Maciej Medyk – COT6930 – Homework 3

Question 1

Network community [0.50pt].

Network community is a set of nodes within the network between which the interactions are relatively frequent and intense.

Link prediction [0.50pt].

Link predictions are methods and algorithms that allow to forecast which nodes are most likely to create a connection between them based on various similarity and correlation factors. There are three different algorithms for link predication called Distance Based Method, Collaborative Filtering Based Method, and Supervised Method.

Linear threshold influence model [0.50pt].

Linear threshold influence model is a diffusion model used to assess the influence of nodes within the network where nodes are either active or inactive and where over iterations an inactive node may become active as more neighbors collective influence reach the inactive node's threshold.

Independent cascade influence model [0.50pt].

Independent cascade influence model is a stochastic information diffusion model where information flows over the network through cascade. Nodes have are classified as inactive, active, or newly active. The difference to linear threshold model is that there is propagation probability by which one node can influence another node and each edge connecting two nodes has a value between 0 – 1 that expresses that probability.

How Facebook suggests friends for users [0.50pt].

Facebook network when user is using phone application is permitted to go through phone contacts on the smart phone and searches for users already on the network. Facebook also uses the profile information like where user went to school, where he was born, and what cities he lived in. All that information is used by their mapping algorithm that uses cliques between people you know and people your neighbors know based on similarity. At that moment Facebook can predict most likely people you may know.

Dewey, C. (2015). How Facebook knows who all your friends are, even better than you do. Washington: WP Company LLC. http://search.proquest.com.ezproxy.fau.edu/docview/1669416250?pq-origsite=summon&accountid=10902

Question 2

Please use item-based collaborative filtering to calculate Angelica's rating on item 3 (using Cosine distance). Please show your solution and the final matrix [1.00pt].

Users	Item 1	Item 2	Item 3	Item 4	Item 5
Angelica	3.50	2.00		4.50	5.00
Bill	2.00	3.50	4.00		2.00
Chan	5.00	1.00	1.00	3.00	5.00
Dan	3.00	4.00	4.50		3.00

Sim(3,1)	0.704354
Sim(3,2)	0.999742
Sim(3,4)	1.000000
Sim(3,5)	0.704354

Angelicas rating on item 3 is calculated as (1 / (0.7044 + 0.9997 + 1.0000 + 0.7044)) * ((3.50*0.7044) + (2.00 * 0.9997) + (4.50 * 1.0000) + (5.00 * 0.7044)) = 0.2934 * (2.4652 + 1.9995 + 4.5000 + 3.5218) = 3.6634

Please use user-based collaborative filtering to calculate Bill's rating on Item 4 (using Cosine distance). Please show your solution and the final matrix [1.00pt].

Users	Item 1	Item 2	Item 3	Item 4	Item 5	Mean
Angelica	3.50	2.00		4.50	5.00	3.75
Bill	2.00	3.50	4.00		2.00	2.88
Chan	5.00	1.00	1.00	3.00	5.00	3.00
Dan	3.00	4.00	4.50		3.00	3.63

Sim(2,1)	0.8304
Sim(2,3)	0.6334
Sim(2,4)	0.9922

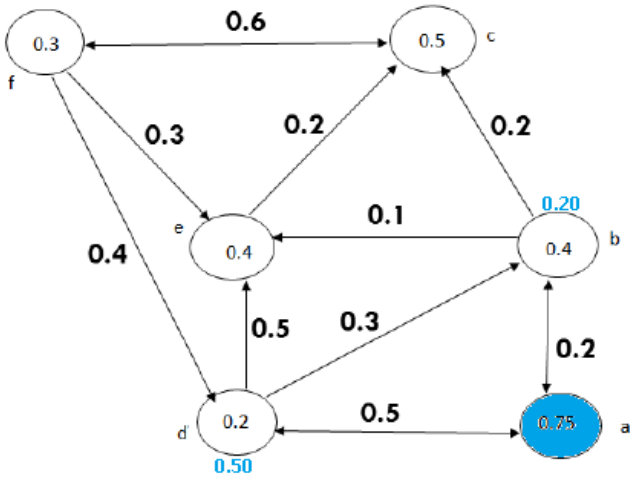
Bills rating on item 4 is calculated as 2.88 + 0.8304 (4.50 – 3.75) + 0.6334 (3.00 – 3.00) = 2.88 + 062 = 3.50

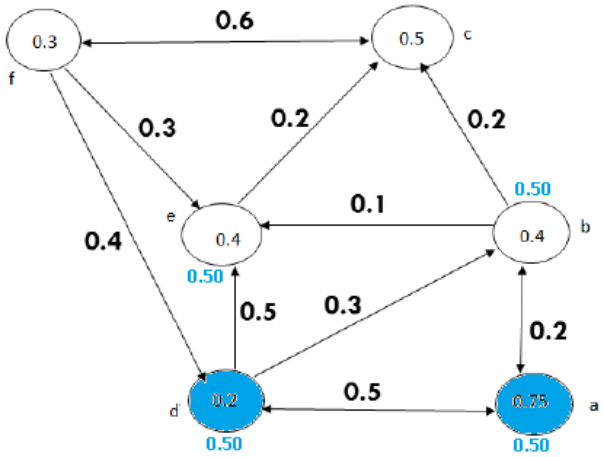
Question 3

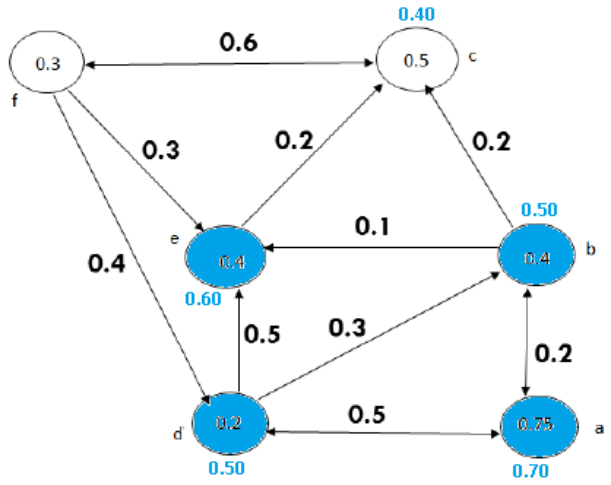
Assume node "a" is initially selected to be activated, please use Linear Threshold influence model to show (1) the order of the nodes which will be influenced [0.50pt] and the order of the nodes which will be activated [0.50pt].

Order of node influence : { B , D } , { A , B , E } , { A , B , C , E }

Order of node activation : { A } , { D } , { B , E }

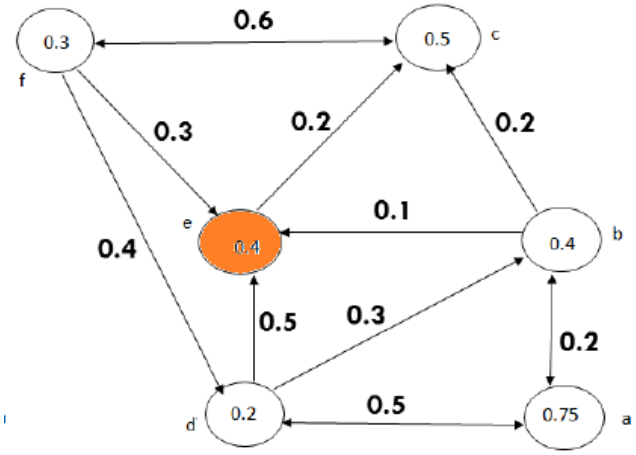






Please determine which node has the least influential power, and explain why [1.00pt].

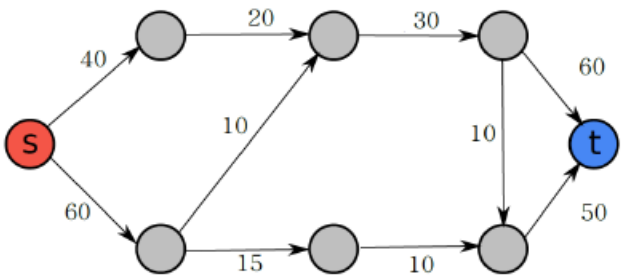
The least influential node is E as its only output is 0.2 and when node E activates it does contribute to activation of any other node.



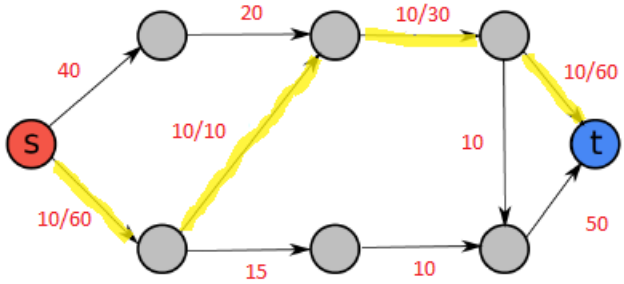
Question 4

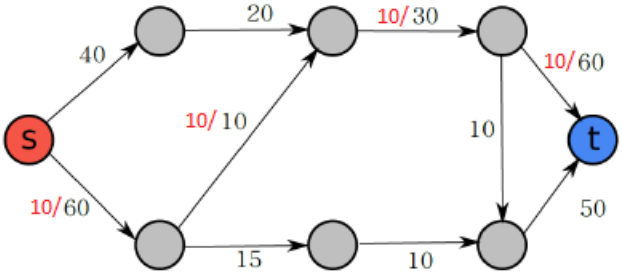
Please use Ford-Fulkerson algorithm to find the min-cut which separate the network into two community (one includes "s" and the other includes "t"). Please show your solution [2.00pt].

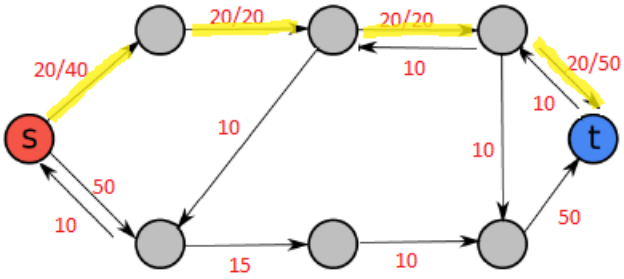
Flow Network

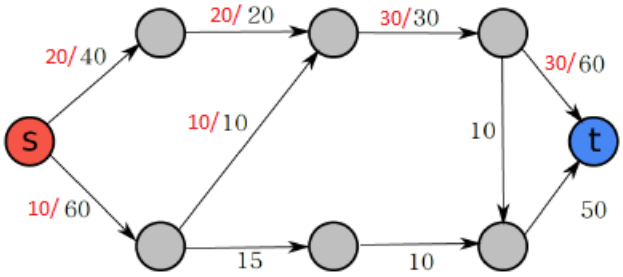


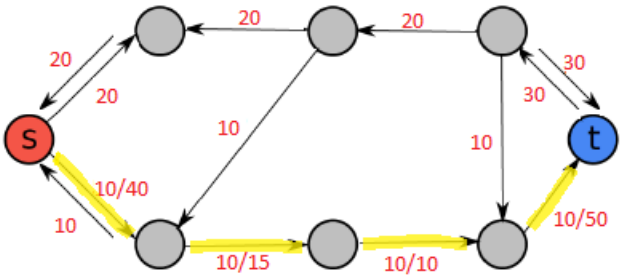
Residual Network

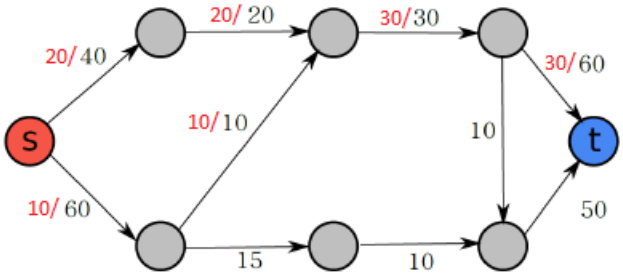


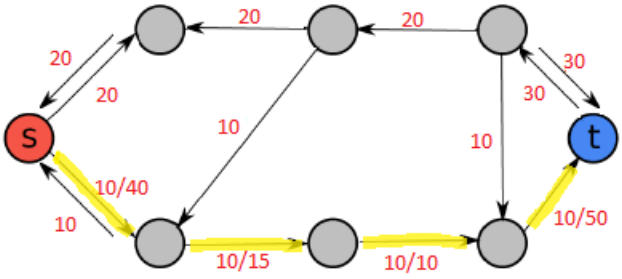


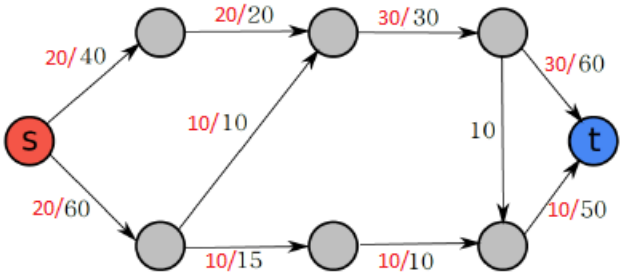


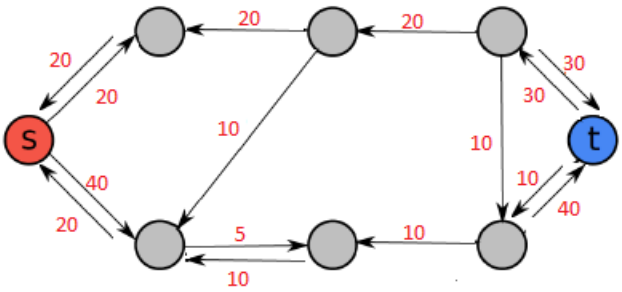






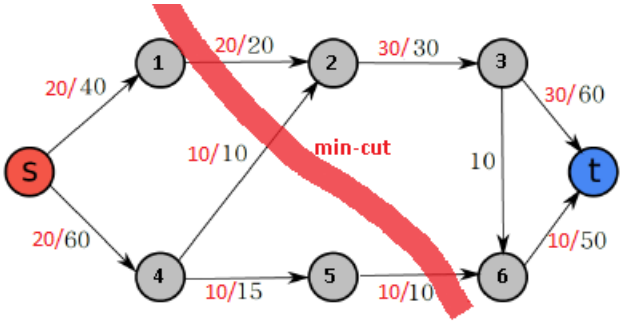






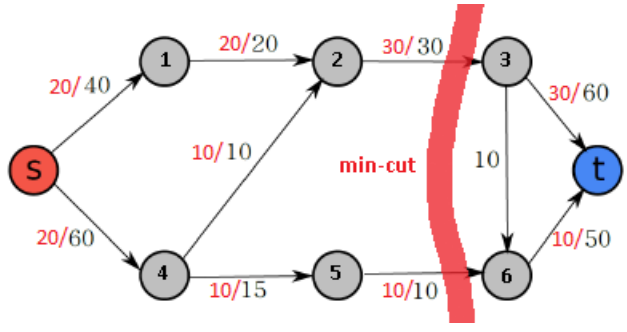
Minimum Cut

S = { 1 , 4 , 5 } o T = { 2 , 3 , 6 }



Minimum Cut

S = { 1 , 2 , 4 , 5 } o T = { 2 , 6 }



Question 5

Assume the influence value for all edges from member u to member v, i.e., (u,v), are equal to a, and the probability value for each member to influence his/her neighbor is p. We are now trying to select one representative from the network to broadcast the message, please use Independent Cascade model to determine which member should be selected such that the number of numbers can be reached from the selected representative is maximized (Please show your solution and report the results with respect to different parameter settings including p = {0.05 , 0.20 , 0.40}). Because Independent Cascade model uses a random process, please repeat the experiments for 10 times and calculate the average results for each node [2.00pt].

Result Report

Node	Prob	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	Run 9	Run 10	Average	Score
1	0.05	1	1	2	1	1	1	1	6	1	6	2.10	54.00
1	0.20	3	20	25	4	8	2	6	19	21	16	12.40	
1	0.40	23	28	25	25	32	7	24	26	27	15	23.20	
2	0.05	2	4	1	4	1	2	2	1	1	1	1.90	41.83
2	0.20	2	2	20	3	8	6	5	10	14	6	7.60	
2	0.40	27	31	26	29	24	3	3	24	29	2	19.80	
3	0.05	1	1	2	1	2	1	1	3	3	2	1.70	35.58
3	0.20	2	2	3	3	6	6	6	2	2	13	4.50	
3	0.40	24	11	29	26	21	28	19	23	17	3	20.10	
4	0.05	1	1	1	1	1	1	2	1	1	3	1.30	41.17
4	0.20	6	6	22	13	24	8	1	2	2	2	8.60	
4	0.40	22	24	2	24	27	26	24	22	24	23	21.80	
5	0.05	1	1	2	1	2	1	1	4	1	1	1.50	30.83
5	0.20	6	21	1	1	2	2	1	1	1	1	3.70	
5	0.40	5	2	2	2	29	30	33	30	19	24	17.60	
6	0.05	1	1	2	1	2	1	1	2	1	1	1.30	18.58
6	0.20	1	1	9	5	1	1	1	1	1	1	2.20	
6	0.40	31	2	2	2	2	2	2	29	2	1	7.50	
7	0.05	1	1	1	1	3	1	1	1	1	1	1.20	18.08
7	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
7	0.40	1	25	3	3	3	3	30	3	25	5	10.10	
8	0.05	1	1	3	1	1	1	1	1	1	1	1.20	27.08
8	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
8	0.40	27	24	29	6	24	24	25	29	3	18	20.90	
9	0.05	1	1	1	3	2	4	4	1	1	3	2.10	36.58
9	0.20	11	4	1	1	1	1	1	1	1	1	2.30	
9	0.40	30	1	24	24	23	25	23	28	22	25	22.50	
10	0.05	1	1	3	1	1	1	1	1	1	1	1.20	24.75
10	0.20	13	1	1	1	1	1	1	1	1	1	2.20	
10	0.40	30	21	1	1	1	28	20	25	1	29	15.70	
11	0.05	1	3	1	1	1	1	1	1	1	1	1.20	17.58
11	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
11	0.40	26	11	1	1	1	1	1	27	25	1	9.50	
12	0.05	1	1	1	1	1	1	5	1	1	1	1.40	16.75
12	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
12	0.40	1	1	24	1	8	30	1	1	1	1	6.90	

13	0.05	1	1	1	1	1	1	1	1	1	1	1.00	21.58
13	0.20	1	1	1	1	1	1	28	18	1	19	7.20	
13	0.40	1	1	1	1	1	1	1	1	1	26	3.50	
14	0.05	1	1	1	1	1	1	1	2	1	3	1.30	42.25
14	0.20	1	1	5	19	4	4	23	22	14	4	9.70	
14	0.40	22	29	32	27	23	23	27	23	2	1	20.90	
15	0.05	1	1	1	1	1	1	1	1	1	4	1.30	23.83
15	0.20	1	1	1	1	1	1	1	1	1	17	2.60	
15	0.40	1	1	1	1	26	29	23	23	1	24	13.00	
16	0.05	1	1	1	1	1	1	1	4	1	1	1.30	20.25
16	0.20	19	1	1	1	1	1	1	1	1	1	2.80	
16	0.40	30	26	20	1	1	1	1	1	1	1	8.30	
17	0.05	2	1	1	1	1	1	1	1	1	1	1.10	20.17
17	0.20	1	1	1	1	23	23	11	5	5	1	7.20	
17	0.40	1	1	1	1	1	1	1	1	1	1	1.00	
18	0.05	1	1	1	1	1	1	1	1	1	1	1.00	11.50
18	0.20	1	1	1	1	1	5	5	1	1	1	1.80	
18	0.40	1	1	1	1	1	1	1	1	7	7	2.20	
19	0.05	1	1	1	1	1	1	1	1	1	1	1.00	24.42
19	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
19	0.40	29	22	27	27	21	19	24	1	1	22	19.30	
20	0.05	1	1	1	1	1	1	1	4	1	1	1.30	23.58
20	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
20	0.40	1	1	1	7	26	23	26	21	24	29	15.90	
21	0.05	1	1	1	1	1	1	1	1	1	1	1.00	20.92
21	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
21	0.40	26	23	26	1	1	1	19	31	22	1	15.10	
22	0.05	1	1	1	1	1	1	1	1	1	1	1.00	24.67
22	0.20	1	1	1	1	1	1	1	1	1	5	1.40	
22	0.40	1	1	26	29	27	33	15	25	30	1	18.80	
23	0.05	1	1	1	1	1	1	1	1	1	1	1.00	16.00
23	0.20	8	1	1	1	24	1	1	1	1	1	4.00	
23	0.40	1	23	1	1	1	1	1	1	1	1	3.20	
24	0.05	1	1	1	1	1	1	1	1	1	1	1.00	40.50
24	0.20	9	21	12	1	1	1	18	20	11	15	10.90	
24	0.40	1	29	27	28	21	27	1	1	26	27	18.80	
25	0.05	1	1	1	1	1	1	1	1	1	1	1.00	20.75
25	0.20	4	1	1	1	1	1	1	1	1	1	1.30	
25	0.40	22	1	1	3	30	4	1	27	26	28	14.30	
26	0.05	2	1	1	1	3	1	1	1	2	1	1.40	31.50
26	0.20	1	16	21	1	1	1	1	1	1	1	4.50	
26	0.40	19	32	25	27	1	1	26	27	17	1	17.60	
27	0.05	1	1	1	1	1	1	1	1	1	1	1.00	17.58
27	0.20	1	1	1	1	1	1	1	1	1	1	1.00	
27	0.40	29	1	1	1	24	29	1	23	1	1	11.10	
28	0.05	1	1	1	1	2	3	1	6	1	1	1.80	41.25
28	0.20	21	1	1	16	2	22	18	2	2	2	8.70	
28	0.40	7	18	1	1	22	24	20	25	28	31	17.70	

29	0.05	1	1	1	1	1	1	1	5	1	1	1.40	29.25
29	0.20	4	1	1	12	1	1	1	1	17	1	4.00	
29	0.40	22	1	1	1	29	23	24	26	1	31	15.90	
30	0.05	1	1	2	1	1	4	2	1	1	1	1.50	25.00
30	0.20	1	1	1	1	1	2	2	2	2	11	2.40	
30	0.40	1	1	28	22	29	10	10	2	28	1	13.20	
31	0.05	1	1	1	1	1	1	1	1	1	1	1.00	15.67
31	0.20	6	1	1	1	1	1	1	1	1	1	1.50	
31	0.40	1	1	1	1	3	3	9	30	1	28	7.80	
32	0.05	3	4	1	1	1	3	1	2	1	1	1.80	39.33
32	0.20	1	1	1	1	21	1	1	1	1	1	3.00	
32	0.40	24	25	24	26	29	28	28	24	32	28	26.80	
33	0.05	3	1	1	1	1	1	2	1	1	1	1.30	32.67
33	0.20	1	1	1	1	1	14	12	1	1	1	3.40	
33	0.40	24	28	24	24	11	32	23	23	6	25	22.00	
34	0.05	2	2	6	1	6	2	1	1	1	1	2.30	55.08
34	0.20	3	3	22	8	4	12	16	5	24	16	11.30	
34	0.40	24	29	25	27	27	28	19	22	28	22	25.10	

Nodes in order of their influence. Score = (((X / 0.40) + (Y / 0.20) + (Z / 0.05)) / 3) ; where X is active node count at probability 0.40, Y is active node count at probability 0.20, and Z is active node count at probability of 0.05

Node	Score	Probability		
		0.40	0.20	0.05
34	55.08	25.10	11.30	2.30
1	54.00	23.20	12.40	2.10
14	42.25	20.90	9.70	1.30
2	41.83	19.80	7.60	1.90
28	41.25	17.70	8.70	1.80
4	41.17	21.80	8.60	1.30
24	40.50	18.80	10.90	1.00
32	39.33	26.80	3.00	1.80
9	36.58	22.50	2.30	2.10
3	35.58	20.10	4.50	1.70
33	32.67	22.00	3.40	1.30
26	31.50	17.60	4.50	1.40
5	30.83	17.60	3.70	1.50
29	29.25	15.90	4.00	1.40
8	27.08	20.90	1.00	1.20
30	25.00	13.20	2.40	1.50
10	24.75	15.70	2.20	1.20
22	24.67	18.80	1.40	1.00
19	24.42	19.30	1.00	1.00
15	23.83	13.00	2.60	1.30
20	23.58	15.90	1.00	1.30
13	21.58	3.50	7.20	1.00
21	20.92	15.10	1.00	1.00
25	20.75	14.30	1.30	1.00
16	20.25	8.30	2.80	1.30

17	20.17	1.00	7.20	1.10
6	18.58	7.50	2.20	1.30
7	18.08	10.10	1.00	1.20
11	17.58	9.50	1.00	1.20
27	17.58	11.10	1.00	1.00
12	16.75	6.90	1.00	1.40
23	16.00	3.20	4.00	1.00
31	15.67	7.80	1.50	1.00
18	11.50	2.20	1.80	1.00

Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace COT6930 Q5
{
    class Program
    {
        static void Main(string[] args)
        {
            Matrix matrix = new Matrix();
            List<double> prob = new List<double>();
            prob.Add(0.05);
            prob.Add(0.20);
            prob.Add(0.40);
            List<List<float>> list = matrix.CreateMatrix("ClassMatrix.txt", 1);
            matrix.DisplayMatrix(list);
            System.IO.StreamWriter filelong = new System.IO.StreamWriter("resultlong.txt");
            System.IO.StreamWriter fileshort = new System.IO.StreamWriter("resultshort.txt");
            Console.WriteLine("\n-------------------------------------------------------------------------");
            filelong.WriteLine("-------------------------------------------------------------------------");
            fileshort.WriteLine("-------------------------------------------------------------------------");
            for (int i = 0; i < prob.Count; i++)
            {
                Console.WriteLine("Probability   : " + prob[i].ToString());
                filelong.WriteLine("Probability   : " + prob[i].ToString());
                fileshort.WriteLine("Probability   : " + prob[i].ToString());
                Console.WriteLine("-------------------------------------------------------------------------");
                filelong.WriteLine("-------------------------------------------------------------------------");
                fileshort.WriteLine("-------------------------------------------------------------------------");
                list = matrix.CreateMatrix("ClassMatrix.txt", prob[i]);
                for (int j = 0; j < list.Count; j++)
                {
                    for (int k = 0; k < 10; k++) matrix.CascadeInfluence(list, j, k, filelong, fileshort);
                }
            }
            filelong.Close();
            fileshort.Close();
            Console.Write("\nPress any key to continue . . . ");
            Console.ReadKey(true);
        }

        public class Matrix
        {
            public List<List<float>> CreateMatrix(string filename, double probability)
            {
                var lines = File.ReadLines(filename);
                int seed = unchecked(DateTime.Now.Ticks.GetHashCode());

                Random random = new Random(seed);
                List<List<float>> result = new List<List<float>>();

                foreach (var line in lines)
                {
                    List<float> list = new List<float>();
                    for (int i = 0; i < line.Length; i++)
                    {
                        if (line[i] == '1') list.Add(1);
                        else if (line[i] == ' ') continue;
                        else list.Add((float)(Math.Round(0.00, 1)));
                    }
                    result.Add(list);
                }

                for (int i = 0; i < result.Count; i++)
                {
                    for (int j = i; j < result.Count; j++)
                    {
                        if (result[i][j] == 1)
```

```
                    {
                        float number = ((float) probability);
                        result[i][j] = number;
                        result[j][i] = number;
                    }
                }
            }
            return result;
        }
        public void CascadeInfluence(List<List<float>> matrix, int startnode, int runtime, System.IO.StreamWriter filelong,
System.IO.StreamWriter fileshort)
        {
            List<float> neighbors = CreateList(matrix.Count);
            List<float> oldactive = CreateList(matrix.Count);
            List<float> newactive = CreateList(matrix.Count);
            newactive[startnode] = 1;
            Console.WriteLine("Iteration Run : " + (runtime + 1).ToString() +  " => Initiation");
            filelong.WriteLine("Iteration Run : " + (runtime + 1).ToString() + " => Initiation");
            fileshort.WriteLine("Iteration Run : " + (runtime + 1).ToString() + " => Initiation");
            Console.WriteLine("Starting Node : " + (startnode + 1).ToString());
            filelong.WriteLine("Starting Node : " + (startnode + 1).ToString());
            fileshort.WriteLine("Starting Node : " + (startnode + 1).ToString());
            Console.WriteLine("-----------------------------------------------------------------");
            filelong.WriteLine("-----------------------------------------------------------------");
            fileshort.WriteLine("-----------------------------------------------------------------");
            do
            {
                Console.WriteLine("\nOld Active Nodes   : " + DisplayNodes(oldactive));
                filelong.WriteLine("\nOld Active Nodes   : " + DisplayNodes(oldactive));
                Console.WriteLine("New Active Nodes   : " + DisplayNodes(newactive));
                filelong.WriteLine("New Active Nodes   : " + DisplayNodes(newactive));
                neighbors = AddNeighbors(matrix, newactive, oldactive);
                Console.WriteLine("Inactive Neighbors : " + DisplayNodes(neighbors) + "\n");
                filelong.WriteLine("Inactive Neighbors : " + DisplayNodes(neighbors) + "\n");
                List<List<float>> result = Cascade(neighbors, newactive, oldactive, matrix, filelong);
                oldactive = result[0];
                newactive = result[1];
            } while (DisplayNodes(neighbors) != "None");

            int count = 0;
            for (int i = 0; i < oldactive.Count; i++) if (oldactive[i] > 0) count++;
            Console.WriteLine("Count : " + count.ToString());
            filelong.WriteLine("Count : " + count.ToString());
            fileshort.WriteLine("Count : " + count.ToString());
            Console.WriteLine("-----------------------------------------------------------------");
            filelong.WriteLine("-----------------------------------------------------------------");
            fileshort.WriteLine("-----------------------------------------------------------------");
            Console.WriteLine("Iteration Run : " + (runtime + 1).ToString() + " => Complete");
            filelong.WriteLine("Iteration Run : " + (runtime + 1).ToString() + " => Complete");
            Console.WriteLine("-----------------------------------------------------------------\n");
            filelong.WriteLine("-----------------------------------------------------------------\n");
            Console.WriteLine("-----------------------------------------------------------------");
            filelong.WriteLine("-----------------------------------------------------------------");
        }
        public List<float> CreateList(int count)
        {
            List<float> result = new List<float>();
            for (int i = 0; i < count; i++) result.Add(0);
            return result;
        }
        private List<float> AddNeighbors(List<List<float>> matrix, List<float> newactive, List<float> oldactive)
        {
            List<float> neighbors = CreateList(matrix.Count);
            for (int j = 0; j < matrix.Count; j++)
            {
                if (newactive[j] > 0)
                {
                    for (int k = 0; k < matrix.Count; k++)
                    {
                        if (neighbors[k] == 0) neighbors[k] = matrix[j][k];
                        if (newactive[k] > 0) neighbors[k] = 0;
                        if (oldactive[k] > 0) neighbors[k] = 0;
                    }
                }
            }
            return neighbors;
        }
        private List<float> GetNeighbors(List<List<float>> matrix, int index)
        {
            List<float> neighbors = CreateList(matrix.Count);
            for (int j = 0; j < matrix.Count; j++)
            {
                neighbors[j] = matrix[index][j];
            }
            return neighbors;
        }
        public void DisplayMatrix(List<List<float>> matrix)
        {
            for(int i = 0; i < matrix.Count; i++)
            {
                for(int j = 0; j < matrix.Count; j++)
                {
                    if(matrix[i][j] > 0) Console.Write("1 ");
                    else Console.Write("0 ");
```

```
                }
                Console.Write("\n");
            }
        }
        private string DisplayNodes(List<float> list)
        {
            string result = "";
            for (int i = 0; i < list.Count; i++) if (list[i] > 0) result += (i + 1).ToString() + ", ";
            if (result.Length < 3) return "None";
            return result.Substring(0, result.Length - 2);
        }
        private List<List<float>> Cascade(List<float> neighbors, List<float> newact, List<float> oldact, List<List<float>> matrix,
System.IO.StreamWriter filelong)
        {
            List<List<float>> result = new List<List<float>>();
            List<float> tempactive = CreateList(matrix.Count);
            int seed = unchecked(DateTime.Now.Ticks.GetHashCode());
            Random random = new Random(seed);
            for (int k = 0; k < matrix.Count; k++) if (newact[k] > 0) oldact[k] = newact[k];

            for (int k = 0; k < matrix.Count; k++)
            {
                if (newact[k] > 0)
                {
                    neighbors = GetNeighbors(matrix, k);
                    for (int i = 0; i < matrix.Count; i++)
                    {
                        if (neighbors[i] > 0 && oldact[i] == 0)
                        {
                            float roll = (float)Math.Round(random.NextDouble(), 2);
                            if (roll <= matrix[k][i])
                            {
                                Console.WriteLine("Node " + SpaceOut((k + 1), (newact.Count.ToString()).Length) + " -> " +
SpaceOut((i + 1), (newact.Count.ToString()).Length) + " => " + DisplayNeighborValue(matrix[k][i]) + " => Successful ");
                                filelong.WriteLine("Node " + SpaceOut((k + 1), (newact.Count.ToString()).Length) + " -> " +
SpaceOut((i + 1), (newact.Count.ToString()).Length) + " => " + DisplayNeighborValue(matrix[k][i]) + " => Successful ");
                                tempactive[i] = 1;
                            }
                            else
                            {
                                Console.WriteLine("Node " + SpaceOut((k + 1), (newact.Count.ToString()).Length) + " -> " +
SpaceOut((i + 1), (newact.Count.ToString()).Length) + " => " + DisplayNeighborValue(matrix[k][i]) + " => Unsuccessful ");
                                filelong.WriteLine("Node " + SpaceOut((k + 1), (newact.Count.ToString()).Length) + " -> " +
SpaceOut((i + 1), (newact.Count.ToString()).Length) + " => " + DisplayNeighborValue(matrix[k][i]) + " => Unsuccessful ");
                            }
                        }
                    }
                }
            }
            newact = tempactive;
            result.Add(oldact);
            result.Add(newact);
            return result;
        }
        private string SpaceOut(int number, int numbermax)
        {
            string result = "";
            for (int i = 0; i < (numbermax - ((number).ToString()).Length); i++) { result += "0"; }
            string n = number.ToString();
            result = result + n;
            return result;
        }

        private string DisplayNeighborValue(double value)
        {
            if (value == 0) return "0.0";
            else if (value == 1) return "1.0";
            else return (string)Math.Round(value, 2).ToString();
        }
    }
  }
}
```