Dijsktra

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) {
  int cost[MAX][MAX], distance[MAX], pred[MAX];
  int visited[MAX], count, mindistance, nextnode, i, j;

  // Creating cost matrix
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (Graph[i][j] == 0)
        cost[i][j] = INFINITY;
      else
        cost[i][j] = Graph[i][j];

  for (i = 0; i < n; i++) {
    distance[i] = cost[start][i];
    pred[i] = start;
    visited[i] = 0;
  }

  distance[start] = 0;
  visited[start] = 1;
  count = 1;

  while (count < n - 1) {
    mindistance = INFINITY;

    for (i = 0; i < n; i++)
      if (distance[i] < mindistance && !visited[i]) {
        mindistance = distance[i];
        nextnode = i;
      }

    visited[nextnode] = 1;
    for (i = 0; i < n; i++)
      if (!visited[i])
        if (mindistance + cost[nextnode][i] < distance[i]) {
          distance[i] = mindistance + cost[nextnode][i];
          pred[i] = nextnode;
        }
    count++;
  }

  // Printing the distance
  for (i = 0; i < n; i++)
    if (i != start) {
      printf("\nDistance from source to %d: %d", i, distance[i]);
    }
}
int main() {
```

```
  int graph[MAX][MAX], i, j, n, u;
  n = 7;

  for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
  u = 0;
  Dijkstra(graph, n, u);

  return 0;
}


// TCP Client

import java.net.*;
import java.io.*;
import java.util.*;

public class client{
  public static void main(String[] args){
    try{
      Socket client = new Socket("localhost" , 7000);
      System.out.println("Client is connected");

      Scanner sc = new Scanner(System.in);

      BufferedReader br = new BufferedReader(new InputStreamReader(client.getInputStream()));
      PrintWriter pw = new PrintWriter(client.getOutputStream());

        // String data = br.readLine();
        // System.out.println("Data from server: " + data);


        System.out.println("Enter text: ");
        String input = sc.nextLine();

        pw.println(input);
        pw.flush();

        String output = br.readLine();

        System.out.println("Data from server: " +  output);


    }catch(Exception err){
      System.out.println(err);
    }
  }
}


// TCP server
```

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

public class server {
  public static void main (String[] args){
      try{
        ServerSocket ss = new ServerSocket(7000);
        System.out.println("Waiting for Client");


        System.out.println("Server is Connected");

        Scanner sc = new Scanner(System.in);


        while(true){
          // System.out.println("Enter Data:");
          // String data = sc.nextLine();
          // pw.println(data);
          // pw.flush();
        Socket server = ss.accept();
        PrintWriter pw = new PrintWriter(server.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(server.getInputStream()));

          String input;
          String output;

          input = br.readLine();


          output = input.toUpperCase() + '\n';

          pw.println(output);
          pw.flush();
        }
      } catch(Exception err){
        System.out.println(err);
      }
  }
}


// UDP Client

import java.net.*;
import java.io.*;
import java.util.*;

public class udpClient{
  public static void main(String[] args){
    try{
      DatagramSocket client = new DatagramSocket();
      Scanner sc = new Scanner(System.in);
```

```
      byte[] receiveData = new byte[1024];
      byte[] sendData;


      System.out.println("Enter text: ");
      String input = sc.nextLine();

      sendData = input.getBytes();

      InetAddress Ip = InetAddress.getByName("localhost");

      DatagramPacket sendPacket = new DatagramPacket(sendData , sendData.length , Ip , 7000);

      client.send(sendPacket);

      DatagramPacket receivePacket = new DatagramPacket(receiveData , receiveData.length);

      client.receive(receivePacket);

      String output = new String(receivePacket.getData() , 0 , receivePacket.getLength());

      System.out.println("From Server: " + output);

      client.close();


    } catch(Exception err){
      System.out.println(err);
    }
  }
}


// UDP Server

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

public class udpServer {
  public static void main(String[] args){

  try{
    DatagramSocket server = new DatagramSocket(7000);

    byte[] receivedData =  new byte[1024];
    byte[] sendData;

    while(true){
      DatagramPacket receivePacket = new DatagramPacket(receivedData , receivedData.length);

      System.out.println("Waiting for client...");

      server.receive(receivePacket);
```

```
      System.out.println("Received...");

      String input = new String(receivePacket.getData() , 0 , receivePacket.getLength());

      InetAddress Ip = receivePacket.getAddress();

      int port = receivePacket.getPort();

      String output = input.toUpperCase();

      sendData = output.getBytes();

      DatagramPacket sendPacket = new DatagramPacket(sendData , sendData.length , Ip , port);

      server.send(sendPacket);

      System.out.println("Sent Successfully..." + "\n");
    }
  }
  catch(Exception err){
    System.out.println(err);
  }
  }
}



// Floyd-Warshall Algorithm in C

#include <stdio.h>

// defining the number of vertices
#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

// Implementing floyd warshall algorithm
void floydWarshall(int graph[][nV]) {
  int matrix[nV][nV], i, j, k;

  for (i = 0; i < nV; i++)
    for (j = 0; j < nV; j++)
      matrix[i][j] = graph[i][j];

  // Adding vertices individually
  for (k = 0; k < nV; k++) {
    for (i = 0; i < nV; i++) {
      for (j = 0; j < nV; j++) {
        if (matrix[i][k] + matrix[k][j] < matrix[i][j])
          matrix[i][j] = matrix[i][k] + matrix[k][j];
      }
    }
  }
```

```
  printMatrix(matrix);
}

void printMatrix(int matrix[][nV]) {
  for (int i = 0; i < nV; i++) {
    for (int j = 0; j < nV; j++) {
      if (matrix[i][j] == INF)
        printf("%4s", "INF");
      else
        printf("%4d", matrix[i][j]);
    }
    printf("\n");
  }
}

int main() {
  int graph[nV][nV] = {{0, 3, INF, 5},
              {2, 0, INF, 4},
              {INF, 1, 0, INF},
              {INF, INF, 2, 0}};
  floydWarshall(graph);
}
```