Below is a list of 10 essential Windows network commands, along with their descriptions and illustrative examples. These commands are executed in the Windows Command Prompt (CMD) to troubleshoot, configure, or monitor network-related activities. Each command is accompanied by a brief explanation and an example to demonstrate its usage.

1. ping

Description: Tests the connectivity between the local computer and a remote host by sending ICMP echo requests and receiving replies. It helps verify if a remote device is reachable and measures round-trip time.

Example:

ping google.com

Output Illustration:

Pinging google.com [142.250.190.78] with 32 bytes of data:

Reply from 142.250.190.78: bytes=32 time=25ms TTL=117

Reply from 142.250.190.78: bytes=32 time=24ms TTL=117

Reply from 142.250.190.78: bytes=32 time=26ms TTL=117

Reply from 142.250.190.78: bytes=32 time=25ms TTL=117

Ping statistics for 142.250.190.78:

Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 24ms, Maximum = 26ms, Average = 25ms

Explanation: The command pings `google.com`, sending four packets and receiving replies, indicating the host is reachable with an average latency of 25ms.

2. ipconfig

Description: Displays the IP configuration details of all network adapters on the system, including IP address, subnet mask, and default gateway.

Example:

ipconfig

Output Illustration:

Windows IP Configuration

Ethernet adapter Ethernet:

Connection-specific DNS Suffix . : example.com

IPv4 Address. . . . . : 192.168.1.100

Subnet Mask . . . . . : 255.255.255.0

Default Gateway . . . . . : 192.168.1.1

Explanation: Shows the IP configuration for the Ethernet adapter, including the assigned IP address (`192.168.1.100`) and default gateway.

3. **ipconfig /all**

Description: Provides detailed IP configuration information, including DNS servers, MAC address, and DHCP status, for all network adapters.

Example:

ipconfig /all

Output Illustration:

Windows IP Configuration

Host Name . . . . . : DESKTOP-XYZ

Primary Dns Suffix . . . . . :

Node Type . . . . . : Hybrid

IP Routing Enabled. . . . . : No

WINS Proxy Enabled. . . . . : No

Ethernet adapter Ethernet:

Physical Address. . . . . : 00-14-22-01-23-45

DHCP Enabled. . . . . : Yes

IPv4 Address. . . . . : 192.168.1.100

Subnet Mask . . . . . : 255.255.255.0

Default Gateway . . . . . : 192.168.1.1

DNS Servers . . . . . : 8.8.8.8

Explanation: Displays detailed network adapter information, including the MAC address and DNS server.

4. tracert

Description: Traces the route packets take to a network host, showing each hop and latency. Useful for diagnosing where network delays or failures occur.

Example:

tracert google.com

Output Illustration:

Tracing route to google.com [142.250.190.78]

over a maximum of 30 hops:

1 2 ms 1 ms 1 ms 192.168.1.1

2 8 ms 7 ms 8 ms 10.0.0.1

3 15 ms 14 ms 15 ms isp-router.example.com [203.0.113.1]

4 25 ms 24 ms 25 ms 142.250.190.78

Trace complete.

Explanation: Shows the path from the local machine to `google.com`, listing each router (hop) and the time taken.

5. nslookup

Description: Queries DNS servers to resolve domain names to IP addresses or retrieve DNS records.

Example:

nslookup google.com

Output Illustration:

Server: dns.google

Address: 8.8.8.8

Non-authoritative answer:

Name: google.com

Address: 142.250.190.78

Explanation: Resolves the IP address of `google.com` using the DNS server `8.8.8.8`.

6. netstat

Description: Displays active network connections, listening ports, and network statistics.

Example:

netstat -an

Output Illustration:

Active Connections

Proto	Local Address	Foreign Address	State
TCP	192.168.1.100:135	0.0.0.0:0	LISTENING
TCP	192.168.1.100:49152	142.250.190.78:443	ESTABLISHED
UDP	192.168.1.100:123	*:*	

Explanation: Lists all active connections and listening ports, with `-a` showing all connections and `-n` displaying numerical addresses.

---

7. arp -a

Description: Displays the Address Resolution Protocol (ARP) cache, mapping IP addresses to MAC addresses for devices on the local network.

Example:

arp -a

Output Illustration:

Interface: 192.168.1.100 --- 0x2

Internet Address	Physical Address	Type
192.168.1.1	00-14-22-01-23-45	dynamic
192.168.1.101	00-16-17-22-33-44	dynamic

Explanation: Shows the IP-to-MAC address mappings for devices in the local network.

8. route print

Description: Displays the routing table, showing how packets are routed to different network destinations.

Example:

route print

Output Illustration:

IPv4 Route Table

==========

Active Routes:

Network Destination Netmask Gateway Interface Metric

0.0.0.0 0.0.0.0 192.168.1.1 192.168.1.100 25

192.168.1.0 255.255.255.0 On-link 192.168.1.100 281

Explanation: Lists routing rules, including the default gateway (`192.168.1.1`) for internet traffic.

9. netsh interface show interface

Description: Displays the status of network interfaces, including whether they are enabled or disabled.

Example:

netsh interface show interface

Output Illustration:

Admin State State Type Interface Name

Enabled Connected Dedicated Ethernet

Disabled Disconnected Dedicated Wi-Fi

Explanation: Shows the status of network interfaces, indicating the Ethernet interface is connected.

10. netsh wlan show profiles

Description: Lists all saved Wi-Fi profiles on the system, useful for managing wireless network connections.

Example:

netsh wlan show profiles

Output Illustration:

Profiles on interface Wi-Fi:

Group policy profiles (read only)

<None>

User profiles

All User Profile : HomeWiFi

All User Profile : OfficeWiFi

Explanation: Displays the names of saved Wi-Fi networks, such as `HomeWiFi`.

11. getmac

Description: Displays the MAC (Media Access Control) address of the network adapters on the system.

Example:

getmac

Output Illustration:

Physical Address Transport Name

==========
==========

00-14-22-01-23-45 \Device\Tcpip_{12345678-1234-1234-1234-1234567890AB}

Explanation: Lists the MAC address of the network adapter.

12. pathping

Description: Combines features of `ping` and `tracert` to provide detailed information about packet loss and latency at each hop.

Example:

pathping google.com

Output Illustration:

Tracing route to google.com [142.250.190.78]

over a maximum of 30 hops:

0 DESKTOP-XYZ [192.168.1.100]

1 192.168.1.1

2 isp-router.example.com [203.0.113.1]

3 142.250.190.78

Computing statistics for 75 seconds...

		Source to Here	This Node/Link	
Hop	RTT	Lost/Sent = Pct	Lost/Sent = Pct	Address
0				DESKTOP-XYZ [192.168.1.100]
		0/ 100 = 0%		
1	2ms	0/ 100 = 0%	0/ 100 = 0%	192.168.1.1
		0/ 100 = 0%		
2	15ms	0/ 100 = 0%	0/ 100 = 0%	isp-router.example.com [203.0.113.1]

0/ 100 = 0% |

3 25ms 0/ 100 = 0% 0/ 100 = 0% 142.250.190.78

Explanation: Shows the route to `google.com` and statistics on packet loss and latency at each hop.

Notes:

- How to Run: Open Command Prompt by typing `cmd` in the Windows search bar and pressing Enter. Type the commands as shown.

- Permissions: Some commands (e.g., `netsh`) may require administrative privileges. Right-click Command Prompt and select "Run as administrator."

- Real-Time Use: Commands like `ping`, `tracert`, and `pathping` rely on network connectivity, so results depend on the network environment.

Latency refers to the time it takes for data to travel from one point to another in a network, typically measured in milliseconds (ms). It represents the delay between sending a request and receiving a response. In networking, latency is a critical factor affecting the performance of applications, especially those requiring real-time interaction, such as online gaming, video calls, or web browsing.

Transit time is the duration it takes for a data packet to travel from the sender to the receiver (one-way) or for a request to travel to the destination and receive a response (round-trip). It is closely related to **latency**, which often refers to the round-trip time (RTT) in tools like ping. Transit time is influenced by:

- **Distance**: Physical distance between source and destination.
- **Network Hops**: Number of routers or devices the packet passes through.
- **Network Congestion**: Traffic load on the network.
- **Medium**: Type of connection (e.g., fiber, wireless, satellite).

BitStuffingClient.java

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class BitStuffingClient {
   public static void main(String[] args) throws IOException {
      Socket socket = new Socket("localhost", 6789);
      DataInputStream dis = new DataInputStream(socket.getInputStream());
      DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter data:");
      String data = sc.nextLine();
      int cnt = 0;
      String s = "";
      for (int i = 0; i < data.length(); i++) {
         char ch = data.charAt(i);
         if (ch == '1') {
            cnt++;
            if (cnt < 5)
               s += ch;
            else {
               s = s + ch + '0';
               cnt = 0;
            }
         } else {
            s += ch;
            cnt = 0;
         }
      }
      s = "01111110" + s + "01111110"
System.out.println("Data stuffed in client: " + s);
      System.out.println("Sending to server for unstuffing");
      dos.writeUTF(s);
      socket.close();
      sc.close();
   }
}
```

BitStuffingServer.java

```
import java.io.*;
import java.net.*;
public class BitStuffingServer {
   public static void main(String[] args) throws IOException {
      int port = 6789;
      if (args.length > 0) {
         try {
            port = Integer.parseInt(args[0]);
         } catch (NumberFormatException e) {
            System.out.println("Invalid port number. Using default port 6789.");
         }
      }
      ServerSocket skt = new ServerSocket(port);
      System.out.println("Server started on port: " + port);
      Socket socket = skt.accept();
      DataInputStream dis = new DataInputStream(socket.getInputStream());
      DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
      String s = dis.readUTF();
      System.out.println("Stuffed data from client: " + s);
      System.out.print("Unstuffed data: ");
      int cnt = 0;
      for (int i = 8; i < s.length() - 8; i++) {
         char ch = s.charAt(i);
         if (ch == '1') {
            cnt++;
            System.out.print(ch);
            if (cnt == 5) {
               i++;
               cnt = 0;
            }
         } else {
            System.out.print(ch);
            cnt = 0;
         }
      }
      System.out.println();
      socket.close();
      skt.close();
   }
}
```

SenderByte.java

```
import java.io.*;
import java.util.*;
import java.net.*;
public class SenderByte {
   public static void main(String[] args) throws IOException {
      InetAddress ip = InetAddress.getLocalHost();
      int port = 45678;
      Scanner sc = new Scanner(System.in);
      Socket s = new Socket(ip, port);
      DataInputStream dis = new DataInputStream(s.getInputStream());
      DataOutputStream dos = new DataOutputStream(s.getOutputStream());
      while (true) {
         System.out.println("Enter the Message to be Sent:");
         String data = sc.nextLine();
         String res = "";
         // Add frame delimiters
         data = 'F' + data + 'F';
         for (int i = 0; i < data.length(); i++) {
            // Stuff with 'E' if 'F' or 'E' is found in the data
            if ((data.charAt(i) == 'F' && i != 0 && i != data.length() - 1) || data.charAt(i) ==
'E')
               res += "E" + data.charAt(i);
            else
               res += data.charAt(i);
         }
         System.out.println("The data being sent (with byte stuffed) is: " + res);
         dos.writeUTF(res);
         System.out.println("Sending Message...");
         if (dis.readUTF().equals("success"))
            System.out.println("Thanks for the Feedback Server!!");
         dos.writeUTF("bye");
         break;
      }
      s.close();
      dis.close();
      dos.close();
      sc.close();
   }
}
```

RecieverByte.java

```
import java.io.*;
import java.net.*;
public class ReceiverByte {
   public static void main(String[] args) throws IOException {
      ServerSocket servsock = new ServerSocket(45678);
      Socket socket = servsock.accept();
      DataInputStream dis = new DataInputStream(socket.getInputStream());
      DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
      while (true) {
         String out = "";
         String res = dis.readUTF();
         System.out.println("Message Received... Successfully!!!");
         System.out.println("The Stuffed Message is: " + res);
         for (int i = 1; i < res.length() - 1; i++) {
            if (res.charAt(i) == 'E') {
               // Handle 'EE' → E
               if (res.charAt(i + 1) == 'E') {
                  out += 'E';
                  i++;
               }
               // Handle 'EF' → F
               else if (res.charAt(i + 1) == 'F') {
                  out += 'F';
                  i++;
               }
            } else {
               out += res.charAt(i);
            }
         }
         System.out.println("The Destuffed Message is: " + out);
         dos.writeUTF("success");
         String ch = dis.readUTF();
         if (ch.equals("bye")) {
            System.out.println("Messaging is over... EXITING");
            break;
         }
      }
      socket.close();
      dis.close();
      dos.close();
      servsock.close();
   }
}
```

CRC code:

```
import java.util.Scanner;
class CRC {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      // Accept the input data bits
      System.out.println("Enter the size of the data:");
      int dataSize = scan.nextInt();
      int[] data = new int[dataSize];
      System.out.println("Enter the data bits (0 or 1):");
      for (int i = 0; i < dataSize; i++) {
         System.out.print("Enter bit number " + (i + 1) + ": ");
         data[i] = scan.nextInt();
      }
      // Accept the divisor bits
      System.out.println("Enter the size of the divisor:");
      int divisorSize = scan.nextInt();
      int[] divisor = new int[divisorSize];
      System.out.println("Enter the divisor bits (0 or 1):");
      for (int i = 0; i < divisorSize; i++) {
         System.out.print("Enter bit number " + (i + 1) + ": ");
         divisor[i] = scan.nextInt();
      }
      int[] remainder = divide(data, divisor);
      System.out.println("\nThe CRC code generated is:");
      for (int bit : data) System.out.print(bit);
      for (int bit : remainder) System.out.print(bit);
      System.out.println();
      // Create and display the transmitted data (data + CRC)
      int[] sentData = new int[data.length + remainder.length];
      System.arraycopy(data, 0, sentData, 0, data.length);
      System.arraycopy(remainder, 0, sentData, data.length, remainder.length);
      System.out.println("Simulated sent data:");
      for (int bit : sentData) System.out.print(bit);
      System.out.println();
      // Check received data for errors
      receive(sentData, divisor);
      scan.close();
   }
   static int[] divide(int[] oldData, int[] divisor) {
      int[] data = new int[oldData.length + divisor.length - 1];
      System.arraycopy(oldData, 0, data, 0, oldData.length);
      for (int i = 0; i < oldData.length; i++) {
         if (data[i] == 1) {
            for (int j = 0; j < divisor.length; j++) {
               data[i + j] ^= divisor[j]; // XOR operation
```

```
                }
            }
        }
        int[] remainder = new int[divisor.length - 1];
        System.arraycopy(data, data.length - divisor.length + 1, remainder, 0, divisor.length - 1);
        return remainder;
    }
    static void receive(int[] data, int[] divisor) {
        int[] remainder = divide(data, divisor);
        boolean hasError = false;

        for (int bit : remainder) {
            if (bit != 0) {
                hasError = true;
                break;
            }
        }
        if (hasError)
            System.out.println("There is an error in the received data.");
        else
            System.out.println("Data was received without any error.");
    }
}
```

HammingCode.java

```
import java.util.Scanner;
class HammingCode {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] d = new int[7];
        System.out.println("Enter the 7-bit data code:");
        for (int i = 0; i < 7; i++) {
            d[i] = sc.nextInt();
        }
        // Calculate parity bits
        int[] p = new int[4];
        p[0] = d[0] ^ d[1] ^ d[3] ^ d[4] ^ d[6];
        p[1] = d[0] ^ d[2] ^ d[3] ^ d[5] ^ d[6];
        p[2] = d[1] ^ d[2] ^ d[3];
        p[3] = d[4] ^ d[5] ^ d[6];
        // Construct codeword
        int[] c = new int[11];
        c[0] = p[0];
        c[1] = p[1];
        c[2] = d[0];
        c[3] = p[2];
        c[4] = d[1];
        c[5] = d[2];
        c[6] = d[3];
        c[7] = p[3];
        c[8] = d[4];
        c[9] = d[5];
        c[10] = d[6];
        System.out.println("Complete Code Word is:");
        for (int i = 0; i < 11; i++) {
            System.out.print(c[i] + " ");
        }
        // Input received code (may contain error)
        System.out.println("\n\nEnter the received 11-bit code:");
        int[] r = new int[11];
        for (int i = 0; i < 11; i++) {
            r[i] = sc.nextInt();
        }

        // Extract parity and data bits from received code
        int[] pr = new int[4];
        int[] rd = new int[7];
        pr[0] = r[0];
        pr[1] = r[1];
        rd[0] = r[2];
```

```
        pr[2] = r[3];
        rd[1] = r[4];
        rd[2] = r[5];
        rd[3] = r[6];
        pr[3] = r[7];
        rd[4] = r[8];
        rd[5] = r[9];
        rd[6] = r[10];
        // Calculate syndrome bits
        int[] s = new int[4];
        s[0] = pr[0] ^ rd[0] ^ rd[1] ^ rd[3] ^ rd[4] ^ rd[6];
        s[1] = pr[1] ^ rd[0] ^ rd[2] ^ rd[3] ^ rd[5] ^ rd[6];
        s[2] = pr[2] ^ rd[1] ^ rd[2] ^ rd[3];
        s[3] = pr[3] ^ rd[4] ^ rd[5] ^ rd[6];
        // Convert syndrome bits to decimal
        int dec = (s[0] * 1) + (s[1] * 2) + (s[2] * 4) + (s[3] * 8);
        if (dec == 0) {
            System.out.println("No error detected in the received code.");
        } else {
            System.out.println("Error detected at position: " + dec);
            r[dec - 1] = (r[dec - 1] == 0) ? 1 : 0; // Correct the bit
        }
        System.out.println("Corrected Code Word is:");
        for (int i = 0; i < 11; i++) {
            System.out.print(r[i] + " ");
        }
        System.out.println();
        sc.close();
    }
}
```

Tcpserver.java:

```
import java.net.*;
import java.io.*;
public class tcpserver {
   public static void main(String[] args) {
      ServerSocket server = null;
      Socket client = null;
      BufferedReader inFromClient = null;
      BufferedReader inFromServer = null;
      PrintWriter outToClient = null;
      try {
         server = new ServerSocket(9222);
         System.out.println("Server started. Waiting for client connection...");
         client = server.accept();
         System.out.println("Client connected.");
         inFromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
         inFromServer = new BufferedReader(new InputStreamReader(System.in));
         outToClient = new PrintWriter(client.getOutputStream(), true);
         String line;
         do {
            line = inFromClient.readLine();
            if (line == null) break;
            System.out.println("Client: " + line);
            System.out.print("Server: ");
            String response = inFromServer.readLine();
            outToClient.println(response);
            if (response.equalsIgnoreCase("quit"))
               break;
         } while (true);
         inFromClient.close();
         outToClient.close();
         client.close();
         server.close();
         System.out.println("Connection closed.");
      } catch (IOException e) {
         System.out.println("Error: " + e.getMessage());
      }
   }
}
```

Tcpclient.java:

```
import java.net.*;
import java.io.*;
public class tcpclient {
   public static void main(String[] args) {
      Socket client = null;
      BufferedReader inFromUser = null;
      BufferedReader inFromServer = null;
      PrintWriter outToServer = null;
      try {
         client = new Socket("127.0.0.1", 9222);
         System.out.println("Connected to server.");

         inFromUser = new BufferedReader(new InputStreamReader(System.in));
         inFromServer = new BufferedReader(new
InputStreamReader(client.getInputStream()));
         outToServer = new PrintWriter(client.getOutputStream(), true);

         String line;
         do {
            System.out.print("Client: ");
            line = inFromUser.readLine();
            outToServer.println(line);
            String response = inFromServer.readLine();
            if (response == null) break;
            System.out.println("Server: " + response);
            if (line.equalsIgnoreCase("quit"))
               break;
         } while (true);
         inFromUser.close();
         inFromServer.close();
         outToServer.close();
         client.close();
         System.out.println("Connection closed.");
      } catch (IOException e) {
         System.out.println("Socket Closed! Message passing is over.");
      }
   }
}
```

UDPServer.java:

```
import java.net.*;
import java.io.*;
public class UDPServer {
   public static void main(String[] args) {
      DatagramSocket socket = null;
      BufferedReader br = null;
      try {
         socket = new DatagramSocket(9222); // Server on port 9222
         byte[] receiveData = new byte[1024];
         byte[] sendData;
         br = new BufferedReader(new InputStreamReader(System.in));
         System.out.println("UDP Server started...");
         while (true) {
            // Receive message from client
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            socket.receive(receivePacket);
            String clientMsg = new String(receivePacket.getData(), 0,
receivePacket.getLength());
            System.out.println("Client: " + clientMsg);
            if (clientMsg.equalsIgnoreCase("quit")) {
               System.out.println("Client exited. Closing server...");
               break;
            }
            // Send reply to client
            System.out.print("Server: ");
            String reply = br.readLine();
            sendData = reply.getBytes();
            InetAddress clientAddr = receivePacket.getAddress();
            int clientPort = receivePacket.getPort();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
clientAddr, clientPort);
            socket.send(sendPacket);
            if (reply.equalsIgnoreCase("quit")) {
               System.out.println("Server exiting...");
               break;
            }
         }
      } catch (Exception e) {
         System.out.println("Error: " + e.getMessage());
      } finally {
         if (socket != null) socket.close();
         System.out.println("Socket closed.");
      }
   }   }
```

UDPClient.java:

```
import java.net.*;
import java.io.*;
public class UDPClient {
   public static void main(String[] args) {
      DatagramSocket socket = null;
      try {
         socket = new DatagramSocket();
         InetAddress serverAddr = InetAddress.getByName("127.0.0.1");
         int serverPort = 9222;
         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
         byte[] sendData;
         byte[] receiveData = new byte[1024];
         System.out.println("UDP Client started...");
         while (true) {
            // Send to server
            System.out.print("Client: ");
            String msg = br.readLine();
            sendData = msg.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
serverAddr, serverPort);
            socket.send(sendPacket);
            if (msg.equalsIgnoreCase("quit")) {
               System.out.println("Client exiting...");
               break;
            }

            // Receive reply from server
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
            socket.receive(receivePacket);
            String reply = new String(receivePacket.getData(), 0, receivePacket.getLength());
            System.out.println("Server: " + reply);
            if (reply.equalsIgnoreCase("quit")) {
               System.out.println("Server ended chat.");
               break;
            }
         }
      } catch (Exception e) {
         System.out.println("Error: " + e.getMessage());
      } finally {
         if (socket != null) socket.close();
         System.out.println("Socket closed.");
      }
   }   }
```

Dijkstrarouting.java:

```
import java.util.Scanner;
public class Dijkstrarouting {
   static final int INF = 9999; // Represents infinite distance
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter number of nodes: ");
      int n = sc.nextInt();
      int[][] graph = new int[n][n];
      System.out.println("Enter adjacency matrix (0 means no direct link):");
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            graph[i][j] = sc.nextInt();
         }
      }
      // Run Dijkstra for each node as source
      for (int src = 0; src < n; src++) {
         dijkstra(graph, n, src);
      }
      sc.close();
   }
   // Dijkstra's algorithm for one source node
   static void dijkstra(int[][] graph, int n, int src) {
      int[] dist = new int[n];
      boolean[] visited = new boolean[n];
      int[] parent = new int[n];

      // Initialize arrays
      for (int i = 0; i < n; i++) {
         dist[i] = INF;
         visited[i] = false;
         parent[i] = -1;
      }
      dist[src] = 0;
      for (int count = 0; count < n - 1; count++) {
         int u = minDistance(dist, visited, n);
         if (u == -1) break;
         visited[u] = true;
         // Update distance values of adjacent nodes
         for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] != 0 && dist[u] != INF
                  && dist[u] + graph[u][v] < dist[v]) {
               dist[v] = dist[u] + graph[u][v];
               parent[v] = u;
            }
         }
```

```
        }
        printRoutingTable(src, n, dist, parent);
    }
    // Find the node with the minimum distance not yet visited
    static int minDistance(int[] dist, boolean[] visited, int n) {
        int min = INF, minIndex = -1;
        for (int i = 0; i < n; i++) {
            if (!visited[i] && dist[i] < min) {
                min = dist[i];
                minIndex = i;
            }
        }
        return minIndex;
    }
    // Print routing table for one source node
    static void printRoutingTable(int src, int n, int[] dist, int[] parent) {
        System.out.println("\nRouting Table for Node " + src + ":");
        System.out.println("Destination\tNextHop\tDistance");
        for (int i = 0; i < n; i++) {
            if (i == src) continue;
            int nextHop = findNextHop(src, i, parent);
            String nh = (nextHop == -1) ? "-" : Integer.toString(nextHop);
            String d = (dist[i] == INF) ? "INF" : Integer.toString(dist[i]);
            System.out.println(i + "\t\t" + nh + "\t\t" + d);
        }
    }

    // Find the next hop from src to destination using parent array
    static int findNextHop(int src, int dest, int[] parent) {
        if (dest == src) return src;
        if (parent[dest] == -1) return -1;
        int hop = dest;
        while (parent[hop] != -1 && parent[hop] != src) {
            hop = parent[hop];
        }
        if (parent[hop] == -1 && parent[dest] != src) return -1;
        return hop;
    }
}
```