subnet

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

unsigned int ipToInt(char* ip)
{
   unsigned int a, b, c, d;
   sscanf(ip, "%u.%u.%u.%u", &a, &b, &c, &d);
   return (a << 24) | (b << 16) | (c << 8) | d;
}

void intToIp(unsigned int ip, char* buffer)
{
   sprintf(buffer, "%u.%u.%u.%u", (ip >> 24) & 0xFF, (ip >> 16) & 0xFF, (ip >> 8) & 0xFF, ip &0xFF);
}

unsigned int calculateSubnetMask(int prefixLength)
{
   return prefixLength == 0 ? 0 : ~((1 << (32 - prefixLength)) - 1);
}

int main()
{
   char ip[16];
   int prefixLength, newPrefixLength;
   unsigned int subnetMask, newSubnetMask, ipInt;
   char buffer[16];

   printf("Enter IP address (e.g., 192.168.1.0): ");
   scanf("%s", ip);
```

```
printf("Enter current prefix length (e.g., 24): ");
scanf("%d", &prefixLength);

newPrefixLength = prefixLength + 1;
ipInt = ipToInt(ip);
subnetMask = calculateSubnetMask(prefixLength);
newSubnetMask = calculateSubnetMask(newPrefixLength);
int hostsPerSubnet = (1 << (32 - newPrefixLength)) - 2;

printf("\nNumber of subnets: 2\n");
printf("Number of hosts per subnet: %d\n", hostsPerSubnet);

for (int i = 0; i < 2; i++)
{
   unsigned int subnetNetwork = (ipInt & subnetMask) | (i << (32 - newPrefixLength));
   unsigned int subnetBroadcast = subnetNetwork | ~newSubnetMask;
   unsigned int firstHost = subnetNetwork + 1;
   unsigned int lastHost = subnetBroadcast - 1;

   printf("\nSubnet %d:\n", i + 1);
   printf("Network Address: ");
   intToIp(subnetNetwork, buffer);
   printf("%s\n", buffer);
   printf("Broadcast Address: ");
   intToIp(subnetBroadcast, buffer);
   printf("%s\n", buffer);
   printf("Subnet Mask: ");
   intToIp(newSubnetMask, buffer);
   printf("%s\n", buffer);
   printf("First Host: ");
   intToIp(firstHost, buffer);
```

```
        printf("%s\n", buffer);
        printf("Last Host: ");
        intToIp(lastHost, buffer);
        printf("%s\n", buffer);
    }
    return 0;
}
```

distance vector

```
#include <stdio.h>
#include <stdlib.h>

#define INF 9999
#define MAX_NODES 10

void initialize(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int
distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES])
{
    for (int i = 0; i < numNodes; i++)
    {
        for (int j = 0; j < numNodes; j++)
        {
            distVector[i][j] = costMatrix[i][j];

            if (costMatrix[i][j] != INF && i != j)
            {
                nextHop[i][j] = j;
            }

            else
            {
                nextHop[i][j] = -1;
```

```
            }
        }
    }
}

void printRoutingTable(int numNodes, int distVector[MAX_NODES][MAX_NODES], int
nextHop[MAX_NODES][MAX_NODES])
{
    for (int i = 0; i < numNodes; i++)
    {
        printf("Routing table for node %d:\n", i);
        printf("Destination\tNext Hop\tDistance\n");

        for (int j = 0; j < numNodes; j++)
        {
            if (distVector[i][j] == INF)
            {
                printf("%d\t\t-\t\tINF\n", j);
            }

            else
            {
                printf("%d\t\t%d\t\t%d\n", j, nextHop[i][j], distVector[i][j]);
            }
        }
        printf("\n");
    }
}

void distanceVectorRouting(int numNodes, int costMatrix[MAX_NODES][MAX_NODES], int
distVector[MAX_NODES][MAX_NODES], int nextHop[MAX_NODES][MAX_NODES])
{
```

```
    int updated;

    do
    {
        updated = 0;

        for (int i = 0; i < numNodes; i++)
        {
            for (int j = 0; j < numNodes; j++)
            {
                for (int k = 0; k < numNodes; k++)
                {
                    if (distVector[i][k] + distVector[k][j] < distVector[i][j])
                    {
                        distVector[i][j] = distVector[i][k] + distVector[k][j];
                        nextHop[i][j] = nextHop[i][k];
                        updated = 1;
                    }
                }
            }
        }
    } while (updated);
}

int main()
{
    int numNodes, costMatrix[MAX_NODES][MAX_NODES];
    int distVector[MAX_NODES][MAX_NODES];
    int nextHop[MAX_NODES][MAX_NODES];

    printf("Enter the number of nodes: ");
```

```
    scanf("%d", &numNodes);
    printf("Enter the cost matrix (use %d for INF):\n", INF);

    for (int i = 0; i < numNodes; i++)
    {
        for (int j = 0; j < numNodes; j++)
        {
            scanf("%d", &costMatrix[i][j]);
        }
    }

    initialize(numNodes, costMatrix, distVector, nextHop);
    distanceVectorRouting(numNodes, costMatrix, distVector, nextHop);
    printRoutingTable(numNodes, distVector, nextHop);
    return 0;
}
```

: dns

```
import java.net.*;
import java.util.Scanner;

public class SimpleDNSResolver
{
    public static void main(String[] args)
    {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Welcome to the Simple DNS Resolver!");
        System.out.println("Enter a domain name to resolve (or type 'exit' to quit):");

        while (true)
        {
```

```
        System.out.print("Domain: ");
        String domain = scanner.nextLine();

        if ("exit".equalsIgnoreCase(domain))
        {
            System.out.println("Exiting DNS Resolver. Goodbye!");
            break;
        }

        try
        {
            InetAddress[] addresses = InetAddress.getAllByName(domain);
            System.out.println("IP addresses for " + domain + ":");

            for (InetAddress address : addresses)
            {
                System.out.println(" - " + address.getHostAddress());
            }
        }
        catch (UnknownHostException e)
        {
            System.out.println("Could not resolve domain: " + e.getMessage());
        }
      }
      scanner.close();
   }
}
```

ping

```
import java.net.*;
import java.io.*;
```

```
public class PingService
{
   public static void main(String[] args)
   {
      if (args.length != 1)
      {
         System.out.println("Usage: java PingService <hostname>");
         return;
      }

      String hostname = args[0];

      try
      {
         System.out.println("Pinging " + hostname + "...");
         InetAddress inetAddress = InetAddress.getByName(hostname);
         boolean isReachable = inetAddress.isReachable(15000);

         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());
        }
    }
}
```