is done using a static route between the ISP and the gateway, and a default route between the gateway and the ISP. The ISP connection to the Internet will be represented by a loopback address on the ISP router.

Cable a network similar to the one in the diagram above. Any router that meets the interface requirements displayed on the above diagram may be used. This includes the following and any of their possible combinations:

- 800 series routers
- 1600 series routers
- 1700 series routers
- 2500 series routers
- 2600 series routers

.

Please refer to the chart at the end of the lab to correctly identify the interface identifiers to be used based on the equipment in the lab. The configuration output used in this lab is produced from 1721 series routers. Any other router used may produce slightly different output. Conduct the following steps on each router unless specifically instructed otherwise.

Start a HyperTerminal session.

Note: Refer to the erase and reload instructions at the end of this lab. Perform those steps on all routers in this lab assignment before continuing.

Step 1 Configure the routers

Configure all of the following according to the chart:

- The hostname
- The console
- The virtual terminal
- The enable passwords
- The interfaces

If problems occur during this configuration, refer to Lab 1.1.4a Configuring NAT.

Step 2 Save the configuration

At the privileged EXEC mode prompt, on both routers, type the command **`copy running config startup-config`**.

Step 3 Configure the hosts with the proper IP address, subnet mask, and default gateway

Each workstation should be able to ping the attached router. If for some reason this is not the case, troubleshoot as necessary. Check and verify that the workstation has been assigned a specific IP address and default gateway. If running Windows 98, check using **Start** > **Run** > **winipcfg**. If running Windows 2000 or higher, check using **`ipconfig`** in a DOS window.

Step 4 Verify that the network is functioning

a. From the attached hosts, ping the FastEthernet interface of the default gateway router. b. Was the ping from the first host successful? - YES

c. Was the ping from the second host successful? - YES

d. If the answer is no for either question, troubleshoot the router and host configurations to find the error. Then ping again until they both are successful.

Step 5 Create a static route

a. Create a static route from the ISP to the Gateway router. Addresses 199.99.9.32/27 have been allocated for Internet access outside of the company. Use the **`ip route`** command to create the static route.

```
ISP(config)#ip route 199.99.9.32 255.255.224.0 200.2.2.18
```

Step 6 Create a default route

a. Add a default route, using the **`ip route`** command, from the Gateway router to the ISP router. This will forward any unknown destination address traffic to the ISP:

```
Gateway(config)#ip route 0.0.0.0 0.0.0.0 200.2.2.17
```

Step 7 Define the pool of usable public IP addresses

To define the pool of public addresses, use the **`ip nat pool`** command:

```
Gateway(config)#ip nat pool public-access 199.99.9.32
199.99.9.35 netmask 255.255.255.252
```

Step 8 Define an access list that will match the inside private IP addresses

To define the access list to match the inside private addresses, use the **access list** command:

```
Gateway(config)#access-list 1 permit 10.10.10.0 0.0.0.255
```

Step 9 Define the NAT translation from inside list to outside pool

To define the NAT translation, use the **ip nat inside source** command:

```
Gateway(config)#ip nat inside source list 1 pool public-access overload
```

Step 10 Specify the interfaces

The active interfaces on the router need to be identified as either inside or outside interfaces with respect to NAT. To do this, use the **ip nat inside** or **ip nat outside** command:

```
Gateway(config)#interface fastethernet 0
Gateway(config-if)#ip nat inside
Gateway(config-if)#interface serial 0
Gateway(config-if)#ip nat outside
```

Step 11 Testing the configuration

a. From the workstations, **ping 172.16.1.1**. Open multiple DOS windows on each workstation and Telnet to the 172.16.1.1 address. Next, view the NAT translations on the Gateway router, with the command **show ip nat translations**.

Step 12 Verify NAT and PAT Statistics

a. To view the NAT and PAT statistics type the **show ip nat statistics** command at the privileged EXEC mode prompt.

Upon completion of the previous steps finish the lab by doing the following:

- Logoff by typing **exit**
- Turn the router off
- Remove and store the cables and adapter

Configuration reference sheet

This sheet contains the basic configuration commands for the ISP and Gateway routers:

```
ISP
Router#configure terminal
Router(config)#hostname ISP
ISP(config)#enable password cisco
ISP(config)#enable secret class
ISP(config)#line console 0
ISP(config-line)#password cisco
ISP(config-line)#login
ISP(config-line)#exit
ISP(config)#line vty 0 4
ISP(config-line)#password cisco
ISP(config-line)#login
ISP(config-line)#exit
ISP(config)#interface loopback 0
ISP(config-if)#ip address 172.16.1.1
255.255.255.255 ISP(config-if)#no shutdown
ISP(config-if)#exit
ISP(config)#interface serial 0
ISP(config-if)#ip address 200.2.2.17
255.255.255.252 ISP(config-if)#no shutdown
ISP(config-if)#clockrate 64000
ISP(config)#ip route 199.99.9.32 255.255.255.224
200.2.2.18 ISP(config)#end
ISP#copy running-config startup-config
```

Gateway

```
Router#configure terminal
Router(config)#hostname Gateway
Gateway(config)#enable password cisco
Gateway(config)#enable secret class
Gateway(config)#line console 0
Gateway(config-line)#password cisco
Gateway(config-line)#login
Gateway(config-line)#exit
Gateway(config)#line vty 0 4
Gateway(config-line)#password cisco
Gateway(config-line)#login
Gateway(config-line)#exit
Gateway(config)#interface fastethernet 0
Gateway(config-if)#ip address 10.10.10.1
255.255.255.0 Gateway(config-if)#no shutdown
Gateway(config-if)#exit
Gateway(config)#interface serial 0
Gateway(config-if)#ip address 200.2.2.18
255.255.255.252 Gateway(config-if)#no shutdown
Gateway(config)#ip route 0.0.0.0 0.0.0.0 200.2.2.17
```

Erasing and reloading the router

Enter into the privileged EXEC mode by typing **enable**.

If prompted for a password, enter **class** (if that does not work, ask the

instructor). `Router>enable`

At the privileged EXEC mode, enter the command **erase startup-config**.

```
Router#erase startup-config
```

The responding line prompt will be:

```
Erasing the nvram filesystem will remove all files! Continue? [confirm]
```

Press **Enter** to confirm.

The response should be:

```
   Erase of nvram: complete
```

Now at the privileged EXEC mode, enter the command **`reload`**.

```
   Router(config)#reload
```

The responding line prompt will be:

```
   System configuration has been modified. Save? [yes/no]:
```

Type **n** and then press **Enter**.

The responding line prompt will be:

```
   Proceed with reload? [confirm]
```

Press **Enter** to confirm.

In the first line of the response will be:

```
   Reload requested by console.
```

After the router has reloaded the line prompt will be:

```
   Would you like to enter the initial configuration dialog?
[yes/no]:
```

Type **n** and then press **Enter**.

The responding line prompt will be:

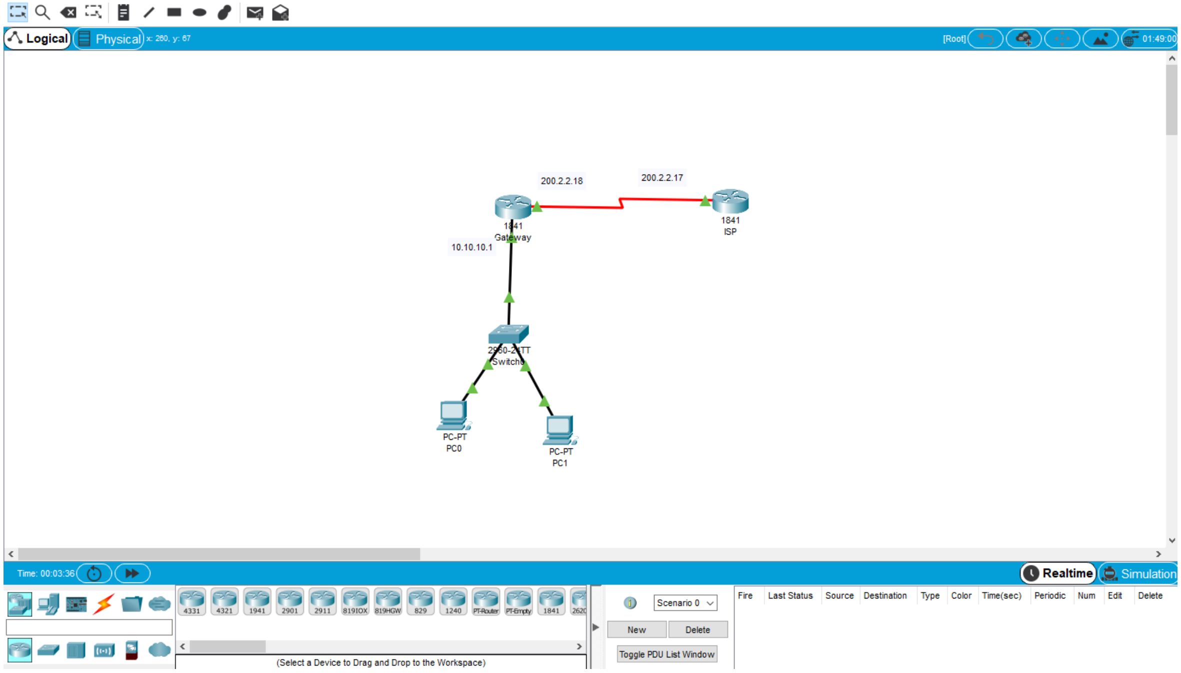
```
   Press RETURN to get started!
```

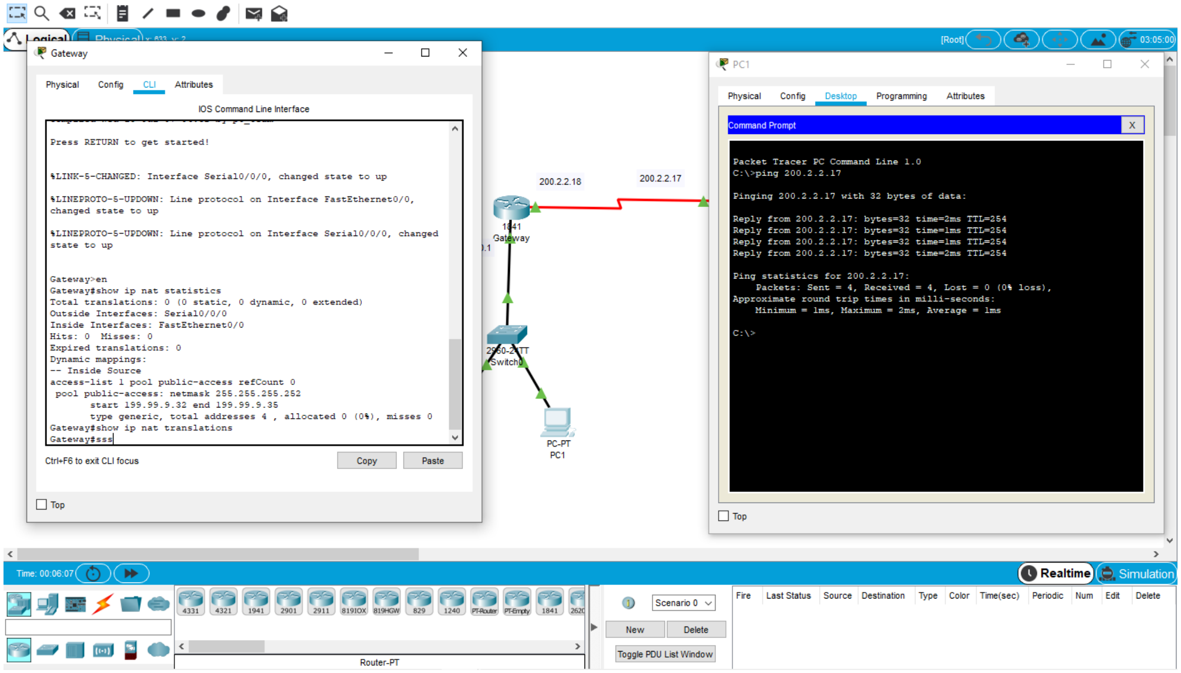
Press **Enter**.

Now the router is ready for the assigned lab to be performed.

Router Serial

Ethernet Interface #1	Ethernet Interface #2	Serial Interface #1
Ethernet 0 (E0)	Ethernet 1 (E1)	NA
Ethernet 0 (E0)	Ethernet 1 (E1)	Serial 0 (S0)
FastEthernet 0 (FA0)	FastEthernet 1 (FA1)	Serial 0 (S0)
Ethernet 0 (E0)	Ethernet 1 (E1)	Serial 0 (S0)
FastEthernet 0/0 (FA0/0)	FastEthernet 0/1 (FA0/1)	Serial 0/0 (S0/0)

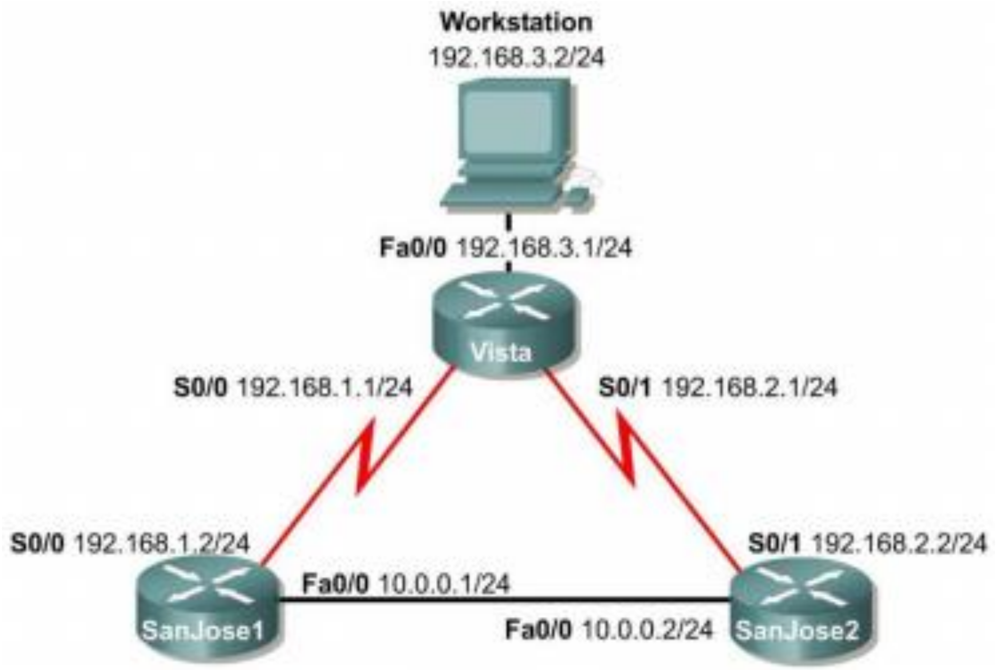




```
Press RETURN to get started!


%LINK-5-CHANGED: Interface Serial0/0/0, changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface FastEthernet0/0,
changed state to up

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial0/0/0, changed
state to up


Gateway>en
Gateway#show ip nat statistics
Total translations: 0 (0 static, 0 dynamic, 0 extended)
Outside Interfaces: Serial0/0/0
Inside Interfaces: FastEthernet0/0
Hits: 0  Misses: 0
Expired translations: 0
Dynamic mappings:
-- Inside Source
access-list 1 pool public-access refCount 0
 pool public-access: netmask 255.255.255.252
       start 199.99.9.32 end 199.99.9.35
       type generic, total addresses 4 , allocated 0 (0%), misses 0
Gateway#show ip nat translations
Gateway#sss
```

```
Packet Tracer PC Command Line 1.0
C:\>ping 200.2.2.17

Pinging 200.2.2.17 with 32 bytes of data:

Reply from 200.2.2.17: bytes=32 time=2ms TTL=254
Reply from 200.2.2.17: bytes=32 time=1ms TTL=254
Reply from 200.2.2.17: bytes=32 time=1ms TTL=254
Reply from 200.2.2.17: bytes=32 time=2ms TTL=254

Ping statistics for 200.2.2.17:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 1ms, Maximum = 2ms, Average = 1ms

C:\>
```

6. Access Control List Basics and Extended Ping



Objective

This lab activity reviews the basics of standard and extended access lists.

Scenario

The LAN users connected to the Vista router are concerned about access to the network from hosts on network 10.0.0.0. A standard access list must be used to block all access to the Vista LAN from network 10.0.0.0 /24.

Also, an extended ACL must be used to block network 192.168.3.0 host access to Web servers on the 10.0.0.0 /24 network.

Step 1

Build and configure the network according to the diagram. Use RIPv1, and enable updates on all active interfaces with the appropriate **network** commands. The commands necessary to configure SanJose1 are shown in the following example:

```
SanJose1(config)#router rip
SanJose1(config-router)#network 192.168.1.0
SanJose1(config-router)#network 10.0.0.0
```

Use the **ping** command to verify the work and test connectivity between all interfaces.

Step 2

Check the routing table on Vista using the **show ip route** command. Vista should have all four networks in the routing table. Troubleshoot, if necessary

Access Control List Basics

Access Control Lists (ACLs) are simple but powerful tools. When the access list is configured, each statement in the list is processed by the router in the order in which it was created. If an individual packet meets a statement's criteria, the permit or deny is applied to that packet, and no further list entries are checked. Each packet starts at the top of the list, every time.

It is not possible to reorder an access list, skip statements, edit statements, or delete statements from a numbered access list, while in the router configuration mode. With numbered access lists, any attempt to delete a single statement results in deletion of the entire list. Named ACLs (NACLs) do allow for the deletion of individual statements. It is suggested that ACLs, of all kinds, be created in an off-line editor and pasted into the configuration.

The following concepts apply to both standard and extended access lists:

Two step process

The access list may be created with one or more **`access-list`** commands while in global configuration mode. Second, the access list is applied to or referenced by other commands, such as the **ip `access-group`** command which applies the ACL to an interface. An example would be the following:

```
Vista#config terminal
Vista(config)#access-list 50 deny 10.0.0.0 0.0.0.255
Vista(config)#access-list 50 permit any
Vista(config)#interface fastethernet 0/0
Vista(config-if)#ip access-group 50 out
Vista(config-if)#^Z
```

Syntax and Keywords

The basic syntax for creating an access list entry is as follows:

```
router(config)#access-list # {permit | deny}ip address wildcard
mask
```

The **`permit`** command allows packets matching the specified criteria to be accepted for whatever application the access list is being used for. The **`deny`** command discards packets matching the criteria on that line.

Two important keywords, **any** and **host**, can be used with IP addresses and the access list. The keyword **any** matches all hosts on all networks, equivalent to **0.0.0.0 255.255.255.255**. The keyword **host** can be used with an IP address to indicate a single host address. The syntax is **`host`** *`ip address`* (**`host 192.168.1.10`**). This is the same as entering 192.168.1.10 0.0.0.0.

Implicit deny statement

Every access list contains a final “deny” statement that matches all packets. This is called the implicit deny. Because the implicit deny statement is not visible in **`show`** command output, it is often overlooked, with serious consequences. As an example, consider the following single-line access list:

```
Router(config)#access-list 75 deny host 192.168.1.10
```

Access- list 75 clearly denies all traffic sourced from the host, 192.168.1.10. What might not be obvious is that all other traffic will be discarded as well. This is because the implicit **`deny any`** is the final statement in any access list.

At least one permit statement is required

There is no requirement that an ACL contain a **`deny`** statement. If nothing else, the implicit **`deny any`** statement takes care of that. But if there are no **`permit`** statements, the effect will be the same as if there were only a single **`deny any`** statement.

Wildcard mask

In identifying IP addresses, ACLs use a wildcard mask instead of a subnet mask. Initially, the masks might look the same, but closer observation reveals that they are very different. Remember that a binary 0 in a wildcard mask instructs the router to match the corresponding bit in the IP address. **In/out**

When deciding whether an ACL should be applied to inbound or outbound traffic, always view things from the perspective of the router. In other words, determine whether traffic is coming into the router, inbound, or leaving the router, outbound.

Applying ACLs

Extended ACLs should be applied as close to the source as possible, thereby conserving network resources. Standard ACLs, by necessity, must be applied as close to the destination as possible. This is because the standard ACL can match only at the source address of a packet.

Step 3

On the Vista router, create the following standard ACL and apply it to the LAN interface:

```
Vista#config terminal
Vista(config)#access-list 50 deny 10.0.0.0 0.0.0.255
Vista(config)#access-list 50 permit any
Vista(config)#interface fastethernet 0/0
Vista(config-if)#ip access-group 50 out
Vista(config-if)#^Z
```

Try pinging 192.168.3.2 from SanJose1.

The ping should be successful. This result might be surprising, because all traffic from the 10.0.0.0/8 network was just blocked. The ping is successful because, even though it came from SanJose1, it is not sourced from the 10.0.0.0/8 network. A ping or traceroute from a router uses the closest interface to the destination as the source address. Therefore, the ping is coming from the 192.168.1.0/24, SanJose1’s Serial 0/0.

```
SanJose1#ping 192.168.3.2
Sending 5, 100-byte ICMP Echos to 192.168.3.2, timeout
 is 2 seconds: !!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max =
4/4/4 ms
```

Step 4

In order to test the ACL from SanJose1, the extended ping command must be used to specify a source interface as follows:

On SanJose1, issue the following commands:

```
SanJose1#ping
Protocol [ip]:
Target IP address: 192.168.3.2
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 10.0.0.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.3.2, timeout is 2
seconds:
```

```
.....
Success rate is 0 percent (0/5)
```

Step 5

Standard ACLs are numbered 1 - 99. IOS version 12.xx allows additional numbering from 1300 - 1699. Extended ACLs are numbered 100 - 199. IOS version 12.xx allows additional numbering from 2000 - 2699. Extended ACLs can be used to enforce highly specific criteria for filtering packets. In this step, configure an extended ACL to block access to a Web server. Before proceeding, issue the **no access-list 50** and **no ip access-group 50** commands on the Vista router to remove the ACL configured previously.

First, configure both SanJose1 and SanJose2 to act as Web servers, by using the **ip http server** command, as shown in the following:

```
SanJose1(config)#ip http server
SanJose2(config)#ip http server
```

From the workstation at 192.168.3.2, use a Web browser to view both Web servers on the router at 10.0.0.1 and 10.0.0.2. The Web login requires that the enable secret password for the router be entered as the password.

After verifying Web connectivity between the workstation and the routers, proceed to Step 6.

Step 6

On the Vista router, enter the following commands:

```
Vista(config)#access-list 101 deny tcp 192.168.3.0 0.0.0.255
10.0.0.0  0.0.0.255 eq www
Vista(config)#access-list 101 deny tcp 192.168.3.0 0.0.0.255
any eq ftp Vista(config)#access-list 101 permit ip any any
Vista(config)#interface fastethernet 0/0
Vista(config-if)#ip access-group 101 in
```

From the workstation at 192.168.3.2, again attempt to view the Web servers at 10.0.0.1 and 10.0.0.2. Both attempts should fail.

Next, browse SanJose1 at 192.168.1.2.

Dijsktra's Algorithm

Introduction

Dijkstra's algorithm is very similar to Prim's algorithm for a minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with a given source as root. We maintain two sets, one set contains vertices included in the shortest path tree, the other set includes vertices not yet included in the shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Below are the detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

Algorithm

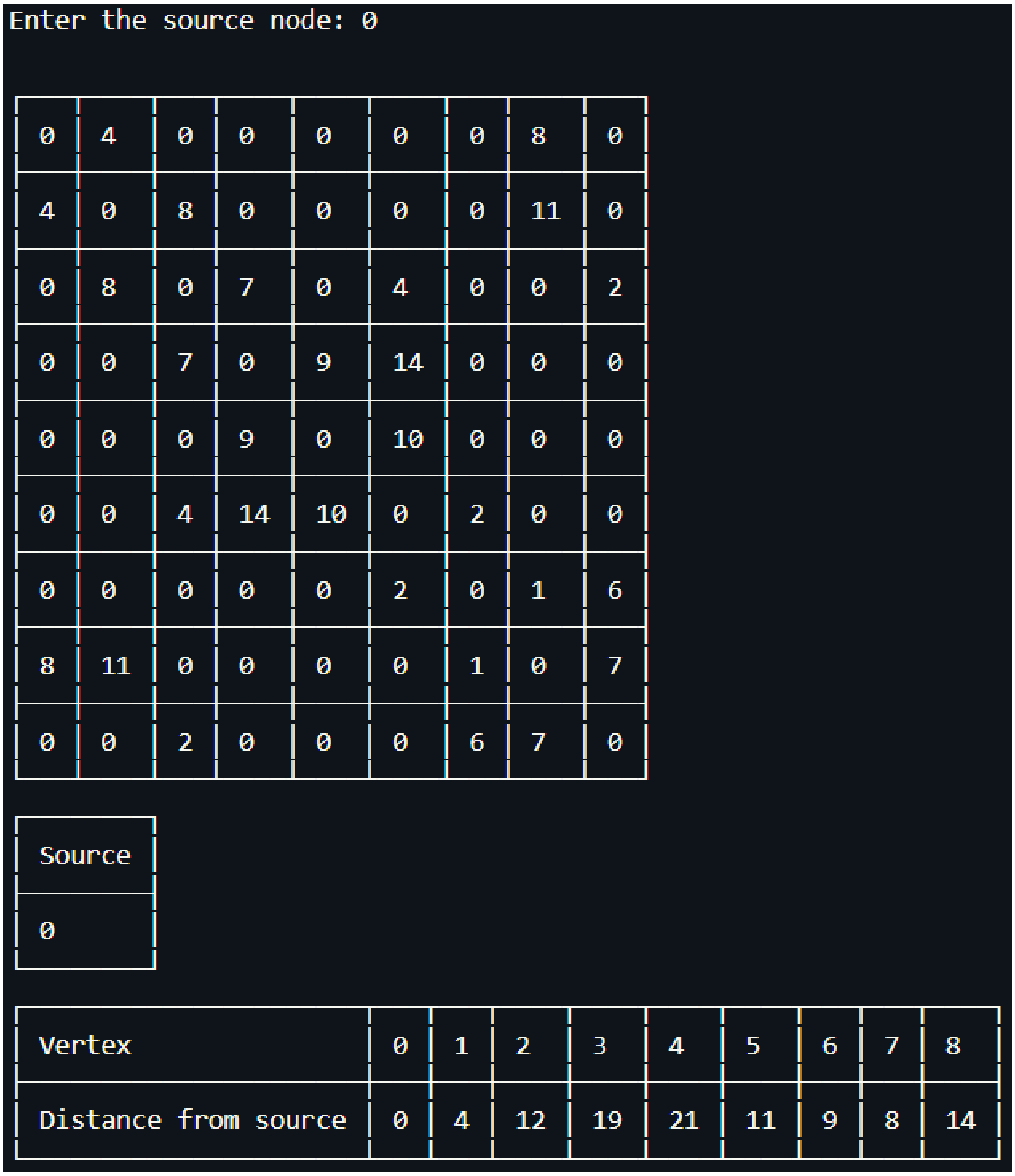
1. Create a set sptSet (shortest path tree set) that keeps track of vertices included in the shortest path tree, whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2. Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
3. While sptSet doesn't include all vertices.
    a. Pick a vertex u which is not there in sptSet and has minimum distance value.
    b. Include u to sptSet.
    c. Update the distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if the sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Code

```
import sys
from terminaltables import SingleTable
from common import clear


class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printFinalDistances(self,src, dist):
        clear()
        self.printGraph()
        sourceInfo = ["Source"], [src]
        sourceTable = SingleTable(sourceInfo)
        print(sourceTable.table)

        data = ["Vertex"]+list(range(self.V)), ["Distance from
source"]+dist
        table = SingleTable(data)
        print(table.table)

    def printGraph(self):
        table = SingleTable(self.graph)
        table.inner_row_border = True
        print(table.table)


    def findMinimumDistance(self, dist, sptSet):
        min = sys.maxsize
        for v in range(self.V):
            if dist[v] < min and sptSet[v] == False:
                min = dist[v]
                min_index = v
        return min_index

    def dijkstra(self, src):
        dist = [sys.maxsize] * self.V
        dist[src] = 0
        sptSet = [False] * self.V

        for cout in range(self.V):
            u = self.findMinimumDistance(dist, sptSet)
            sptSet[u] = True
            for v in range(self.V):
                if self.graph[u][v] > 0 and sptSet[v] == False and
dist[v] > dist[u] + self.graph[u][v]:
                    dist[v] = dist[u] + self.graph[u][v]

        self.printFinalDistances(src,dist)


g = Graph(9)
g.graph = [
    [0, 4, 0, 0, 0, 0, 0, 8, 0],
    [4, 0, 8, 0, 0, 0, 0, 11, 0],
    [0, 8, 0, 7, 0, 4, 0, 0, 2],
    [0, 0, 7, 0, 9, 14, 0, 0, 0],
    [0, 0, 0, 9, 0, 10, 0, 0, 0],
    [0, 0, 4, 14, 10, 0, 2, 0, 0],
    [0, 0, 0, 0, 0, 2, 0, 1, 6],
    [8, 11, 0, 0, 0, 0, 1, 0, 7],
    [0, 0, 2, 0, 0, 0, 6, 7, 0]
]
```

```
g.dijkstra(int(input("Enter the source node: ")))
```

Result



```
Enter the source node: 0

| 0 | 4  | 0 | 0  | 0  | 0  | 0 | 8  | 0 |
| 4 | 0  | 8 | 0  | 0  | 0  | 0 | 11 | 0 |
| 0 | 8  | 0 | 7  | 0  | 4  | 0 | 0  | 2 |
| 0 | 0  | 7 | 0  | 9  | 14 | 0 | 0  | 0 |
| 0 | 0  | 0 | 9  | 0  | 10 | 0 | 0  | 0 |
| 0 | 0  | 4 | 14 | 10 | 0  | 2 | 0  | 0 |
| 0 | 0  | 0 | 0  | 0  | 2  | 0 | 1  | 6 |
| 8 | 11 | 0 | 0  | 0  | 0  | 1 | 0  | 7 |
| 0 | 0  | 2 | 0  | 0  | 0  | 6 | 7  | 0 |

| Source |
| 0      |

| Vertex               | 0 | 1 | 2  | 3  | 4  | 5  | 6 | 7 | 8  |
| Distance from source | 0 | 4 | 12 | 19 | 21 | 11 | 9 | 8 | 14 |
```

Prim’s Algorithm

Introduction

Like Kruskal’s algorithm, Prim’s algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

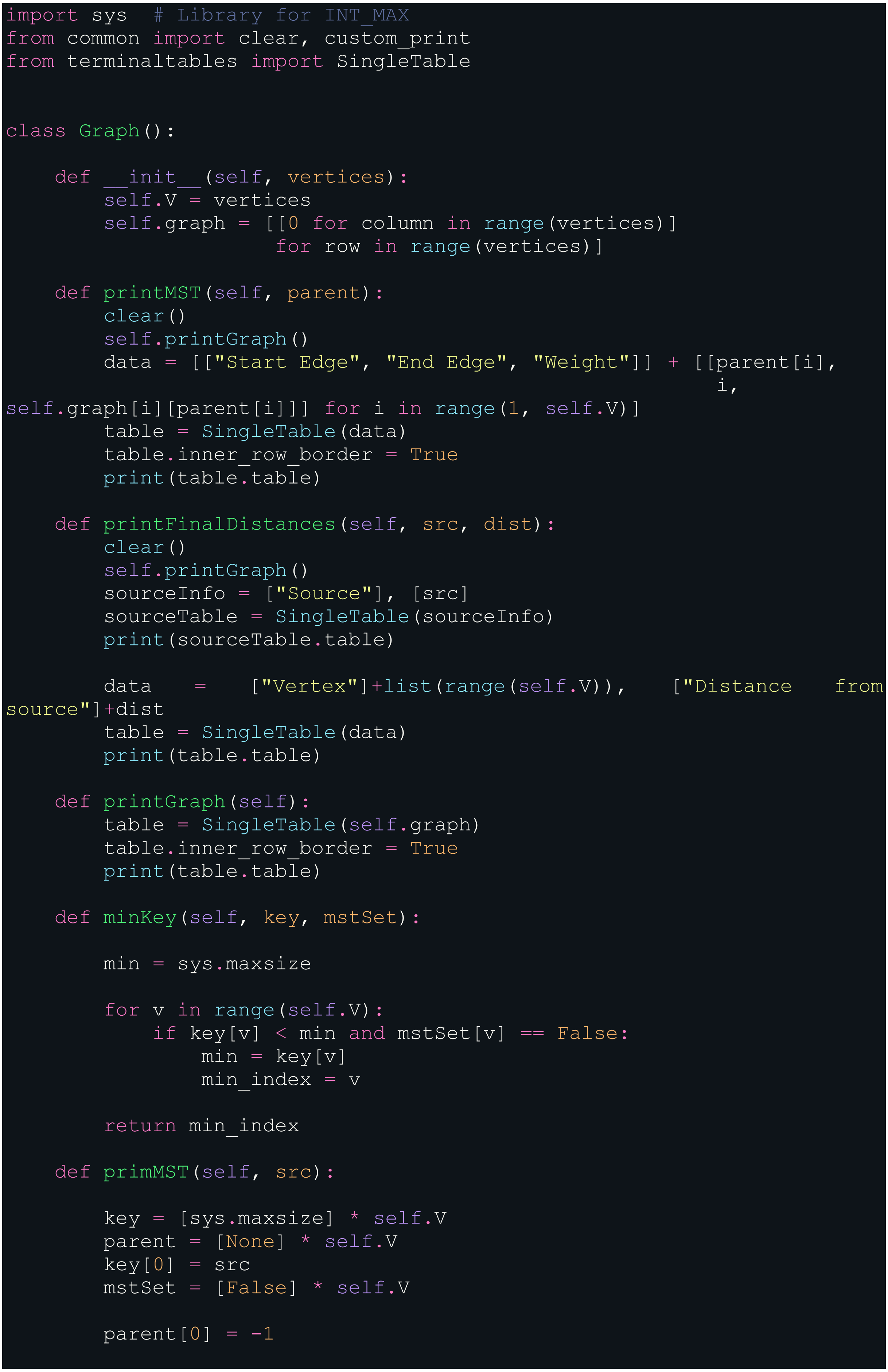
A group of edges that connects two sets of vertices in a graph is called cut in graph theory. So, at every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

Algorithm

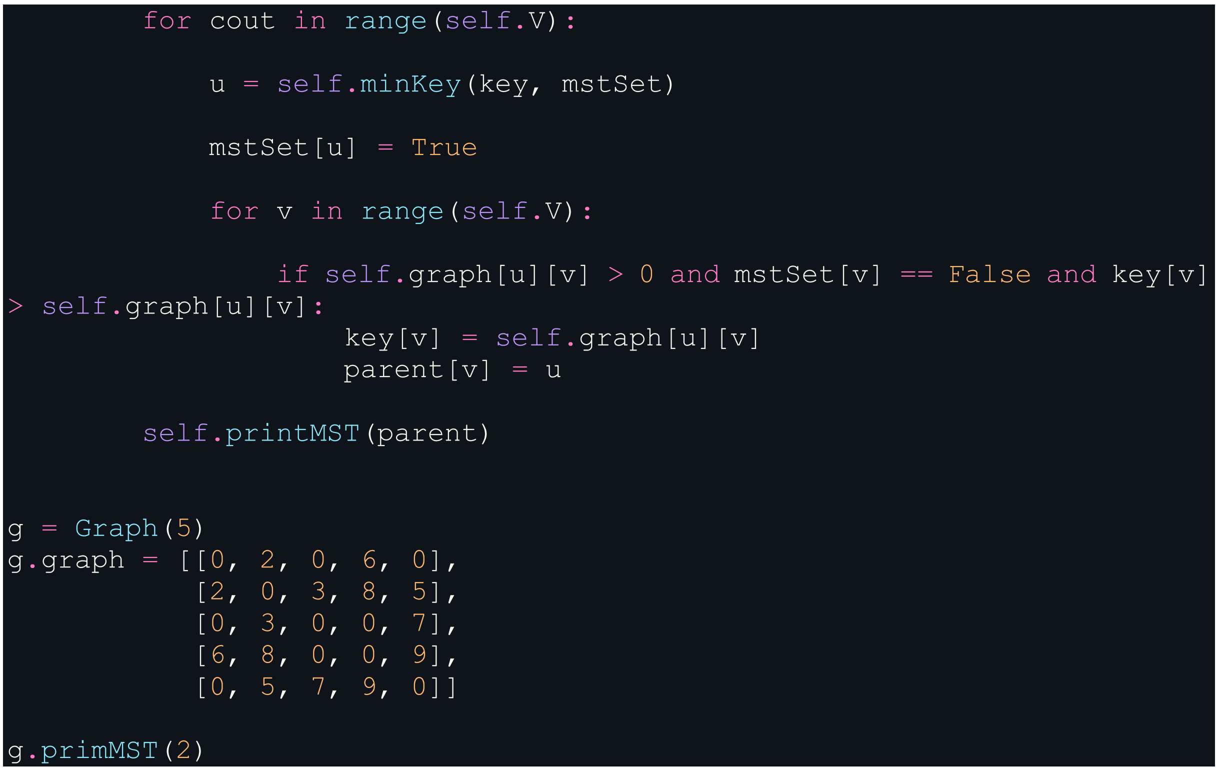
1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices.
    a. Pick a vertex u which is not there in mstSet and has minimum key value.
    b. Include u to mstSet.
    c. Update key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices. For every adjacent vertex v, if weight of edge u-v is less than the previous key value of v, update the key value as weight of u-v

The idea of using key values is to pick the minimum weight edge from the cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

Code

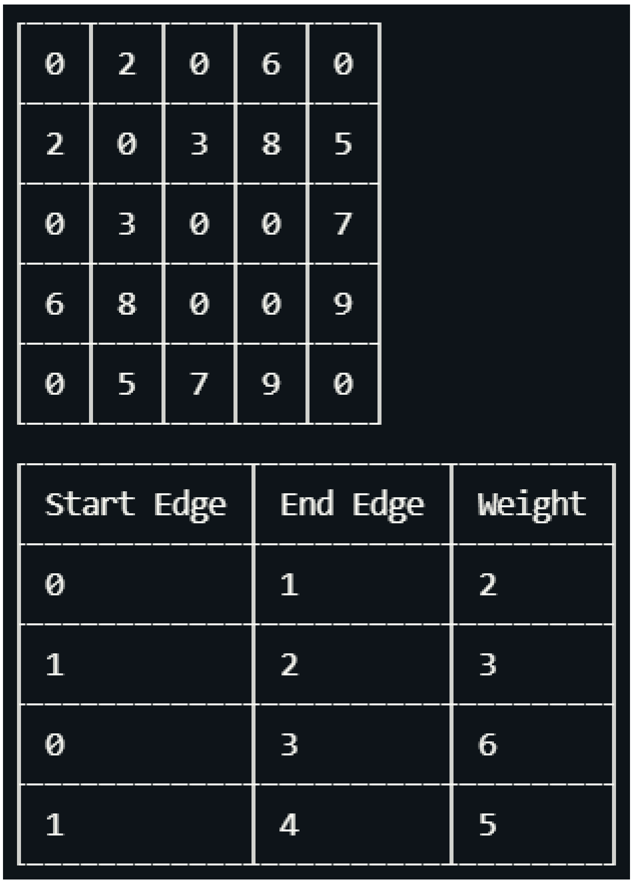


```
import sys  # Library for INT_MAX
from common import clear, custom_print
from terminaltables import SingleTable


class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printMST(self, parent):
        clear()
        self.printGraph()
        data = [["Start Edge", "End Edge", "Weight"]] + [[parent[i],
                                                          i,
self.graph[i][parent[i]]] for i in range(1, self.V)]
        table = SingleTable(data)
        table.inner_row_border = True
        print(table.table)

    def printFinalDistances(self, src, dist):
        clear()
        self.printGraph()
        sourceInfo = ["Source"], [src]
        sourceTable = SingleTable(sourceInfo)
        print(sourceTable.table)

        data = ["Vertex"]+list(range(self.V)), ["Distance from
source"]+dist
        table = SingleTable(data)
        print(table.table)

    def printGraph(self):
        table = SingleTable(self.graph)
        table.inner_row_border = True
        print(table.table)

    def minKey(self, key, mstSet):

        min = sys.maxsize

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v

        return min_index

    def primMST(self, src):

        key = [sys.maxsize] * self.V
        parent = [None] * self.V
        key[0] = src
        mstSet = [False] * self.V

        parent[0] = -1
```



```
        for cout in range(self.V):

            u = self.minKey(key, mstSet)

            mstSet[u] = True

            for v in range(self.V):

                if self.graph[u][v] > 0 and mstSet[v] == False and key[v]
> self.graph[u][v]:
                    key[v] = self.graph[u][v]
                    parent[v] = u

        self.printMST(parent)


g = Graph(5)
g.graph = [[0, 2, 0, 6, 0],
           [2, 0, 3, 8, 5],
           [0, 3, 0, 0, 7],
           [6, 8, 0, 0, 9],
           [0, 5, 7, 9, 0]]

g.primMST(2)
```

Result



0	2	0	6	0
2	0	3	8	5
0	3	0	0	7
6	8	0	0	9
0	5	7	9	0

Start Edge	End Edge	Weight
0	1	2
1	2	3
0	3	6
1	4	5

Kruskal's Algorithm

Introduction

This algorithm is a Greedy Algorithm. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far. Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A minimum spanning tree (MST) or minimum weight spanning tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

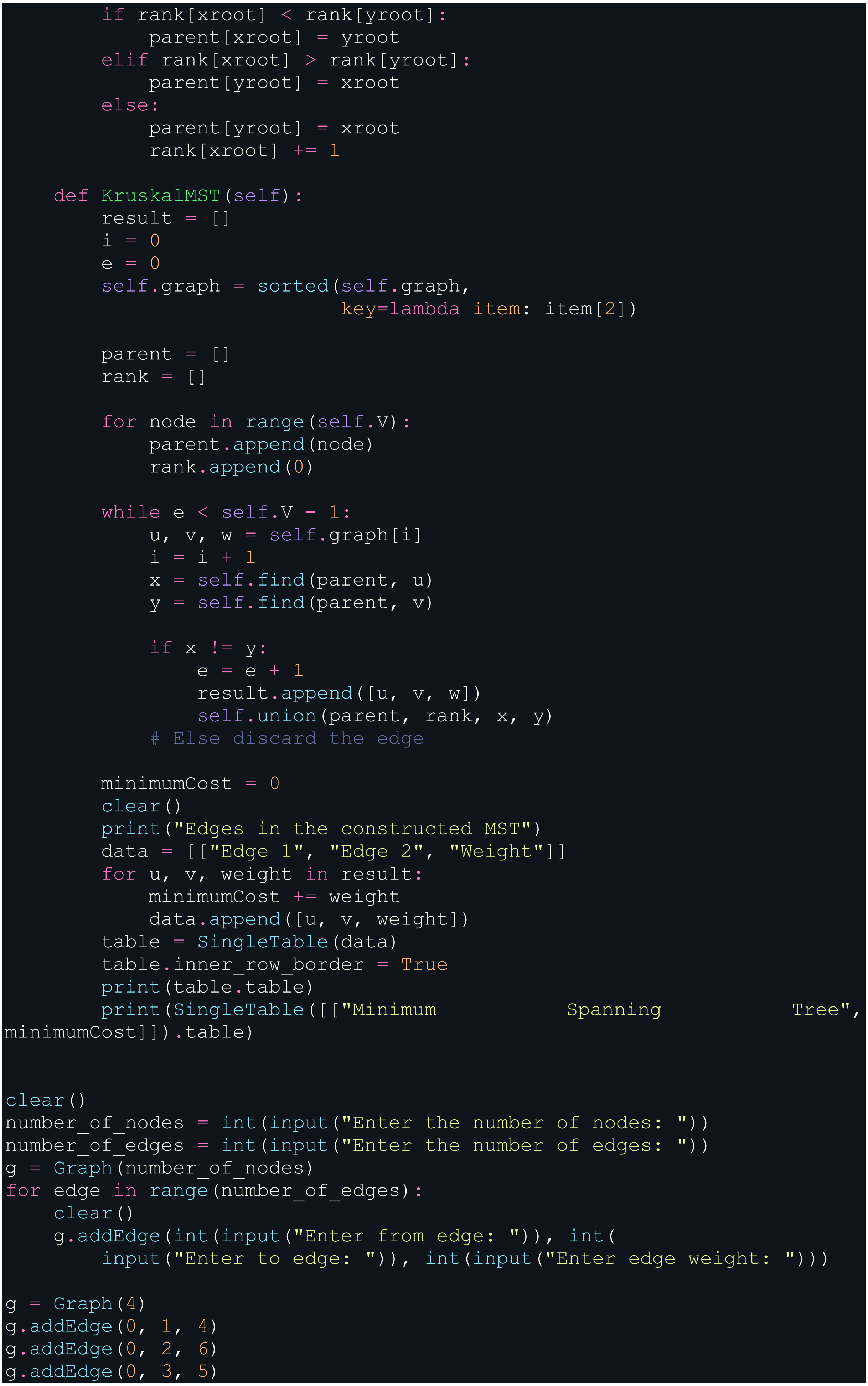
A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

Algorithm

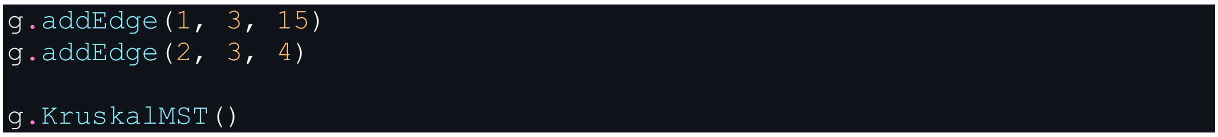
1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Code

```
from terminaltables import SingleTable
from common import clear


class Graph:

    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        if parent[i] == i:
            return i
        return self.find(parent, parent[i])

    def union(self, parent, rank, x, y):
        xroot = self.find(parent, x)
        yroot = self.find(parent, y)
```

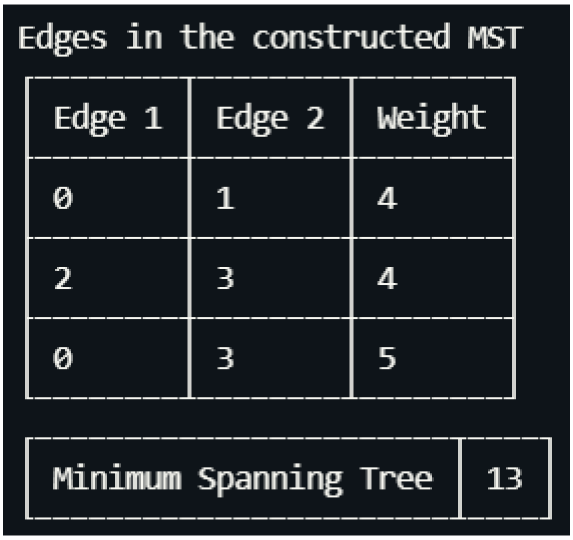


```
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1

    def KruskalMST(self):
        result = []
        i = 0
        e = 0
        self.graph = sorted(self.graph,
                            key=lambda item: item[2])

        parent = []
        rank = []

        for node in range(self.V):
            parent.append(node)
            rank.append(0)

        while e < self.V - 1:
            u, v, w = self.graph[i]
            i = i + 1
            x = self.find(parent, u)
            y = self.find(parent, v)

            if x != y:
                e = e + 1
                result.append([u, v, w])
                self.union(parent, rank, x, y)
            # Else discard the edge

        minimumCost = 0
        clear()
        print("Edges in the constructed MST")
        data = [["Edge 1", "Edge 2", "Weight"]]
        for u, v, weight in result:
            minimumCost += weight
            data.append([u, v, weight])
        table = SingleTable(data)
        table.inner_row_border = True
        print(table.table)
        print(SingleTable([["Minimum           Spanning           Tree",
minimumCost]]).table)


clear()
number_of_nodes = int(input("Enter the number of nodes: "))
number_of_edges = int(input("Enter the number of edges: "))
g = Graph(number_of_nodes)
for edge in range(number_of_edges):
    clear()
    g.addEdge(int(input("Enter from edge: ")), int(
        input("Enter to edge: ")), int(input("Enter edge weight: ")))

g = Graph(4)
g.addEdge(0, 1, 4)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
```



```
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

g.KruskalMST()
```

Result



```
Edges in the constructed MST
+--------+--------+--------+
| Edge 1 | Edge 2 | Weight |
+--------+--------+--------+
| 0      | 1      | 4      |
+--------+--------+--------+
| 2      | 3      | 4      |
+--------+--------+--------+
| 0      | 3      | 5      |
+--------+--------+--------+

+------------------------+----+
| Minimum Spanning Tree  | 13 |
+------------------------+----+
```

Bellman Ford Algorithm

Introduction

Given a graph and a source vertex src in the graph, find shortest paths from src to all vertices in the given graph. The graph may contain negative weight edges. Dijkstra doesn't work for Graphs with negative weight edges, Bellman-Ford works for such graphs.

Bellman-Ford is also simpler than Dijkstra and suits well for distributed systems. But the time complexity of Bellman-Ford is O(VE), which is more than Dijkstra.

Algorithm

Input: Graph and a source vertex src

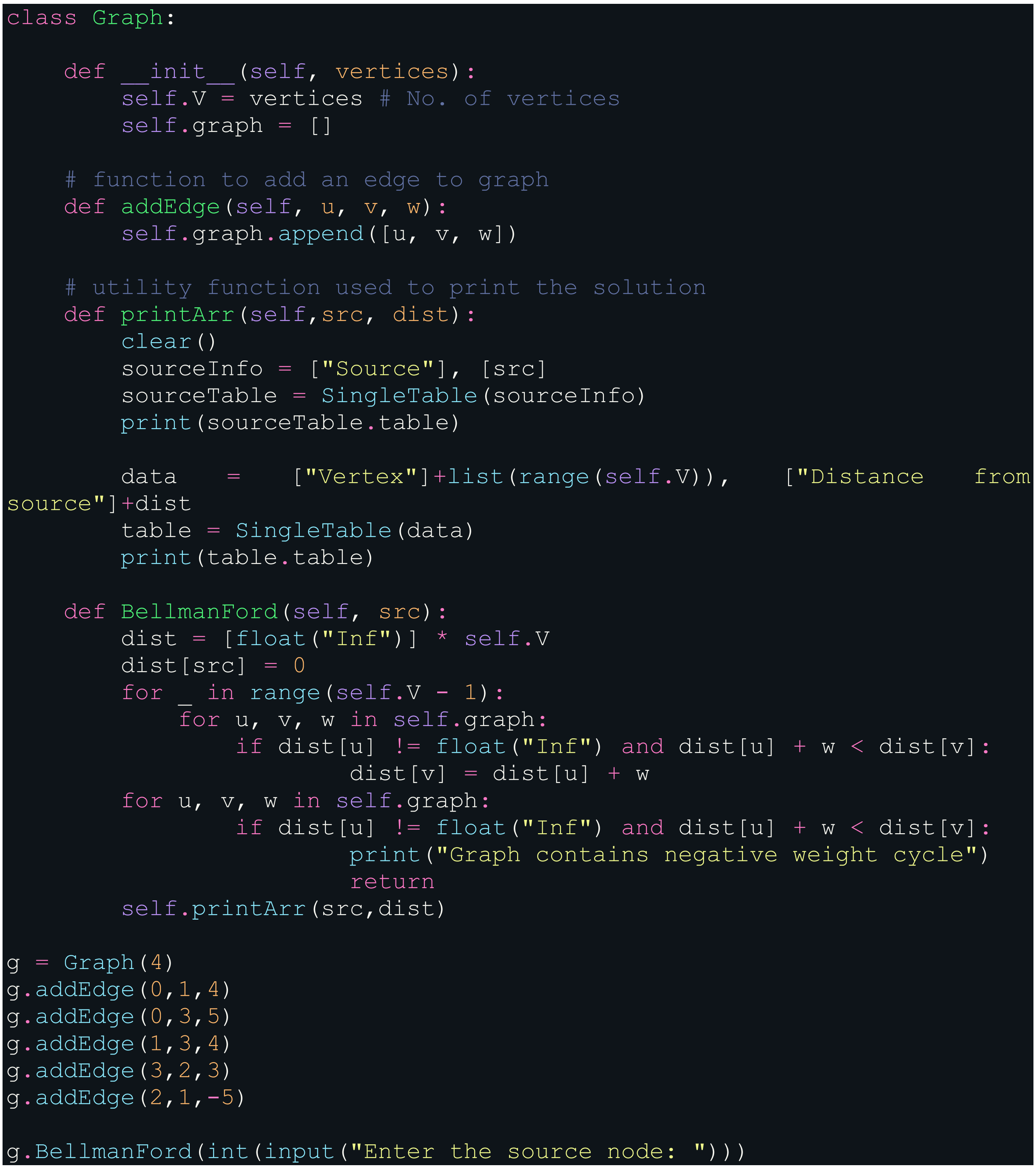
Output: Shortest distance to all vertices from src. If there is a negative weight cycle, then shortest distances are not calculated, negative weight cycle is reported.

1. This step initializes distances from the source to all vertices as infinite and distance to the source itself as 0. Create an array dist[] of size |V| with all values as infinite except dist[src] where src is source vertex.
2. This step calculates shortest distances. Do following |V|-1 times where |V| is the number of vertices in a given graph.
   Do following for each edge u-v
   If dist[v] > dist[u] + weight of edge uv, then update dist[v]
   dist[v] = dist[u] + weight of edge uv
3. This step reports if there is a negative weight cycle in the graph.
   Do following for each edge u-v
   If dist[v] > dist[u] + weight of edge uv, then "Graph contains negative weight cycle"

The idea of step 3 is, step 2 guarantees the shortest distances if the graph doesn't contain a negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle

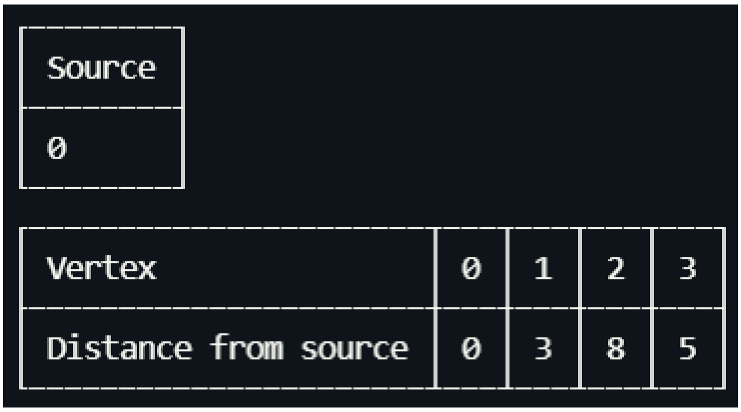
Code

```
from terminaltables import SingleTable
from common import clear
```



```
class Graph:

    def __init__(self, vertices):
        self.V = vertices # No. of vertices
        self.graph = []

    # function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    # utility function used to print the solution
    def printArr(self,src, dist):
        clear()
        sourceInfo = ["Source"], [src]
        sourceTable = SingleTable(sourceInfo)
        print(sourceTable.table)

        data = ["Vertex"]+list(range(self.V)), ["Distance from source"]+dist
        table = SingleTable(data)
        print(table.table)

    def BellmanFord(self, src):
        dist = [float("Inf")] * self.V
        dist[src] = 0
        for _ in range(self.V - 1):
            for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                        dist[v] = dist[u] + w
        for u, v, w in self.graph:
                if dist[u] != float("Inf") and dist[u] + w < dist[v]:
                        print("Graph contains negative weight cycle")
                        return
        self.printArr(src,dist)

g = Graph(4)
g.addEdge(0,1,4)
g.addEdge(0,3,5)
g.addEdge(1,3,4)
g.addEdge(3,2,3)
g.addEdge(2,1,-5)

g.BellmanFord(int(input("Enter the source node: ")))
```

Result



```
+--------+
| Source |
+--------+
| 0      |
+--------+

+----------------------+---+---+---+---+
| Vertex               | 0 | 1 | 2 | 3 |
+----------------------+---+---+---+---+
| Distance from source | 0 | 3 | 8 | 5 |
+----------------------+---+---+---+---+
```

CRC - Cyclic Redundancy Check

Introduction

A cyclic redundancy check (CRC) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error detection and correction.

CRC uses Generator Polynomial which is available on both sender and receiver sides. An example generator polynomial is of the form like $x^3 + 1$. This generator polynomial represents key **1001**. Another example is $x^2 + x$ that represents key **110**.

Algorithm

Sender Side

1. The task is to send a string data to the receiver side.
2. The sender sends a string.
3. First, this string is converted to a binary string “100010110101101001110”.
   The key is known to both the side sender and receiver.
4. This data is encoded using the CRC code using the key in the sender side.
5. This encoded data is sent to the receiver.
6. Receiver later decodes the encoded data string to verify whether there was any error or not.

Receiver Side

1. The receiver receives the encoded data string from the sender.
2. Receiver with the help of the same **key** decodes the data and finds out the remainder.
3. If the **remainder** is **zero** then it means there is **no error** in data sent by the sender to the receiver.

4. If the **remainder** comes out to be **non-zero** it means there was an **error**, a **negative acknowledgement** is sent to the sender. The sender then resends the data until the receiver receives correct data.

Code

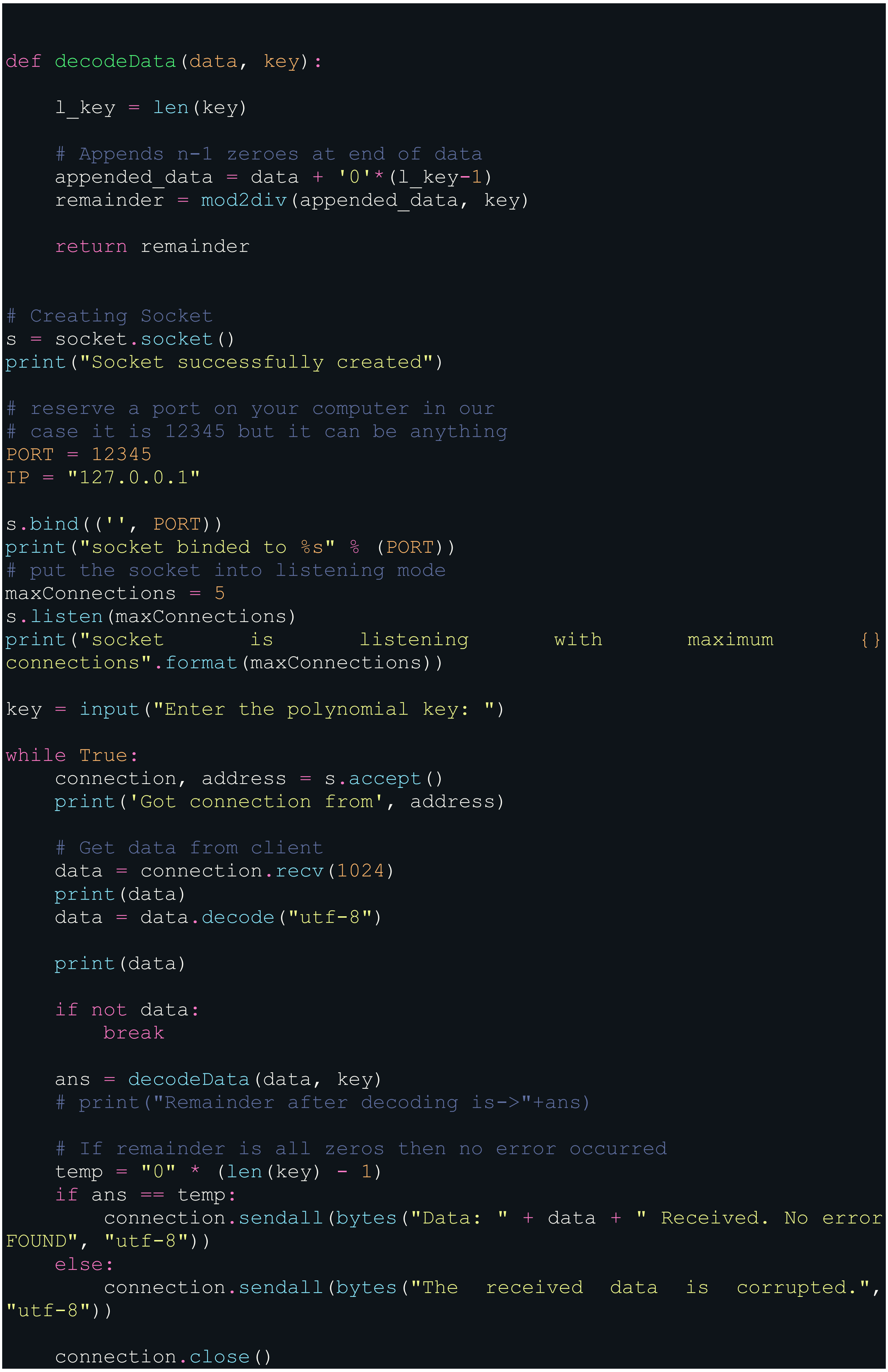
Sender Side

```
import socket

def xor(a, b):

    # initialize result
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)

def bitstring_to_bytes(s):
    v = int(s, 2)
    b = bytearray()
    while v:
        b.append(v & 0xff)
        v >>= 8
    return bytes(b[::-1])


# Performs Modulo-2 division
def mod2div(divident, divisor):
    pick = len(divisor)
    tmp = divident[0 : pick]
    while pick < len(divident):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + divident[pick]
        else:
            tmp = xor('0'*pick, tmp) + divident[pick]
        pick += 1
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)
    checkword = tmp
    return checkword

def encodeData(data, key):

    l_key = len(key)

    # Appends n-1 zeroes at end of data
    appended_data = data + '0'*(l_key-1)
    remainder = mod2div(appended_data, key)

    # Append remainder in the original data
    codeword = data + remainder
    return codeword
```

```
# Create a socket object
s = socket.socket()

# Define the IP & PORT on which you want to connect
PORT = 12345
IP = '127.0.0.1'

# connect to the server on local computer
s.connect((IP, PORT))

input_string = input("Enter data you want to send: ")
key = input("Enter the polynomial key: ")

data = (''.join(format(ord(x), 'b') for x in input_string))
print(data)

ans = encodeData(data,key)
print(ans)
s.sendall(bytes(ans, "utf-8"))


# receive data from the server
print (s.recv(1024).decode("utf-8"))

# close the connection
s.close()
```

Receiver Side

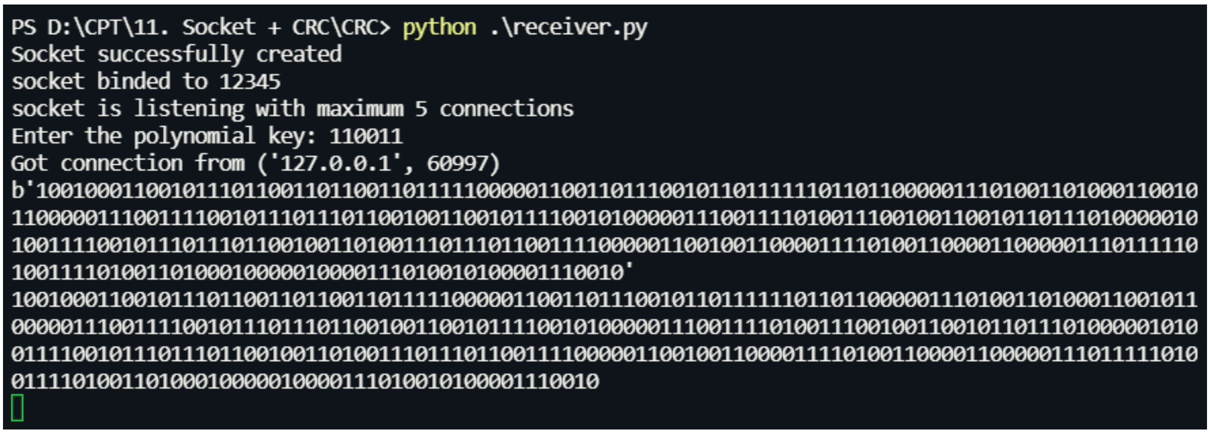
```
import socket

def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')
    return ''.join(result)


# Performs Modulo-2 division
def mod2div(divident, divisor):
    pick = len(divisor)
    tmp = divident[0: pick]

    while pick < len(divident):
        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + divident[pick]

        else:
            tmp = xor('0'*pick, tmp) + divident[pick]
        pick += 1
    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)

    checkword = tmp
    return checkword
```



```
def decodeData(data, key):

    l_key = len(key)

    # Appends n-1 zeroes at end of data
    appended_data = data + '0'*(l_key-1)
    remainder = mod2div(appended_data, key)

    return remainder


# Creating Socket
s = socket.socket()
print("Socket successfully created")

# reserve a port on your computer in our
# case it is 12345 but it can be anything
PORT = 12345
IP = "127.0.0.1"

s.bind(('', PORT))
print("socket binded to %s" % (PORT))
# put the socket into listening mode
maxConnections = 5
s.listen(maxConnections)
print("socket is listening with maximum {}
connections".format(maxConnections))

key = input("Enter the polynomial key: ")

while True:
    connection, address = s.accept()
    print('Got connection from', address)

    # Get data from client
    data = connection.recv(1024)
    print(data)
    data = data.decode("utf-8")

    print(data)

    if not data:
        break

    ans = decodeData(data, key)
    # print("Remainder after decoding is->"+ans)

    # If remainder is all zeros then no error occurred
    temp = "0" * (len(key) - 1)
    if ans == temp:
        connection.sendall(bytes("Data: " + data + " Received. No error
FOUND", "utf-8"))
    else:
        connection.sendall(bytes("The received data is corrupted.",
"utf-8"))

    connection.close()
```

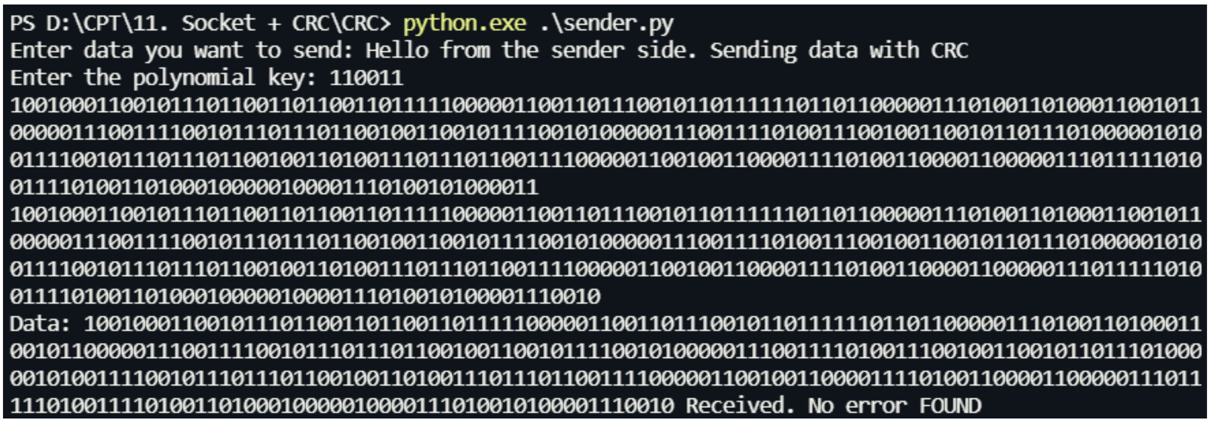
Result

Receiver Side



```
PS D:\CPT\11. Socket + CRC\CRC> python .\receiver.py
Socket successfully created
socket binded to 12345
socket is listening with maximum 5 connections
Enter the polynomial key: 110011
Got connection from ('127.0.0.1', 60997)
b'10010001100101110110011011001101111100000110011011100101101111110110110000011101001101000110010
1100000111001111001011101110110010011001011110010100000111001111010011100100110010110111010000010
1001111001011101110110010011010011101110110011110000011001001100001111010011000011000001110111110
10011110100110100010000010000111010010100001110010'
1001000110010111011001101100110111110000011001101110010110111111011011000001110100110100011001011
0000011100111100101110111011001001100101111001010000011100111101001110010011001011011101000001010
0111100101110111011001001101001110111011001111000001100100110000111101001100001100000111011111010
011110100110100010000010000111010010100001110010
```

Sender Side



```
PS D:\CPT\11. Socket + CRC\CRC> python.exe .\sender.py
Enter data you want to send: Hello from the sender side. Sending data with CRC
Enter the polynomial key: 110011
1001000110010111011001101100110111110000011001101110010110111111011011000001110100110100011001011
0000011100111100101110111011001001100101111001010000011100111101001110010011001011011101000001010
0111100101110111011001001101001110111011001111000001100100110000111101001100001100000111011111010
0111101001101000100000100001110100101000011
1001000110010111011001101100110111110000011001101110010110111111011011000001110100110100011001011
0000011100111100101110111011001001100101111001010000011100111101001110010011001011011101000001010
0111100101110111011001001101001110111011001111000001100100110000111101001100001100000111011111010
011110100110100010000010000111010010100001110010
Data: 1001000110010111011001101100110111110000011001101110010110111111011011000001110100110100011
0010110000011100111100101110111011001001100101111001010000011100111101001110010011001011011101000
0010100111100101110111011001001101001110111011001111000001100100110000111101001100001100000111011
111010011110100110100010000010000111010010100001110010 Received. No error FOUND
```

Sockets

Introduction

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while another socket reaches out to the other to form a connection. Server forms the listener socket while the client reaches out to the server.

They are the real backbones behind web browsing. In simpler terms there is a server and a client.

Code

Client Side

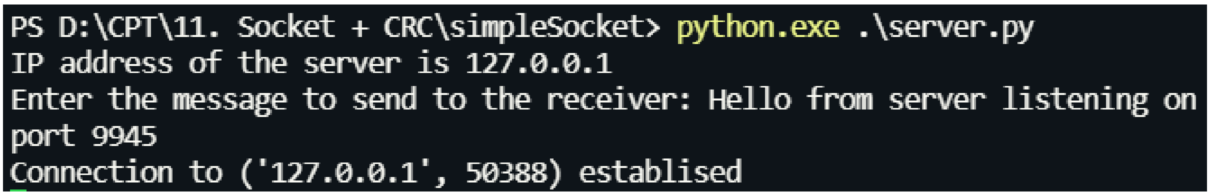
```
import socket

IP = "127.0.0.1"

s = socket.socket()
s.connect((IP, 9945))

received = ''

while True:
    msg = s.recv(4096)
    if (len(msg) <= 0):
        break

    received += msg.decode("utf-8")

print(received)
```

Server Side

```
import socket

IP = "127.0.0.1"

s = socket.socket()
s.bind((IP, 9945))

print("IP address of the server is {}".format(IP))


s.listen()
message = input("Enter the message to send to the receiver: ")

while True:
    connection, clientAddress = s.accept()
    print(f"Connection to {clientAddress} establised")
    connection.send(bytes(message, "utf-8"))
    connection.close()
```

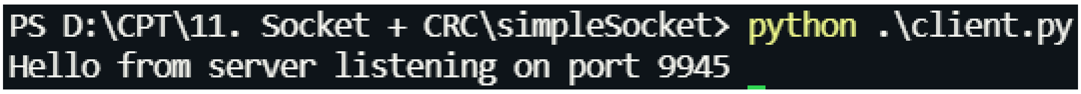
Result

Server Side



```
PS D:\CPT\11. Socket + CRC\simpleSocket> python.exe .\server.py
IP address of the server is 127.0.0.1
Enter the message to send to the receiver: Hello from server listening on
port 9945
Connection to ('127.0.0.1', 50388) establised
```

Client Side



```
PS D:\CPT\11. Socket + CRC\simpleSocket> python .\client.py
Hello from server listening on port 9945
```

Bit Stuffing

Data link layer is responsible for something called Framing, which is the division of stream of bits from network layer into manageable units (called frames). Frames could be of fixed size or variable size. In variable-size framing, we need a way to define the end of the frame and the beginning of the next frame.

Bit stuffing is the insertion of non information bits into data. Note that stuffed bits should not be confused with overhead bits. **Overhead bits** are non-data bits that are necessary for transmission (usually as part of headers, checksums etc.).

Applications of Bit Stuffing –

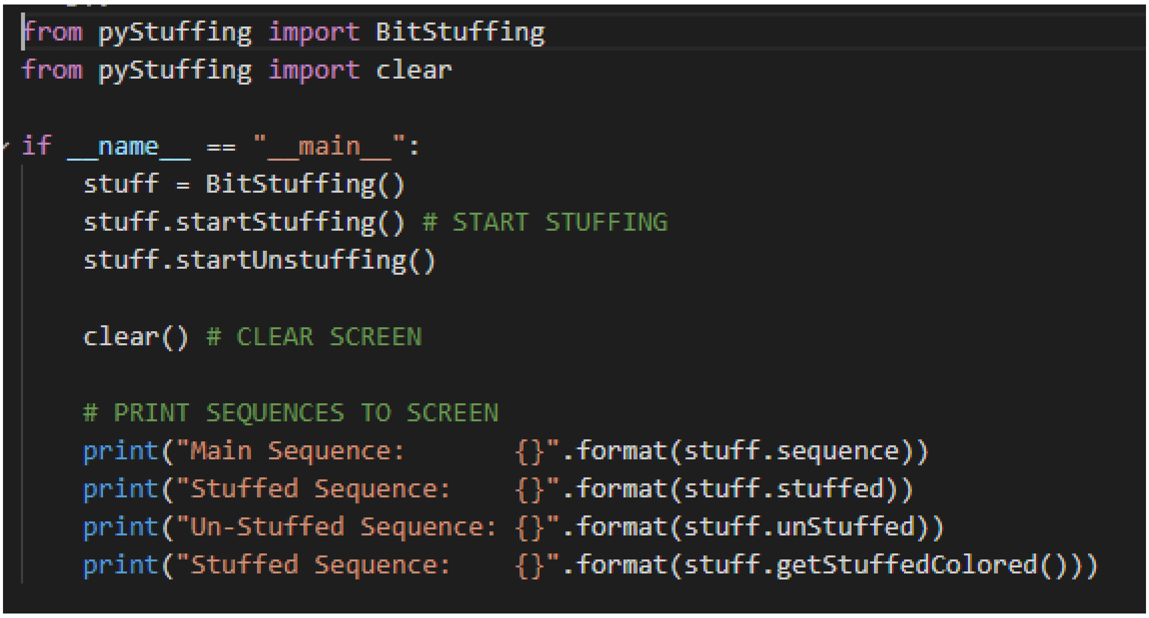
1. synchronize several channels before multiplexing
2. rate-match two single channels to each other
3. run length limited coding

Run length limited coding – To limit the number of consecutive bits of the same value(i.e., binary value) in the data to be transmitted. A bit of the opposite value is inserted after the maximum allowed number of consecutive bits.

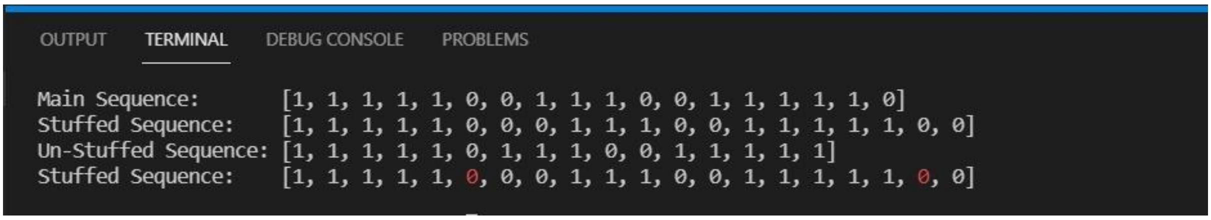
Bit stuffing technique does not ensure that the sent data is intact at the receiver side (i.e., not corrupted by transmission errors). It is merely a way to ensure that the transmission starts and ends at the correct places.

Disadvantages of Bit Stuffing –

The code rate is unpredictable; it depends on the data being transmitted.



```python
from pyStuffing import BitStuffing
from pyStuffing import clear

if __name__ == "__main__":
    stuff = BitStuffing()
    stuff.startStuffing() # START STUFFING
    stuff.startUnstuffing()

    clear() # CLEAR SCREEN

    # PRINT SEQUENCES TO SCREEN
    print("Main Sequence:       {}".format(stuff.sequence))
    print("Stuffed Sequence:    {}".format(stuff.stuffed))
    print("Un-Stuffed Sequence: {}".format(stuff.unStuffed))
    print("Stuffed Sequence:    {}".format(stuff.getStuffedColored()))
```

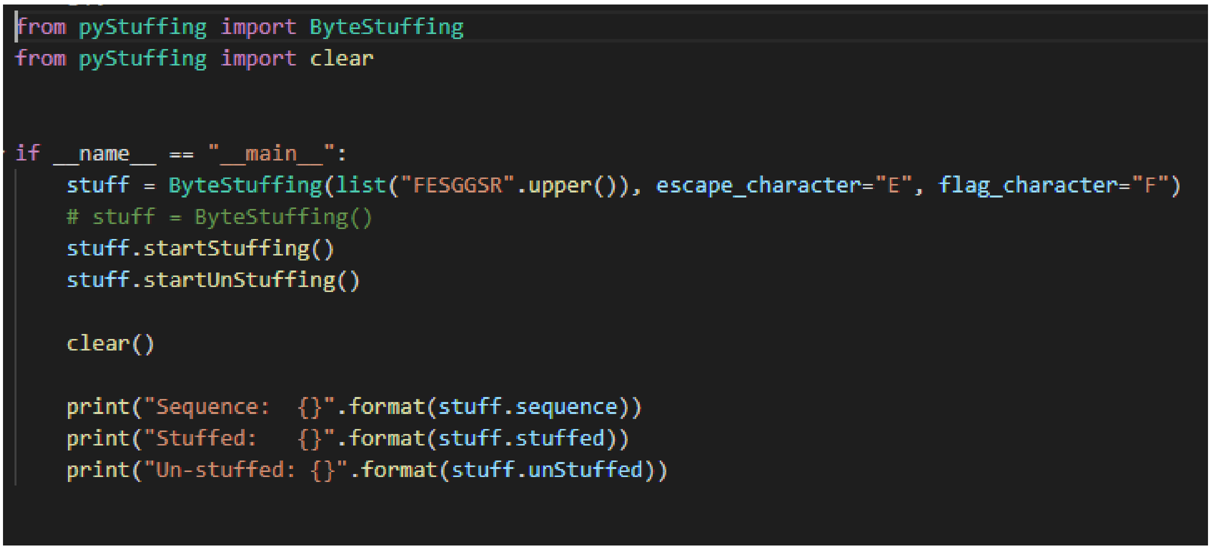


```
Main Sequence:       [1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0]
Stuffed Sequence:    [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0]
Un-Stuffed Sequence: [1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1]
Stuffed Sequence:    [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0]
```

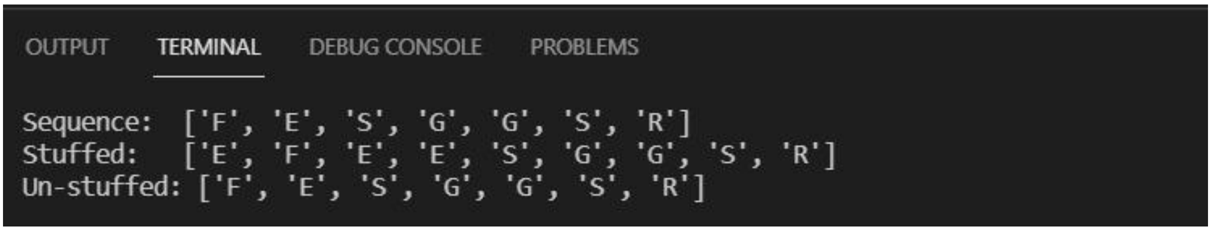
Byte Stuffing

A byte (usually escape character(ESC)), which has a predefined bit pattern is added to the data section of the frame when there is a character with the same pattern as the flag. Whenever the receiver encounters the ESC character, it removes from the data section and treats the next character as data, not a flag.

But the problem arises when the text contains one or more escape characters followed by a flag. To solve this problem, the escape characters that are part of the text are marked by another escape character i.e., if the escape character is part of the text, an extra one is added to show that the second one is part of the text.



```python
from pyStuffing import ByteStuffing
from pyStuffing import clear


if __name__ == "__main__":
    stuff = ByteStuffing(list("FESGGSR".upper()), escape_character="E", flag_character="F")
    # stuff = ByteStuffing()
    stuff.startStuffing()
    stuff.startUnStuffing()

    clear()

    print("Sequence:  {}".format(stuff.sequence))
    print("Stuffed:   {}".format(stuff.stuffed))
    print("Un-stuffed: {}".format(stuff.unStuffed))
```



```
Sequence:  ['F', 'E', 'S', 'G', 'G', 'S', 'R']
Stuffed:   ['E', 'F', 'E', 'E', 'S', 'G', 'G', 'S', 'R']
Un-stuffed: ['F', 'E', 'S', 'G', 'G', 'S', 'R']
```