

```
1.txt
```

```
gcc DVR.c -o DVR.exe
./DVR.exe
```

```
#include <stdio.h>

#define INF 999

int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    int cost[10][10];
    printf("Enter cost matrix (use 999 for no link):\n");

    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            scanf("%d",&cost[i][j]);

    int dist[10][10], nextHop[10][10];

    // Initialize distance and next-hop tables
    for(int i=0;i<n;i++) {
        for(int j=0;j<n;j++) {
            dist[i][j] = cost[i][j];
            if(i == j)
                nextHop[i][j] = i;
            else if(cost[i][j] != INF)
                nextHop[i][j] = j;
            else
                nextHop[i][j] = -1;
        }
    }

    // Run Bellman-Ford updates
    int updated;
    do {
        updated = 0;
        for(int i=0;i<n;i++) {
            for(int j=0;j<n;j++) {
                for(int k=0;k<n;k++) {
                    if(dist[i][k] + dist[k][j] < dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                        nextHop[i][j] = nextHop[i][k];
                        updated = 1;
                    }
                }
            }
        }
    } while(updated);

    // Print final routing table
    for(int i=0;i<n;i++) {
        printf("\nRouting table for node %d:\n", i);
        printf("Dest\tCost\tNextHop\n");
        for(int j=0;j<n;j++) {
            printf("%d\t%d\t%d\n", j, dist[i][j], nextHop[i][j]);
        }
    }

    return 0;
}
```

```
dijkstra -----
```

```

#include <stdio.h>

#define INF 999

int main() {
    int n;
    printf("Enter number of nodes: ");
    scanf("%d", &n);

    int cost[10][10];
    printf("Enter cost matrix (use 999 for no link):\n");

    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            scanf("%d", &cost[i][j]);

    int visited[10] = {0};
    int dist[10];
    int source;

    printf("Enter source node: ");
    scanf("%d", &source);

    // Initialize distances
    for(int i = 0; i < n; i++)
        dist[i] = cost[source][i];

    visited[source] = 1;
    dist[source] = 0;

    // Dijkstra loop
    for(int count = 1; count < n; count++) {
        int min = INF, u = -1;

        // Pick nearest unvisited node
        for(int i = 0; i < n; i++) {
            if(!visited[i] && dist[i] < min) {
                min = dist[i];
                u = i;
            }
        }

        visited[u] = 1;

        // Relax distances
        for(int v = 0; v < n; v++) {
            if(!visited[v] && dist[u] + cost[u][v] < dist[v]) {
                dist[v] = dist[u] + cost[u][v];
            }
        }
    }

    // Print final distances
    printf("\nShortest distances from source %d:\n", source);
    for(int i = 0; i < n; i++)
        printf("Node %d: %d\n", i, dist[i]);
}

return 0;
}

-----
#include <stdio.h>
#include <limits.h>
#include <stdbool.h>

#define V 5 // Number of vertices (routers in the network)

// Function to find the vertex with the minimum distance
int minDistance(int dist[], bool visited[]) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)

```

```

        if (!visited[v] && dist[v] <= min) {
            min = dist[v];
            min_index = v;
        }
    return min_index;
}

// Function to print the shortest path distances
void printSolution(int dist[], int parent[], int src) {
    printf("\nShortest paths from Router %c:\n", src + 'A');
    printf("Router\tDistance\tPath");
    printf("\n-----\n");

    for (int i = 0; i < V; i++) {
        printf("%c\t%d\t", i + 'A', dist[i]);
        int path[10], count = 0;
        int j = i;

        while (j != -1) {
            path[count++] = j;
            j = parent[j];
        }

        for (int k = count - 1; k >= 0; k--) {
            printf("%c", path[k] + 'A');
            if (k != 0)
                printf(" -> ");
        }
        printf("\n");
    }
}

// Dijkstra's Algorithm
void dijkstra(int graph[V][V], int src) {
    int dist[V];
    bool visited[V];
    int parent[V];

    for (int i = 0; i < V; i++) {
        dist[i] = INT_MAX;
        visited[i] = false;
        parent[i] = -1;
    }
    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, visited);
        visited[u] = true;

        for (int v = 0; v < V; v++) {
            if (!visited[v] && graph[u][v] && dist[u] != INT_MAX &&
                dist[u] + graph[u][v] < dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
            }
        }
    }

    printSolution(dist, parent, src);
}

int main() {
    int graph[V][V] = {
        {0, 4, 2, 0, 0},
        {4, 0, 1, 5, 0},
        {2, 1, 0, 8, 10},
        {0, 5, 8, 0, 2},
        {0, 0, 10, 2, 0}
    };

    char start;
    printf("Enter starting router (A-E): ");
}

```

```

scanf(" %c", &start);

int src = start - 'A';
if (src < 0 || src >= V) {
    printf("Invalid router!\n");
    return 1;
}

dijkstra(graph, src);
return 0;
}

Enter starting router (A-E): A

Shortest paths from Router A:
Router  Distance      Path
-----
A      0            A
B      3            A -> C -> B
C      2            A -> C
D      8            A -> C -> B -> D
E      10           A -> C -> B -> D -> E

leaky bucket-----

#include <stdio.h>

int main() {
    int bucketSize, outRate, n, incoming;

    printf("Enter bucket size: ");
    scanf("%d", &bucketSize);

    printf("Enter output rate: ");
    scanf("%d", &outRate);

    printf("Enter number of incoming packets: ");
    scanf("%d", &n);

    int stored = 0;

    for (int i = 0; i < n; i++) {
        printf("\nEnter incoming packet size: ");
        scanf("%d", &incoming);

        if (incoming + stored > bucketSize) {
            printf("Bucket overflow! Dropped packets = %d\n",
                   incoming + stored - bucketSize);
            stored = bucketSize;
        } else {
            stored += incoming;
            printf("Stored = %d\n", stored);
        }

        // Send packets out
        int sent = (stored < outRate) ? stored : outRate;
        stored -= sent;
        printf("Sent = %d | Remaining in bucket = %d\n", sent, stored);
    }

    return 0;
}

input :      bucket = 10
outRate = 4
packets = 3
incoming = [6, 5, 3]
output
Stored = 6

```

```

Sent = 4 | Remaining = 2
Bucket overflow! Dropped = 5 + 2 - 10 = -?
...
-----
#include <stdio.h>

int main() {
    int bucket_size, output_rate, incoming, bucket = 0, time = 1;

    printf("Enter bucket capacity: ");
    scanf("%d", &bucket_size);

    printf("Enter output rate: ");
    scanf("%d", &output_rate);

    // Repeat until user enters 0 packets (to stop)
    while (1) {
        printf("\nTime %d sec - Enter incoming packets (0 to stop): ", time);
        scanf("%d", &incoming);

        if (incoming == 0)
            break;

        // Add incoming packets
        bucket += incoming;

        // Check for overflow
        if (bucket > bucket_size) {
            printf("Bucket overflow! Packets dropped: %d\n", bucket - bucket_size);
            bucket = bucket_size; // retain only what fits
        }

        // Send out packets (leak)
        int sent = (bucket >= output_rate) ? output_rate : bucket;
        bucket -= sent;

        printf("Packets sent: %d | Packets left in bucket: %d\n", sent, bucket);
        time++;
    }

    // Leak remaining packets if any
    while (bucket > 0) {
        int sent = (bucket >= output_rate) ? output_rate : bucket;
        bucket -= sent;
        printf("\nLeaking... Packets sent: %d | Packets left: %d", sent, bucket);
    }

    printf("\n\nTransmission complete!\n");
    return 0;
}

input
Enter bucket capacity: 10
Enter output rate: 4

Time 1 sec - Enter incoming packets (0 to stop): 6
Time 2 sec - Enter incoming packets (0 to stop): 5
Time 3 sec - Enter incoming packets (0 to stop): 3
Time 4 sec - Enter incoming packets (0 to stop): 0

```

```

DOMAIN NAME SYSTEM DNS  -----
#include <stdio.h>
#include <string.h>

int main() {
    int n;
    printf("Enter number of DNS records: ");

```

```

scanf("%d", &n);

char domain[20][50], ip[20][50], query[50];

// Input DNS table
for(int i = 0; i < n; i++) {
    printf("Enter domain and IP: ");
    scanf("%s %s", domain[i], ip[i]);
}

// Query
printf("Enter domain to search: ");
scanf("%s", query);

int found = 0;

// Search for domain
for(int i = 0; i < n; i++) {
    if(strcmp(query, domain[i]) == 0) {
        printf("IP address: %s\n", ip[i]);
        found = 1;
        break;
    }
}

if(!found)
    printf("Domain not found in DNS table\n");

return 0;
}

input
Enter number of DNS records: 3
Enter domain and IP:
google.com 8.8.8.8
yahoo.com 1.1.1.1
example.com 93.184.216.34

Enter domain to search: yahoo.com

output
IP address: 1.1.1.1
-----
DNS LOOKUP

import java.net.*;

public class DNSLookup {

    public static void main(String[] args) {
        try {
            // You can change the domain or take input from Scanner
            String domain = "www.google.com";

            InetAddress address = InetAddress.getByName(domain);

            System.out.println("Domain Name: " + domain);
            System.out.println("IP Address : " + address.getHostAddress());

        } catch (UnknownHostException e) {
            System.out.println("Could not resolve domain. " + e.getMessage());
        }
    }
}

■ DNS Configuration in Cisco Packet Tracer – Clear Steps
■ Step 1: Build the Topology

Devices needed:
```

1 Switch
1 DNS Server
2 PCs (Client1, Client2)

Connections:

Connect Server → Switch (FastEthernet0)
Connect Client1 → Switch (FastEthernet0)
Connect Client2 → Switch (FastEthernet0)

Use Copper Straight-Through cables.

Why: All devices must be in the same LAN so clients can reach the DNS server.

■ Step 2: Assign IP Addresses
Server (DNS Server)

Go to:
Server → Desktop → IP Configuration

IP Address: 192.168.1.2
Subnet Mask: 255.255.255.0

(No gateway needed because all devices are in same LAN.)

Client1

Client1 → Desktop → IP Configuration

IP Address: 192.168.1.3
Subnet Mask: 255.255.255.0
DNS Server: 192.168.1.2

Client2

Client2 → Desktop → IP Configuration

IP Address: 192.168.1.4
Subnet Mask: 255.255.255.0
DNS Server: 192.168.1.2

Why DNS Server IP is required:
Clients use this address to send DNS queries (domain → IP conversion).

■ Step 3: Configure DNS Service on Server

Click Server

Go to Services tab

Select DNS from left panel

Turn DNS Service: ON

Now add domain-IP records:

Example entries:

www.google.com	8.8.8.8
www.vitbhopal.ac.in	192.168.1.2
www.yahoo.com	98.137.11.163

Click Add after each entry.

What this means:

You're programming the DNS server with domain-to-IP mappings so that clients can resolve those names.

■ Step 4: Verify DNS Resolution

Go to Client1 → Desktop → Command Prompt
Type:

```
ping www.google.com
```

If DNS is working, you will see:

```
Pinging 8.8.8.8 with 32 bytes of data:  
Reply from 8.8.8.8: bytes=32 time<1ms TTL=...
```

Try other domains too:

```
ping www.vitbhopal.ac.in  
ping www.yahoo.com
```

What happens:

```
Client sends a DNS query to 192.168.1.2  
Server replies with the IP address  
Client then sends ICMP ping to that resolved IP
```

■ Step 5: Analyze Using Simulation Mode

Switch to Simulation Mode (bottom right corner).

Then:

On Client1 → use ping www.google.com

Watch packets:

DNS Query packet goes from Client → Server

DNS Response returns from Server → Client

Then ICMP packets (ping) start

This proves that name resolution is happening.

■ DHCP Configuration Notes (Packet Tracer)
1. Purpose of DHCP

DHCP (Dynamic Host Configuration Protocol) automatically assigns IP addresses, subnet masks, gateways, a

2. Topology Setup

Router

Switch

PCs (PC0, PC1, PC2...)

Connect using Copper Straight-Through cables:

Router Gi0/0 → Switch

Switch → PCs

3. Configure Router Interface (Gateway)

Open Router → CLI:

```
enable
```

```
configure terminal
interface gigabitEthernet0/0
ip address 192.168.50.1 255.255.255.0
no shutdown
exit
```

Explanation:
Assigns gateway IP and activates router interface.

4. Exclude Router or Reserved IPs
ip dhcp excluded-address 192.168.50.1

Reason:
Prevents DHCP from assigning the router's own IP.

```
5. Create DHCP Pool
ip dhcp pool LAB_POOL
network 192.168.50.0 255.255.255.0
default-router 192.168.50.1
dns-server 8.8.8.8
exit
```

Explanation:
Defines the pool, gateway, network range, and DNS server.

6. Configure PCs to Receive IP Automatically

On each PC:
Desktop → IP Configuration → DHCP

PC will receive:

IP: 192.168.50.x

Mask: 255.255.255.0

Gateway: 192.168.50.1

DNS: 8.8.8.8

7. Verify on PC

Open Command Prompt:

Check IP
ipconfig

Test Connectivity
ping 192.168.50.1

Expected:
Replies from the router = DHCP working.

PING

```
// Implementation of Ping service.
import java.net.*;
import java.io.*;
public class PingService {
    public static void main(String[] args) {
        // Check if the user provided a hostname as an argument
        if (args.length != 1) {
            System.out.println("Usage: java PingService <hostname>");
            return;
    }}
```

```

        String hostname = args[0]; // Get the hostname from command-line arguments
        try {
            System.out.println("Ping " + hostname + "...");
            InetAddress inetAddress = InetAddress.getByName(hostname);
            boolean isReachable = inetAddress.isReachable(5000);
            if (isReachable) {
                System.out.println("Host " + hostname + " is reachable.");
                System.out.println("IP Address: " + inetAddress.getHostAddress());
            } else {
                System.out.println("Host " + hostname + " is not reachable.");
            }
        } catch (UnknownHostException e) {
            System.out.println("Unknown host: " + hostname);
        } catch (IOException e) {
            System.out.println("Error occurred while pinging " + hostname + ": " + e.getMessage());
        }
    }
}

output

PS D:\New folder (4)> javac PingService.java
>>
PS D:\New folder (4)> javac PingService.java
>> java PingService google.com
>>
Pinging google.com...
Host google.com is reachable.
IP Address: 142.251.223.14
PS D:\New folder (4)>

```

VLSM

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024

Mask	CIDR	Block Size
128	/17	128
192	/18	64
224	/19	32
240	/20	16
248	/21	8
252	/22	4
254	/23	2
255	/24	1

subnetting

```

import java.util.*;

public class Subnetting {

    // Convert IP to 32-bit integer
    static long ipToInt(String ip) {
        String[] parts = ip.split("\\.");
        return (Long.parseLong(parts[0]) << 24) |
               (Long.parseLong(parts[1]) << 16) |

```

```

        (Long.parseLong(parts[2]) << 8) |  

        (Long.parseLong(parts[3]));  

    }  
  

    // Convert 32-bit integer back to IP  

    static String intToIp(long ip) {  

        return ((ip >> 24) & 0xFF) + "." +  

            ((ip >> 16) & 0xFF) + "." +  

            ((ip >> 8) & 0xFF) + "." +  

            (ip & 0xFF);  

    }  
  

    // Generate subnet mask from prefix length  

    static long subnetMask(int prefix) {  

        return prefix == 0 ? 0 : ~((1L << (32 - prefix)) - 1);  

    }  
  

    public static void main(String[] args) {  

        Scanner sc = new Scanner(System.in);  
  

        System.out.print("Enter base IP address (e.g., 192.168.10.0): ");  

        String ip = sc.next();  
  

        System.out.print("Enter prefix (e.g., 24): ");  

        int prefix = sc.nextInt();  
  

        System.out.print("Enter number of subnets: ");  

        int subnetCount = sc.nextInt();  
  

        // Convert base IP  

        long baseIp = ipToInt(ip);  
  

        // Extra bits needed for subnetting  

        int extraBits = (int) Math.ceil(Math.log(subnetCount) / Math.log(2));  

        int newPrefix = prefix + extraBits;  
  

        if (newPrefix > 30) {  

            System.out.println("Error: Cannot subnet further.");  

            return;  

        }  
  

        long mask = subnetMask(newPrefix);  

        long blockSize = (1L << (32 - newPrefix));  

        long usableHosts = blockSize - 2;  
  

        System.out.println("\n===== Subnetting Results =====");  

        System.out.println("Original Prefix: /" + prefix);  

        System.out.println("New Prefix: /" + newPrefix);  

        System.out.println("Subnet Mask: " + intToIp(mask));  

        System.out.println("Number of Subnets: " + (1 << extraBits));  

        System.out.println("Usable Hosts per Subnet: " + usableHosts);  
  

        // Print each subnet details  

        System.out.println("\n===== Subnet Details =====");  

        for (int i = 0; i < subnetCount; i++) {  

            long network = baseIp + (i * blockSize);  

            long broadcast = network + blockSize - 1;  
  

            System.out.println("\nSubnet " + (i + 1) + ":" );  

            System.out.println("Network Address: " + intToIp(network));  

            System.out.println("Broadcast Address: " + intToIp(broadcast));  

            System.out.println("Usable Host Range: "  

                + intToIp(network + 1) + " to " + intToIp(broadcast - 1));  

        }
    }
}

Enter base IP: 192.168.10.0
Enter prefix: 24
Enter number of subnets: 4
-----

```

```

import java.util.*;

public class SubnetCalculator {

    // Convert IP to 32-bit integer
    static long ipToInt(String ip) {
        String[] parts = ip.split("\\.");
        return (Long.parseLong(parts[0]) << 24) |
            (Long.parseLong(parts[1]) << 16) |
            (Long.parseLong(parts[2]) << 8) |
            (Long.parseLong(parts[3]));
    }

    // Convert integer back to IP string
    static String intToIp(long ip) {
        return ((ip >> 24) & 0xFF) + "." +
            ((ip >> 16) & 0xFF) + "." +
            ((ip >> 8) & 0xFF) + "." +
            (ip & 0xFF);
    }

    // Create subnet mask using prefix
    static long subnetMask(int prefix) {
        return ~((1L << (32 - prefix)) - 1);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter network address (e.g., 10.0.4.0): ");
        String ip = sc.nextLine();

        System.out.print("Enter prefix (e.g., 21): ");
        int prefix = sc.nextInt();

        long baseIp = ipToInt(ip);
        long mask = subnetMask(prefix);

        long network = baseIp & mask;
        long broadcast = network | ~mask;

        long totalHosts = (1L << (32 - prefix));
        long usableHosts = totalHosts - 2;

        System.out.println("\n===== Subnet Calculation Results =====");
        System.out.println("Network Address : " + intToIp(network));
        System.out.println("Broadcast Address : " + intToIp(broadcast));
        System.out.println("Total Usable Hosts : " + usableHosts);
        System.out.println("First Usable IP : " + intToIp(network + 1));
        System.out.println("Last Usable IP : " + intToIp(broadcast - 1));
    }
}

Enter network address (e.g., 10.0.4.0): 192.168.10.0
Enter prefix (e.g., 21): 24

===== Subnet Calculation Results =====
Network Address : 192.168.10.0
Broadcast Address : 192.168.10.255
Total Usable Hosts : 254
First Usable IP : 192.168.10.1
Last Usable IP : 192.168.10.254

```