A STUDY OF DISCRETE FUZZY SHORTEST PATH PROBLEM IN A NETWORK

Summer Training Report

Submitted by
Tannishtha Som
BTech, Second Year
Computer Science and Engineering Department
NIT Durgapur

Under the Guidance of
Prof. K.K.Shukla

DEPARTMENT OF COMPUTER ENGINEERING
INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY, VARANASI-221005

DEPARTMENT OF COMPUTER ENGINEERING
INSTITUTE OF TECHNOLOGY
BANARAS HINDU UNIVERSITY
VARANASI-221005, INDIA

Dr. K.K.Shukla
Professor
Computer Science and Engineering

Tel # +91-542-2307056
Fax # +91-542-2368428
Email-kkshukla@bhu.ac.in

Date-

CERTIFICATE

This is to certify that Tannishtha Som, student of Department of Computer Science and Engineering, National Institute of Technology, Durgapur has worked for her Summer Training project entitled

"A Study of Discrete Fuzzy Shortest Path Problem"

from 12.5.12 to 12.7.12 in Computer Science and Engineering Department, Institute of Technology, BHU. The report submitted by her embodies the literature from various sources and from the material provided by me during the period.

Prof. K.K. Shukla
Dated:
Supervisor
Department of Computer Engineering
Institute of Technology, BHU

ACKNOWLEDGEMENT

It has been indeed a great privilege for me to have Prof. K. K. Shukla, Department of Computer Science and Engineering, Institute of Technology, Banaras Hindu University, as my supervisor. His awe-inspiring personality, superb guidance and constant encouragement were the motive force behind this project work.

I would also thank my Parents and sister for their encouragement to pursue the work.

Tannishtha Som

ABSTRACT

The Shortest Path Problem is one of the most fundamental graph theory problems that appear in many applications as a sub-problem. In real world problems, arc lengths represent travelling time, cost, distance or other variables. However uncertainty cannot be avoided, and hence the arc lengths can be assumed to be fuzzy sets. This is the discrete fuzzy shortest path problem. The Paper entitled “A new algorithm for the discrete fuzzy shortest path problem in a network”, Tzung-Nan Chuang, Jung-Yuan Kung, Applied Maths and Computation 174(2006) 660-668, has been studied extensively. The paper states an algorithm that obtains the fuzzy shortest length in a type V graph by the discrete fuzzy shortest length method and then fuzzy similarity measure is utilized to obtain the shortest path in the graph which is most similar to the fuzzy shortest length. A new algorithm is then proposed where instead of using fuzzy similarity method, Hamming distance method is employed to obtain the shortest path in the graph. Comparisons in the number of operations between the two methods are shown in a tabular form by taking graphs with various nodes. The code implemented is also shown and it has been described by taking a graph with 30 nodes with its output.

CONTENTS

1 Introduction.....1
1.1 Shortest Path Problem..... 1
1.2 Fuzzy Shortest Path Problem.....1
1.3 Fuzzy Logic.....1
2 Work Done.....3
2.1 Details of the Paper.....3
2.2 The new algorithm proposed 4
2.3 Implementation of the algorithms.....5
3 Observations.....8
4 Conclusions and future directions.....10
5 Applications of Shortest Path Problem.....11
Appendix (code)……..……..12
References.....19

1. INTRODUCTION

1.1 Shortest Path Problem

The Shortest Path Problem is one of the most fundamental graph theory problems that appear in many applications as a sub-problem. It seeks to find a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment [8].

1.2 Fuzzy Shortest Path Problem

In real world problems, arc lengths (or edge weights) in the Shortest Path Problem represent travelling time, cost, distance or other variables. However, in practice, uncertainty cannot be avoided, and usually, the arc lengths cannot be determined precisely. For instance, on road networks, for several reasons like traffic, accidents or weather condition, arc lengths representing the vehicle travel time, is subject to uncertainty. In such situations, a fuzzy shortest path problem seems to be more realistic and reliable.

1.3 Fuzzy Logic

Fuzzy logic is a form of many-valued logic or probabilistic logic; it deals with reasoning that is approximate rather than fixed and exact. In contrast with traditional logic theory, where binary sets have two-valued logic, true or false, fuzzy logic variables may have a truth value that ranges in degree between 0 and 1. Fuzzy logic has been extended to handle the concept of partial truth, where the truth value may range between completely true and completely false.

Fuzzy logic began with the 1965 proposal of fuzzy set theory by Lotfi Zadeh. Fuzzy logic has been applied to many fields, from control theory to artificial intelligence.

Definition of Fuzzy Set [1]:

If X is a collection of objects denoted generically by x then a fuzzy set A in X is a set of ordered pairs:

$$A = \{(x, \mu(x)) \mid x \in X\}$$

where $\mu(x)$ is called the membership function or grade of membership of x in A which maps X to the membership space M. The value of $\mu(x)$ lies between 0 and 1.

Definitions of some basic fuzzy set operations are as follows [2]:

(a) The *intersection* of fuzzy sets A and B is denoted by $A \cap B$ and the membership function of $A \cap B$ is given by

$(A \cap B)(x) = \min [(A(x), B(x))] \quad \forall x \in X$ {A(x), B(x) are the membership functions of sets A and B respectively.}

(b) The *union* of fuzzy sets A and B is denoted by $A \cup B$ and the membership function of $A \cup B$ is given by

$(A \cup B)(x) = \max [(A(x), B(x))] \quad \forall x \in X$

(c) The *addition* of fuzzy sets A and B is denoted by A + B and the membership function of A + B is given by

$(A + B)(z) = \max_{z=x+y} \min[(A(x), B(y))] \quad \forall x, y \in X.$

2.Work Done

The Paper entitled "**A new algorithm for the discrete fuzzy shortest path problem in a network**",Tzung-Nan Chuang,Jung-Yuan Kung, Applied Maths and Computation 174(2006) 660-668, has been studied extensively during the training.

2.1 Details of the Paper

The Paper discusses the fuzzy shortest path problem on Type V Fuzzy graph [4], i.e., crisp graph with fuzzy weights which means that the graph has known vertices and edges but the weights on the edges are fuzzy. The paper has a proposed new algorithm that obtains the *fuzzy shortest length* in a type V graph by the *discrete fuzzy shortest length method* and then *fuzzy similarity measure* is utilized to get the shortest path in the graph which is most similar to the fuzzy shortest length.

The discrete fuzzy shortest length method: In the crisp world, a length is the shortest one if any other length is greater than or equal to it. In other words, if a length is the shortest one of a set of lengths, then this length does exist and the other length smaller than it does not exist. This idea is extended to find the fuzzy shortest length which is a fuzzy set [3].
Assume there are m discrete fuzzy path lengths, L_1, L2, . . . ,L_m. $L_1(x)$/represents that the element x in L_1 has a membership grade $L_1(x)$. As noted for crisp lengths, x is the shortest length if any other length is greater than or equal to x. Therefore, the membership grade $SL_1(x)$ of x in L_1 is the possibility that x in L_1 is the shortest length, and it can be expressed as

$SL_1(x) = L_1(x) \cap L'_2(x) \cap L'_3(x) \cap \ldots\ldots\ldots \cap L'_m(x)$
$= \min [L_1(x), L'_2(x), L'_3(x), \ldots\ldots\ldots, L'_m(x)]$
$= \min [\, L_1(x), \min_{k\neq 1}[L'_k(x)]]$

where $L'_k(x)$ denotes the membership grade for all lengths in L'_k not greater than x. It can be expressed as

$L'_k(x) = 1 - \max_{y\in L_k}^{y<x} L_k(y).$

Thus we get

$SL_1(x) = \min \left[L_1(x), \min_{k\neq 1}\left(1 - \max_{y\in L_k}^{y<x} L_k(y)\right) \right]$

Similarly, we can obtain the membership grade $SL_t(x)$ of x in L_t as

$$SL_t(x) = \min\left[L_t(x), \min_{k\neq t}\left(1 - \max_{y\in L_k}{}^{y<x} L_k(y)\right)\right]$$

The membership grade $L_{min}(x)$ of x as the shortest length of m discrete fuzzy lengths is then

$$L_{min}(x) = SL_1(x) \cup SL_2(x) \cup SL_3(x) \cup \ldots\ldots\ldots \cup SL_m(x) = {}_{t=1}^{m}max\ SL_t(x)$$

Fuzzy similarity measure: In many practical situations, we often encounter how to distinguish between two similar sets or groups. That is to say, we need to employ a measurement tool to measure similarity degree between them.
In this paper the following similarity measure is used to measure the similarity degree between two fuzzy lengths [5, 6]:

$$S(A,B) = \left[\ \sum_{k=1}^{n}[1-|A(x_k) - B(x_k)|]\ \right] / n$$

where 'n' is the number of elements in the Universal Set.

The algorithm stated in the paper:
Step 1. Form the possible paths from source node s to destination node d and compute the corresponding path lengths L_i, i = 1,2,. . . ,m, for possible m paths by fuzzy addition method defined earlier.
Step 2. Find the fuzzy shortest length L_{min} by using the discrete fuzzy shortest length method.
Step 3. Employ fuzzy similarity measure defined above to yield the similarity degree $S(L_{min},L_i)$ between L_{min} and L_i for i = 1,2,. . . ,m.
Step 4. Decide the shortest path with the highest $S(L_{min},L_i)$.

2.2 The new algorithm proposed

The above algorithm used similarity measure to calculate the similarity degree between the fuzzy shortest path and the calculated paths in the graph. This similarity measure is used to determine the shortest path.
This measure of similarity degree is analogical to estimating the distance between two fuzzy sets. And hence in place of similarity measure Hamming Distance method has been used in the changed algorithm.
Hamming Distance is defined as follows [7]:

$$d_H(A,B) = \sum_{i=1}^{n}|A(x_i) - B(x_i)|$$

The new algorithm:
Step 1. Form the possible paths from source node s to destination node d and

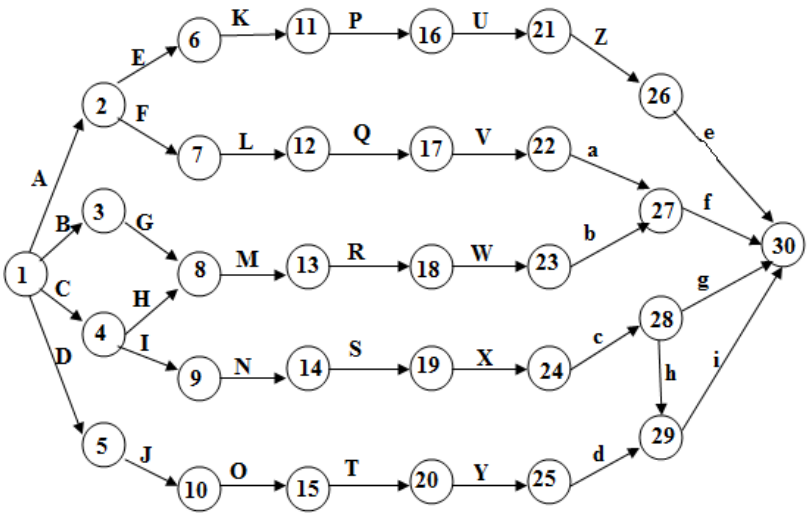
compute the corresponding path lengths L_i, i = 1,2,. . . ,m, for possible m paths by fuzzy addition method defined earlier.
Step 2. Find the fuzzy shortest length L_{min} by using the discrete fuzzy shortest length method.
Step 3. Employ Hamming Distance method defined above to yield the distance H (L_{min}, Li) between L_{min} and L_i for i = 1, 2, . . ., m.
Step 4. Decide the shortest path with the least H (L_{min}, Li).

2.3 Implementation of the above algorithms

The above two algorithms have been implemented in C++ in Microsoft Visual C++ 2008 Express Edition. Adjacency list has been used to store the graph. A predecessor matrix stores the predecessor of each of the vertices in the graph. The predecessors are calculated using a modified BFS where we do not mark the already visited vertex but go on visiting it whenever we come across them and store it in the matrix. The number of vertices in the predecessor matrix of the destination node gives the value of m i.e. the number of paths in the graph from source to destination. The calculated paths are stored as linked list.

An example is given below of a graph with 30 vertices:

Input graph



A= {0.5/2, 1.0/3} B= {0.4/1, 1.0/3} C= {1.0/2, 0.2/3, 0.1/4} D= {1.0/2, 0.6/4}

E= {1.0/3, 0.5/4} F= {1.0/4, 0.7/5} G= {0.7/4, 0.5/5} H= {0.5/5, 0.8/6}

I= {1.0/4, 0.5/5} J= {0.5/1, 0.7/2} K= {0.4/5, 0.7/6} L= {0.3/7, 0.5/8}

M= {0.8/5, 0.6/6} N= {0.4/3, 0.7/4} O= {0.6/5, 0.9/6} P= {0.3/5, 0.7/6}

Q= {0.5/2, 0.7/3} R= {0.4/2, 0.7/3} S= {1.0/1, 0.2/2} T= {0.4/3, 0.8/4}

U= {0.6/4, 1.0/5, 0.5/6} V= {0.4/4, 0.6/5} W= {0.3/1, 0.7/2} X= {0.6/5, 0.1/6}

c= {0.4/5, 0.9/6} d= {0.8/4, 0.6/5} e= {1.0/3, 0.5/5} f= {0.6/3, 0.7/4}

g= {0.1/2, 1.0/6} h= {0.6/5, 1.0/6} i= {0.4/3, 1.0/4}

Output

The number of paths from 1 to 30 is 7

The paths from 1 to 30 are:

a) 1-->4-->9-->14-->19-->24-->28-->29-->30
L[1]=0.4/28 0.4/29 0.4/30 0.6/31 0.6/32 0.5/33 0.2/34 0.2/35 0.1/36 0.1/37

b) 1-->5-->10-->15-->20-->25-->29-->30
L[2]=0.3/21 0.4/22 0.4/23 0.4/24 0.4/25 0.4/26 0.4/27 0.4/28 0.4/29

c) 1-->4-->9-->14-->19-->24-->28-->30
L[3]=0.1/22 0.1/23 0.1/24 0.1/25 0.4/26 0.4/27 0.6/28 0.5/29 0.2/30 0.2/31 0.1/32 0.1/33

d) 1-->4-->8-->13-->18-->23-->27-->30
L[4]=0.3/21 0.4/22 0.4/23 0.4/24 0.5/25 0.6/26 0.7/27 0.6/28 0.2/29 0.1/30

e) 1-->3-->8-->13-->18-->23-->27-->30
L[5]=0.3/19 0.4/20 0.4/21 0.4/22 0.4/23 0.4/24 0.6/25 0.7/26 0.6/27 0.5/28

f) 1-->2-->7-->12-->17-->22-->27-->30
L[6]=0.2/24 0.2/25 0.2/26 0.3/27 0.4/28 0.5/29 0.5/30 0.5/31 0.5/32 0.5/33

g) 1-->2-->6-->11-->16-->21-->26-->30
L[7]=0.3/25 0.4/26 0.5/27 0.6/28 0.7/29 0.5/30 0.5/31 0.5/32 0.5/33 0.4/34 0.4/35

The fuzzy shortest length L_{min} is :
0.3/19 0.4/20 0.4/21 0.4/22 0.4/23 0.4/24 0.6/25 0.5/26 0.4/27 0.3/28

0.3/29 0.3/30 0.3/31 0.3/32 0.3/33 0.3/34 0.3/35 0.1/36 0.1/37

The code with fuzzy similarity measure method and the other code with Hamming distance method give the following results:

The fuzzy similarity measures :

S(Lmin,L[1])= 0.811111
S(Lmin,L[2])= 0.877778
S(Lmin,L[3])= 0.833333
S(Lmin,L[4])= 0.866667
S(Lmin,L[5])= 0.892593
S(Lmin,L[6])= 0.822222
S(Lmin,L[7])= 0.825926

The Hamming distance measures:

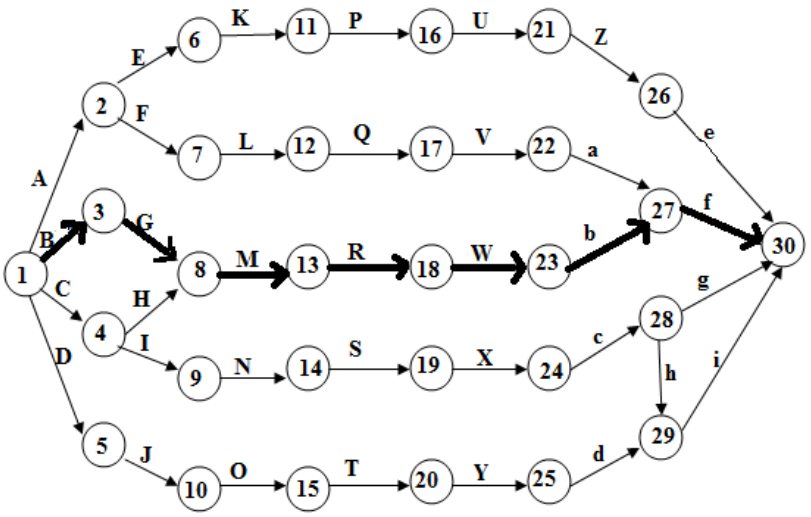
H(Lmin,L[1])= 5.1
H(Lmin,L[2])= 3.3
H(Lmin,L[3])= 4.5
H(Lmin,L[4])= 3.6
H(Lmin,L[5])= 2.9
H(Lmin,L[6])= 4.8
H(Lmin,L[7])= 4.7

Here the highest similarity degree to the fuzzy shortest length is 0.892593 and this corresponds to the path 1→ 3→ 8→ 13→ 18→ 23→ 27→ 30

Also the lowest Hamming Distance to the fuzzy shortest length is 2.9 and this also corresponds to the path 1→ 3→ 8→ 13→ 18→ 23→ 27→ 30

Thus both the methods give us the same result.

The shortest path in the graph is shown below:



3.Observations:

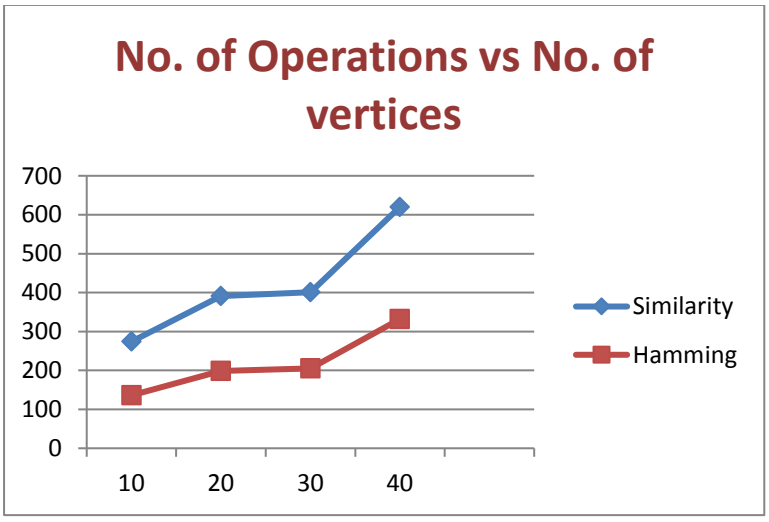
Here we make a comparison in the number of operations between the similarity measure method and the Hamming distance method by taking graphs with various numbers of nodes.

The results are shown in a tabular form as below:

No. of Nodes	Similarity measure						Hamming distance					
	Path	No. Of Operations				Total	Path	No. Of Operations				Total
		Add	sub	Mult	div			add	Sub	mult	Div	
10	L[1]	22	22	0	1	45	L[1]	15	7	0	0	22
	L[2]	22	23	0	1	46	L[2]	15	8	0	0	23
	L[3]	22	24	0	1	47	L[3]	15	9	0	0	24
	L[4]	22	22	0	1	45	L[4]	15	7	0	0	22
	L[5]	22	22	0	1	45	L[5]	15	7	0	0	22
	L[6]	22	23	0	1	46	L[6]	15	8	0	0	23
						274						**136**
20	L[1]	23	27	0	1	51	L[1]	15	12	0	0	27
	L[2]	23	27	0	1	51	L[2]	15	12	0	0	27
	L[3]	23	25	0	1	49	L[3]	15	10	0	0	25
	L[4]	23	25	0	1	49	L[4]	15	10	0	0	25
	L[5]	23	25	0	1	49	L[5]	15	10	0	0	25
	L[6]	23	23	0	1	47	L[6]	15	8	0	0	23
	L[7]	23	22	0	1	46	L[7]	15	7	0	0	22
	L[8]	23	25	0	1	49	L[8]	15	10	0	0	25
						391						**199**

30	L[1]	27	29	0	1	57	L[1]	19	10	0	0	29
	L[2]	27	28	0	1	56	L[2]	19	9	0	0	28
	L[3]	27	31	0	1	59	L[3]	19	12	0	0	31
	L[4]	27	29	0	1	57	L[4]	19	10	0	0	29
	L[5]	27	29	0	1	57	L[5]	19	10	0	0	29
	L[6]	27	29	0	1	57	L[6]	19	10	0	0	29
	L[7]	27	30	0	1	58	L[7]	19	11	0	0	30
						401						**205**
40	L[1]	31	37	0	1	69	L[1]	23	14	0	0	37
	L[2]	31	41	0	1	73	L[2]	23	18	0	0	41
	L[3]	31	38	0	1	70	L[3]	23	15	0	0	38
	L[4]	31	35	0	1	67	L[4]	23	12	0	0	35
	L[5]	31	34	0	1	66	L[5]	23	11	0	0	34
	L[6]	31	35	0	1	67	L[6]	23	12	0	0	35
	L[7]	31	39	0	1	71	L[7]	23	16	0	0	39
	L[8]	31	36	0	1	68	L[8]	23	13	0	0	36
	L[9]	31	37	0	1	69	L[9]	23	14	0	0	37
						620						**332**

The earlier calculations made are shown graphically below where the x-axis shows the number of nodes in various graphs and y-axis shows the total number of operations:



4.Conclusions and Future Directions

From the above study it can be concluded that both the methods employed to determine the fuzzy shortest path i.e. similarity measure method and Hamming distance method yield the same result. But the number of operations while calculating Hamming Distance is much less as compared to the number of operations needed while calculating Similarity Measure. As a result less computation time is needed when shortest path is to be determined with Hamming Distance Method.

Also the highest similarity value determines the shortest path whereas the least Hamming Distance value determines the shortest path.

This type of problem can also be extended to continuous types of fuzzy arc lengths such as trapezoidal type, L-R type and triangular fuzzy sets.

5.Applications of Shortest Path Problem

Shortest path algorithms are applied to automatically find directions between physical locations, such as driving directions on web mapping websites like MapQuest or Google Maps. For this application fast specialized algorithms are available.

If one represents a nondeterministic abstract machine as a graph where vertices describe states and edges describe possible transitions, shortest path algorithms can be used to find an optimal sequence of choices to reach a certain goal state, or to establish lower bounds on the time needed to reach a given state. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can be used to find a solution that uses the minimum possible number of moves.

In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and usually tied with a widest path problem. For example, the algorithm may seek the shortest (min-delay) widest path, or widest shortest (min-delay) path. Other applications include operations research, plant and facility layout, robotics, transportation, and VLSI design.[8]

Appendix(code)

Variables used:

- int size - to store the number of vertices in the graph
- v_node *adjlist – to store the graph in the form of adjacency list
- int queue[CONST] – queue data structure
- int front,rear - front and rear variables of the queue
- int source,destination – source and destination nodes of the graph
- int **pred - predecessor matrix to store the predecessor of each vertex
- int *p – array to keep track of the number of predecessors of a vertex in the predecessor matrix
- arr *head – to store all the extracted paths from source to destination from the predecessor matrix by backtracking
- arr *path – number the extracted paths and store them separately i.e.value 1 of the path array will point to the 1st extracted path
- int count – store the number of edges in the graph
- len_node *len – store the discrete fuzzy lengths of each path
- int m – store the number of possible paths from source to destination
- pd *head1 – store the predecessors which occur twice in the predecessor matrix of a vertex.Used for forming the paths while backtracking.
- node *min_len – store Lmin
- float *result – store the results of Hamming Distance

```
#include <iostream>
using namespace std;
#define MAX 4
#define CONST 100
class fuzzy_graph
{
public:
    class arr   //class to store all the possible paths between source and
      {         //destination
        public:
            int num;arr *left;arr(){left=NULL;}
            arr(int x){num=x;left=NULL;}
            ~arr(){}
      };
     struct flength  //structure to store the fuzzy length of an edge
      { float grade; int length;};
     class e_node    //class for the adjacent vertices of a vertex
      {
        public:
            int data;flength arr[MAX];e_node *next;e_node(){next=NULL;}
            e_node(int x){data =x;for(int i=0;i<MAX;i++){arr[i].length=0;}
                                  next=NULL;}
            ~e_node(){}
       };
      class v_node   //class for storing all vertices of the graph
      {
        public:
            int vertex;e_node *edge; v_node(){edge=NULL;}
            ~v_node(){}
```

```
      };
      class node   //class to store Lmin
      {
        public:
            int data;float y;node *next;node(){next=NULL;}
      };
      class len_node  //class for storing the discrete fuzzy lengths
      {               //of the possible paths
         public:
            int num;node *next;len_node(){next=NULL;}
      };
      class pd        //class to store the predecessors of a vertex
      {               //which occur twice in the predecessor matrix
        public:
            int number;int pred;pd *next;pd(){next=NULL;}
            pd(int x){number=x; next=NULL;}
            ~pd(){}
       };
      int size; v_node *adjlist; int queue[CONST];int front,rear;
      int source,destination; int **pred; int *p;arr *head;arr *path;
      int count; len_node *len;int m;pd *head;node *min_len;float *result;
      fuzzy_graph(int t)
      {size=t;count=0;front=rear=-1;
       adjlist=new v_node[t];
       for(int i=0;i<t;i++) {adjlist[i].vertex=(i+1);adjlist[i].edge=NULL;}
       p=new int[t];for(int i=0;i<size;i++){p[i]=0;}
       head=NULL; min_len=NULL; path=NULL; head1=NULL;
      }
    ~fuzzy_graph()
     {delete []adjlist;delete []p;}

 void make_list()  //function to store the adjacency list for storing the
      {int ver,arc;float member;   //graph
       cout << "\nEnter the vertices adjacent to each vertex.\nFor no
adjacent vertices or no more adjacent vertices type 0 .";
      cout << "\nFor each adjacent vertex type the fuzzy arc length.\nFor
stop giving the entries to the fuzzy arc length type -1";
      for(int i=0;i<size;i++)
      {e_node *temp1=adjlist[i].edge;int j=i+1;
       cout << "\nenter the adjacent vertices of vertex " << j<<" " ;
       cin >> ver;
       while(ver!= 0)
       {count++; e_node *ptr=new e_node(ver);
       cout << "\nNow enter its fuzzy arc lengths:";
       int k=0; cin >> arc;
       while(arc!=-1)
       {cin >> member; ptr->arr[k].length=arc;ptr->arr[k].grade=member;
        k++;cin >> arc;}
       if(adjlist[i].edge==NULL)
       {adjlist[i].edge=ptr;temp1=ptr;}
       else{temp1->next=ptr;temp1=ptr;}
       cout << "\nEnter the next adjacent vertex:";
       cin >> ver;}}
       pred=new int*[size];
       for(int i=0;i<size;i++){pred[i]=new int[count];}
       for(int i=0;i<size;i++){for(int j=0;j<count;j++){pred[i][j]=-1;}}
       cout << "\nThe adjacency list is as follows.."<<endl;
       for(int i=0;i<size;i++)
       {cout << adjlist[i].vertex << "-->";e_node *temp=adjlist[i].edge;
        while(temp!=NULL){cout << temp->data <<"-->";temp=temp->next;}
        cout << "\n";}
```

```
    cout << "\nEnter the source:";cin >> source;
    cout << "\nEnter the destination:"; cin >> destination;}

 bool isempty() //function to check if queue is empty or not
  {if(front==-1) return(true);else return(false);}

 void enqueue(int data) //function to enter the data in the queue
   {if(rear==CONST-1) {cout<<"Queue is full\n";}
    else{if(rear==-1)
          {rear=0;front=0;queue[rear]=data;}
        else{rear++;queue[rear]=data;}  }}

 int dequeue() //function to take out values from the queue
  {if(!isempty()){if(rear==front){int k=queue[front];rear=front=-1;
                                  return k;}
                 else{int k=queue[front];front++;return k;}
}
void search()  //function to find the predecessors of each vertex and
 {int x;enqueue(source);   // store it in the predecessor matrix
  while(!isempty())
  {x=dequeue();
   if(adjlist[x-1].edge!=NULL)
      {e_node *temp=adjlist[x-1].edge;
         while(temp!=NULL)
         {int y=temp->data;pred[y-1][p[y-1]]=x;
          if((y)!=destination){enqueue(y);}
          p[y-1]=p[y-1]+1;
          temp=temp->next;}}}
  }
 node *add1(flength *pt1, flength *pt2) //function to add two fuzzy sets
 {node *hd=NULL;
  for(int i=0; i<MAX && pt1[i].length !=0;i++)
   {for(int j=0; j<MAX && pt2[j].length!=0;j++)
     {node *pt=new node;pt->data=pt1[i].length + pt2[j].length;
      pt->y= min(pt1[i].grade , pt2[j].grade);
      pt->next=NULL;hd=insertsort(hd,pt);}}
   hd=del(hd);return(hd);
  }

 float min(float x,float y){return(x <= y ? x : y);}

 node *insertsort(node *hd,node *pt)  //function to sort the elements in
  {if(hd==NULL){hd=pt;return hd;}     //ascending order after addition
   else{node *temp1=NULL;node *temp=hd; //of two fuzzy sets
        while((pt->data > temp->data) && temp->next!=NULL)
         {temp1=temp;temp=temp->next;}
        if(pt->data > temp->data){temp->next=pt;}
        else{temp1->next=pt;pt->next=temp;}
   return(hd);}
   }
  node *add2(node *head,flength *pt2) //function to add the third fuzzy
   {node *temp=head;node *head1=NULL; //set to the result of addition
    while(temp!=NULL)                 //of two fuzzy sets
    {for(int i=0;i<3 && pt2[i].length!=0;i++)
      {node *ptr=new node;ptr->data=temp->data + pt2[i].length;
       ptr->y=min (temp->y , pt2[i].grade);
       ptr->next=NULL;head1=insert(head1,ptr);}
     temp=temp->next;delete(head);head=temp;}
     head1=del(head1);return(head1);
   }
float max(float x,float y){return(x >= y ? x : y);}
```

```
   node *insert(node *head1,node *bc)  //function to sort the elements
   {if(head1==NULL){head1=bc;return head1;} //in ascending order after
    else{node *temp1=NULL;node *temp=head1; //addition of two fuzzy sets
         while((bc->data > temp->data) && temp->next!=NULL)
          {temp1=temp;temp=temp->next;}
          if(bc->data > temp->data){temp->next=bc;}
          else{temp1->next=bc;bc->next=temp;}
    return head1;}
   }
   node *del(node *head1) //function to calculate the maximum of membership
   {node *temp1=head1;node *temp=head1->next;node *temp2=NULL; //grades
    while(temp !=NULL)                                  //of the same element
     {if(temp1->data != temp->data)                     //after addition of two
      {temp2=temp1;temp1=temp;temp=temp->next;}         //fuzzy sets
      else{if(temp1==head1)
           {float t=max(temp1->y,temp->y);
            if(t==temp1->y){temp1->next=temp->next;delete (temp);
                            temp=temp1->next;}
            else{delete(temp1);temp1=temp;head1=temp1;temp=temp->next;}
            }else{float t=max(temp1->y,temp->y);if(t==temp1->y)
                  {temp1->next=temp->next;delete (temp);temp=temp1->next;}
                  else{delete(temp1);temp2->next=temp;temp1=temp;
                       temp=temp->next;}}} }
   return(head1);
}
   void display() //function to display the paths and their fuzzy lengths
   {cout << "\nThe number of paths from "<<source << " to " <<destination
<< " are "<< p[destination-1];
    m=p[destination-1];len=new len_node[m];result=new float[m];
    path=new arr[m];
    for(int i=0;i<m;i++){len[i].num=(i+1);len[i].next=NULL;result[i]=0.0;
         path[i].num=(i+1);path[i].left=NULL;}
    for(int i=0;i<size;i++){p[i]=0;}
    if(pred[destination-1][0]==-1)return;
    cout << "\nThe paths from "<<source << " to " << destination << " are
:"<<endl;int t;pd *uv=NULL,*uv1=NULL;
   for(int i=0;i<size ;i++)
   {if(pred[i][1] != -1){while(pred[i][p[i]]!=-1)
     {if(pred[i][p[i]]==pred[i][p[i]+1] && pred[i][p[i]]!=pred[i][p[i]+2])
       {uv=new pd(i+1);uv->pred=pred[i][p[i]];
      if(head1==NULL){ head1=uv;}
      else{uv1=head1;while(uv->pred > uv1->pred && uv1->next!=NULL)
            {uv1=uv1->next;}
            if(uv->pred==uv1->pred){uv->next=uv1->next;uv1->next=uv;}
            else {uv1->next=uv;}}}
      p[i]=p[i]+1;}}}
  for(int i=0;i<size;i++){p[i]=0;}
  arr *temp1=NULL,*temp2=NULL,*temp3=NULL,*temp4=NULL;
  while(pred[destination-1][p[destination-1]]!=-1)
  {temp1=new arr(destination);if(head==NULL){head=temp1;}
   else{temp1->left=head;head=temp1;}
  int k=pred[destination-1][p[destination-1]];temp2=new arr(k);
  temp2->left=head;head=temp2;t=head->num;
  while(pred[t-1][p[t-1]]!=-1)
  {if(pred[t-1][p[t-1]]==source && pred[t-1][p[t-1]+1]==-1)
   {temp3=new arr(source);temp3->left=head;head=temp3;break;}

   else if(pred[t-1][p[t-1]]==source && pred[t-1][p[t-1]+1]!=-1)
    {temp3=new arr(source);temp3->left=head;head=temp3;
     if(head1!=NULL)
```

```
      {if(t==head1->pred && pred[destination-1][p[destination-1]]!=t)
       {pd *gh=head1->next;
        while(gh->number !=pred[destination-1][p[destination-1]+1]   && gh-
>pred==t && gh!=NULL){gh=gh->next;}
        if(gh!=NULL && gh->pred==t){p[t-1]=p[t-1];}
        else {p[t-1]=p[t-1]+1;}
        break;}}
        p[t-1]=p[t-1]+1;break;}

    else if(pred[t-1][1]==-1)
       {int ab=pred[t-1][p[t-1]];temp4=new arr(ab);temp4->left=head;
        head=temp4;t=head->num;}

    else if(pred[t-1][p[t-1]+1]==-1 )
       {int ab=pred[t-1][p[t-1]];temp4=new arr(ab);temp4->left=head;
        head=temp4;p[t-1]=0;t=head->num;}
    else
      {int ab=pred[t-1][p[t-1]];temp4=new arr(ab);temp4->left=head;
       head=temp4;if(head1!=NULL)
        {pd *find=head1;
         while(find->pred!=t || head->left->left->num!=find->number)
         {if(find->next!=NULL){find=find->next;}
          else{find=NULL;break;}}
          if(find!=NULL && find->number!=destination)
           {if(find->next->pred==t && find->next->number==destination)
            {p[t-1]=p[t-1]+1;}
           else if(find->next->pred==t && find->next->number!=destination)
              {p[t-1]=p[t-1];}
            else if(find->next->pred!=t && find->next->number==destination)
            {p[t-1]=p[t-1]+1;}
            else{p[t-1]=p[t-1]+1;}}
            else if(find==NULL){p[t-1]=p[t-1]+1;}
            else if(find!=NULL && find->number==destination)
            {p[t-1]=p[t-1]+1;}
            t=head->num;}
            else{p[t-1]=p[t-1]+1;t=head->num;}}}
           p[destination-1]=p[destination-1]+1;}

     arr *tempr=head;arr *tempr1=NULL;arr *pntr=NULL;
     while(tempr!=NULL){for(int i=0;i<m;i++)
        {tempr1=&path[i];
         while(tempr->num != destination)
         {int x=tempr->num;if(path[i].left==NULL)
            {pntr=new arr(x);path[i].left=pntr;tempr1=pntr;}
          else{pntr=new arr(x);tempr1->left=pntr;tempr1=pntr;}
          tempr=tempr->left;}
         pntr=new arr(destination);tempr1->left=pntr;tempr=tempr->left;}}

    arr *temp=head;int i=0,y;v_node *ptr=NULL;e_node *ptr1=NULL,*ptr2=NULL;
    while(temp!=NULL)
     {cout << temp->num<<"-->";int x=temp->num;ptr=&adjlist[x-1];
      temp=temp->left;if(temp->num == destination)
       {cout << temp->num << "\n";ptr1=ptr->edge;
        while(ptr1->data != destination){ptr1=ptr1->next;}
        node *h=NULL;for(int k=0;k<MAX && ptr1->arr[k].length != 0;k++)
         {node *w=new node;w->data=ptr1->arr[k].length;
          w->y=ptr1->arr[k].grade;w->next=NULL;if(h==NULL){h=w;}
          else{node *tr=h;while(tr->next !=NULL){tr=tr->next;}
          tr->next=w;}}
       len[i].next=h;
       cout << "The fuzzy arc length is L["<<len[i].num<<"]=";
```

```
    node *ht=len[i].next;while(ht!=NULL)
     {cout << ht->y << "/" << ht->data << "    ";ht=ht->next;}
    cout << "\n";i++;temp=temp->left;}
    else{cout<< temp->num<<"-->";y=temp->num;ptr1=ptr->edge;
         while(ptr1->data != y){ptr1=ptr1->next;}
         ptr=&adjlist[y-1];temp=temp->left;y=temp->num;ptr2=ptr->edge;
         while(ptr2->data!=y){ptr2=ptr2->next;}
         node *h=add1(ptr1->arr , ptr2->arr );
         if(y==destination)
          {cout << y << "\n";len[i].next=h;
           cout << "The fuzzy arc length is L["<<len[i].num<<"]=";
           node *ht=len[i].next;
           while(ht!=NULL)
               {cout << ht->y << "/" << ht->data << "    ";ht=ht->next;}
           cout <<"\n";i++;temp=temp->left;}
          else{cout << y << "-->";temp=temp->left;int az=temp->num;
              node *hf=h;
              while(az != destination)
              {cout << az << "-->";ptr=&adjlist[y-1];ptr2=ptr->edge;
               y=temp->num;while(ptr2->data !=az){ptr2=ptr2->next;}
               hf=add2(hf,ptr2->arr);temp=temp->left;az=temp->num;}
               cout << destination << "\n";ptr=&adjlist[y-1];
               ptr2=ptr->edge;y=temp->num;
               while(ptr2->data !=destination){ptr2=ptr2->next;}
               hf=add2(hf,ptr2->arr);temp=temp->left;
               len[i].next=hf;
               cout << "The fuzzy arc length is L["<<len[i].num<<"]=";
               node *ht=len[i].next;
               while(ht!=NULL)
               {cout << ht->y << "/" << ht->data << "    ";ht=ht->next;}
           cout << "\n\n";i++;}}}
 }
void min_length()           //function to calculate Lmin
{node *potr=NULL;node *tn=NULL;node *tn1=NULL;node *h=NULL;node*tp=NULL;
 float q,maximum;node *hp=len[0].next;
 while(hp!=NULL){q=hp->y;for(int j=1 ; j<m  ;j++)
           {h=len[j].next;maximum=0.0;
            while(h!=NULL && h->data < hp->data)
             {maximum=max(maximum,h->y);h=h->next;}
             q=min(q,(1.0-maximum));}
          potr=new node;potr->y=q;potr->data=hp->data;potr->next=NULL;
          if(min_len==NULL){min_len=potr;}
          else{tp=min_len;while(tp->next!=NULL){tp=tp->next;}
               tp->next=potr;}hp=hp->next;}
 for(int i=1;i<m;i++)
 {hp=len[i].next;while (hp!=NULL){q=hp->y;int k=0;
   while(k<m){if(k!=i){h=len[k].next;maximum=0.0;
                       while(h!=NULL && h->data < hp->data)
                       {maximum=max(maximum,h->y);h=h->next;}
                       q=min(q,(1.0-maximum));}
               k++;}
    tn=min_len;tn1=NULL;if(hp->data < tn->data)
   {potr=new node;potr->data=hp->data;potr->y=q;potr->next=min_len;
     min_len=potr;}
   else{while(hp->data > tn->data  && tn->next!=NULL)
         {tn1=tn;tn=tn->next;}
         if(tn->data == hp->data){tn->y=max(tn->y, q);}
         else if(tn->next==NULL)
         {potr=new node;potr->data=hp->data;potr->y=q;potr->next=NULL;
          tn->next=potr;}
         else{ potr=new node;potr->data=hp->data;potr->y=q;
```

```
                potr->next=tn;tn1->next=potr;}}
             hp=hp->next;}}
      cout << "\nThe fuzzy shortest length is.."<<endl;node *gh=min_len;
      while(gh !=NULL){if(gh->y !=0)
                       {cout << gh->y << "/" << gh->data << "\t";}
            gh=gh->next;}
    }
   float abso(float x1,float x2)  //function to return absolute value
     {if(x1 >= x2){return(x1-x2);} //of two floating point numbers
      else {return (x2 - x1);}
     }
  void hamming_distance()  //function to calculate hamming distance
  {node *q2=NULL;node *q3=NULL;int d;
   for(int i=0;i<m;i++)
    {float sum=0.0;q2=min_len;while(q2!=NULL)
      {q3=len[i].next;d=q2->data;if(d < q3->data ){sum=sum+q2->y;}
        else{while(q3->data != d && q3->next!=NULL){q3=q3->next;}
            if(q3->data==d && q3->next!=NULL){sum=sum + abso(q2->y,q3->y);}
            else if(q3->data==d && q3->next==NULL)
            {sum=sum+abso(q2->y,q3->y);}
            else if(q3->data!=d && q3->next==NULL){sum=sum+q2->y;}}
            q2=q2->next;}
            float w = sum; result[i] = w;int y=i+1;
             cout << "H(Lmin,L["<<y<<"])= " << result[i]<<endl;}
    float min=result[0];for(int i=1;i<m;i++)
       {if(result[i] < min){min=result[i];}}
       int y1;for(int i=0;i<m;i++){if(min==result[i]){y1=i;break;}}
     cout << "The lowest Hamming Distance to the fuzzy shortest length is
"<<min << " and this corresponds to the path "<<endl;
       arr *y2=path[y1].left;
       while(y2 !=NULL){cout << y2->num <<"\t";y2=y2->left;}
  }
};
void main()
{int z;cout << "\nEnter the number of vertices in the graph:";
 cin >>z;fuzzy_graph B(z);B.make_list();B.search();B.display();
 B.min_length();
 cout << "\nThe Hamming Distances are as follows.."<<endl;
 B.hamming_distance();
}
```

References

[1] H.Z.Zimmermann, Fuzzy Set Theory and its Applications, Allied Publishers Limited, India, 1996.
[2] A. Kaufmann, Introduction to Fuzzy Arithmetic, Van Nostrand Reinhold, New York, 1985.
[3] Tzung-Nan Chuang and Jung-Yuan Kung, A new algorithm for the discrete fuzzy shortest path problem in a network, Applied Mathematics and Computation 174 (2006) 660–668.
[4] M. Blue, B. Bush, J. Puckett, Unified approach to fuzzy graph problems, Fuzzy Sets Syst. 125 (2002) 355–368.
[5] L.K. Hyung, Y.S. Song, K.M. Lee, Similarity measure between fuzzy sets and between elements, Fuzzy Sets Syst. 62 (1994) 291–293.
[6] W.J. Wang, New similarity measures on fuzzy sets and on elements, Fuzzy Sets Syst. 85 (1997) 305–309.
[7] Vladimir Janis and Susana Montes, Distance Between Fuzzy Sets As A Fuzzy Quantity, Acta Universitatis Matthiae Belii ser. Mathematics, 14(2007), 41-49.
[8] www.Wikipedia.com