IPV4:

```
import java.util.*;
import java.lang.*;

class ipv{

   public static void main(String[] args){

            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the IP address");
            String data = sc.nextLine();
            //System.out.println(data);
            String[] ip = data.split("\\.");
            int num = Integer.parseInt(ip[0]);
            //String value = Integer.toBinaryString(num);
            //System.out.println(value);
            if(num<128){
                        System.out.println("Class A ");
            }else if(num>127 && num<192){
                        System.out.println("Class B ");
            }else if(num>191 && num<224){
                        System.out.println("Class C ");
            }else if(num>223 && num<240){
                        System.out.println("Class D ");
            }else if(num>239 && num<256){
                        System.out.println("Class E ");
            }else{
                        System.out.println("Invalid ");
            }
   }

}
/*
Output :
Enter the IP address
122.45.0.1
Class A

Enter the IP address
192.32.20.0
Class C

Enter the IP address
212.3.0.0
```

Class C

Enter the IP address
356.02.20
Invalid
*/

DISTANCE VECTOR ROUTING CODE:

```
#include <stdio.h>
#include <stdlib.h>
#define INFINITY 99999
int n,m[100][100],source;
void display(int * visited,int * via,int * weight);
void dijkshtra();
int min(int * , int *);
int main()
{
   int i,j;
   printf("Enter the number of nodes : ");
   scanf("%d",&n);
   printf("Enter the adjacency matrix for the graph with weights\n");
   for (i=0;i<n;i++)
   {
      for (j=0;j<n;j++)
      {
         scanf("%d",&m[i][j]);
      }
      printf("\n");
   }
   printf("The Adjacency matrix is given as shown \n");
   for (i=0;i<n;i++)
   {
      for (j=0;j<n;j++)
      {
         printf("%d ",m[i][j]);
      }
      printf("\n");
   }
   for (i=0 ;i < n ; i++)
   {
      source = i;
      printf("The distance matrix from the node %d is \n" , i);
      dijkshtra();
```

```
    }
    return 0;
}
void dijkshtra()
{
    int visited[n],via[n],weight[n];
    int i,j,node;
    for (i=0;i<n;i++)
    {
        visited[i]=0;
        via[i]=-1;
        weight[i]=INFINITY;
    }
    weight[source]=0;
    for (i=0;i<(n-1);i++)
    {
        node = min(weight,visited);
        visited[node]=1;
        for (j=0;j<n;j++)
        {
            if (visited[j]==0 && m[node][j]!=0 && weight[node]!=INFINITY)
            {
                if (weight[j]>m[node][j]+weight[node])
                {
                    via[j] = node;
                    weight[j] = m[node][j]+weight[node];
                }
            }
        }
    }
    display(visited,via,weight);
}
void display(int visited[],int via[],int weight[])
{
    int i;
    printf("Visited : ");
    for (i=0;i<n;i++)
    {
        printf("%d ",visited[i]);
    }
    printf("\nFrom : ");
    for (i=0;i<n;i++)
    {
        printf("%d ",source);
    }
```

```
    printf("\nVia : ");
    for (i=0;i<n;i++)
    {
        printf("%d ",via[i]);
    }
    printf("\nto : ");
    for (i=0;i<n;i++)
    {
        printf("%d ",i);
    }
    printf("\nweight : ");
    for (i=0;i<n;i++)
    {
        printf("%d ",weight[i]);
    }
    printf("\n\n");
}
int min(int weight[],int visited[])
{
    int i,q;
    int temp=INFINITY;
    for (i=0;i<n;i++)
    {
        if (visited[i] == 0 && temp>=weight[i])
        {
            temp=weight[i];
            q=i;
        }
    }
    return q;
}
/*
OUTPUT 1:
Enter the number of nodes : 4
Enter the adjacency matrix for the graph with weights
0 2 7 0

2 0 1 3

7 1 0 1

0 3 1 0

The Adjacency matrix is given as shown
0 2 7 0
```

2 0 1 3
7 1 0 1
0 3 1 0
The distance matrix from the node 0 is
Visited : 1 1 1 0
From : 0 0 0 0
Via : -1 0 1 2
to : 0 1 2 3
weight : 0 2 3 4

The distance matrix from the node 1 is
Visited : 0 1 1 1
From : 1 1 1 1
Via : 1 -1 1 2
to : 0 1 2 3
weight : 2 0 1 2

The distance matrix from the node 2 is
Visited : 0 1 1 1
From : 2 2 2 2
Via : 1 2 -1 2
to : 0 1 2 3
weight : 3 1 0 1

The distance matrix from the node 3 is
Visited : 0 1 1 1
From : 3 3 3 3
Via : 1 2 3 -1
to : 0 1 2 3
weight : 4 2 1 0

Process returned 0 (0x0) execution time : 72.187 s
Press any key to continue.

OUTPUT 2:
Enter the number of nodes : 6
Enter the adjacency matrix for the graph with weights
0 2 5 8 0 0

2 0 2 0 0 6

5 2 0 4 1 0

8 0 4 0 7 0

0 0 1 7 0 4

0 6 0 0 4 0

The Adjacency matrix is given as shown
0 2 5 8 0 0
2 0 2 0 0 6
5 2 0 4 1 0
8 0 4 0 7 0
0 0 1 7 0 4
0 6 0 0 4 0
The distance matrix from the node 0 is
Visited : 1 1 1 0 1 1
From : 0 0 0 0 0 0
Via : -1 0 1 0 2 1
to : 0 1 2 3 4 5
weight : 0 2 4 8 5 8

The distance matrix from the node 1 is
Visited : 1 1 1 0 1 1
From : 1 1 1 1 1 1
Via : 1 -1 1 2 2 1
to : 0 1 2 3 4 5
weight : 2 0 2 6 3 6

The distance matrix from the node 2 is
Visited : 1 1 1 1 1 0
From : 2 2 2 2 2 2
Via : 1 2 -1 2 2 4
to : 0 1 2 3 4 5
weight : 4 2 0 4 1 5

The distance matrix from the node 3 is
Visited : 1 1 1 1 1 0
From : 3 3 3 3 3 3
Via : 3 2 3 -1 2 4
to : 0 1 2 3 4 5
weight : 8 6 4 0 5 9

The distance matrix from the node 4 is
Visited : 0 1 1 1 1 1
From : 4 4 4 4 4 4
Via : 1 2 4 2 -1 4

to : 0 1 2 3 4 5
weight : 5 3 1 5 0 4

The distance matrix from the node 5 is
Visited : 1 1 1 0 1 1
From : 5 5 5 5 5 5
Via : 1 5 4 2 5 -1
to : 0 1 2 3 4 5
weight : 8 6 5 9 4 0

Process returned 0 (0x0) execution time : 71.118 s
Press any key to continue.
*/