Faculty of computer and Information Science – ASU

Analysis & Design of Algorithms

T164 – Small World Phenomenon

Name	ID	DEP	Section
احمد محمد فريد	20191700873	SC	**5**
احمد هاني حامد	20191700875	CS	**6**
اسراء حربي سعد	20191700876	SC	**5**

Graph. cs

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;

namespace small_word_final
{
    class Graph
    {
        Dictionary<string, int> dist, vis, dist2; //o(1)
        Dictionary<string, string> par, mov; //O(1)
        Dictionary<string, int> strength; //O(1)
        Dictionary<string, List<KeyValuePair<string, string>>>
friend; // O(1)
        Dictionary<Tuple<string, string>, int> act_movie; //O(1)
        int id = 0;
        int numOfMovies = 0;
        public Graph(List<string> vertices,
List<KeyValuePair<string, KeyValuePair<string, string>>> edges)
        {
            //objects
            dist = new Dictionary<string, int>(); //O(1)
            dist2 = new Dictionary<string, int>();//O(1)
            par = new Dictionary<string, string>();//O(1)
            mov = new Dictionary<string, string>();//O(1)
            vis = new Dictionary<string, int>();//O(1)
            strength = new Dictionary<string, int>();//O(1)
            friend = new Dictionary<string,
List<KeyValuePair<string, string>>>();//O(1)
            act_movie = new Dictionary<Tuple<string, string>,
int>();//O(1)

            for (int i = 0; i < vertices.Count; i++) //O(V)
            {
                friend.Add(vertices[i], new
List<KeyValuePair<string, string>>());
                dist.Add(vertices[i], 0);
                dist2.Add(vertices[i], 0);
                par.Add(vertices[i], "");
                mov.Add(vertices[i], "");
                vis.Add(vertices[i], 0);
                strength.Add(vertices[i], 0);

            }
            //add edge between 2 actors
            foreach (var edge in edges) //o(e)
            {
                string movie = edge.Key;
                string actor1 = edge.Value.Key;
                string actor2 = edge.Value.Value;
```

```
                // undirected gragh
                friend[actor1].Add(new KeyValuePair<string,
string>(actor2, movie));
                friend[actor2].Add(new KeyValuePair<string,
string>(actor1, movie));
//to save the movies between 2 actors
                Tuple<string, string> act2_moive = new Tuple<string,
string>(actor2, actor1);
                Tuple<string, string> act3_moive = new Tuple<string,
string>(actor1, actor2);
                if (act_movie.ContainsKey(act2_moive))
                {
                    act_movie[act2_moive]++;
                }

                else if (act_movie.ContainsKey(act3_moive))
                {
                    act_movie[act3_moive]++;
                }
                else
                {
                    act_movie.Add(act2_moive, 1);
                }

            }
        } //o(V)+o(E) = O(V)
        // to make the value of dictionaries equal 0 to relation
        public void es(List<string> vertices) //O(V)
        {
            dist = new Dictionary<string, int>();
            dist2 = new Dictionary<string, int>();
            par = new Dictionary<string, string>();
            mov = new Dictionary<string, string>();
            vis = new Dictionary<string, int>();
            strength = new Dictionary<string, int>();


            for (int i = 0; i < vertices.Count; i++)
            {

                dist.Add(vertices[i], 0);
                dist2.Add(vertices[i], 0);
                par.Add(vertices[i], "");
                mov.Add(vertices[i], "");
                vis.Add(vertices[i], 0);
                strength.Add(vertices[i], 0);

            }
        } //O(V)
```

```
        //Dijkstra
        public void Dijkstra(string actor1, string actor2)
        {
            id++;
            vis[actor1] = id;
            dist[actor1] = 0;

            // first = distance , second = actor
            SortedSet<KeyValuePair<int, string>> set = new
SortedSet<KeyValuePair<int, string>>(new MyComparer());
            set.Add(new KeyValuePair<int, string>(0, actor1));
            // loop untill set become empty

            while (set.Count > 0)// o(v)
            {
                int d = 0;
                string node = "";
                // loop for one time to get the fisrt element in set
to start from it
                foreach (var elem in set)//O(1)
                {
                    d = elem.Key;
                    node = elem.Value;
                    set.Remove(elem);
                    break;
                }
                //if the value that sorted in node equal actore ,
then we calulate the shortest path betwwn them
                if (node.Equals(actor2))
                    break;
                // to limit time only
                if (d != dist[node])
                    continue;
                //loop for all niebour of node
                foreach (var niebour in friend[node])//o(E)
                {
                    // to update the shortest path -- the distance
that updated put it in set
                    if (par[niebour.Key] == "")
                    {
                        dist2[niebour.Key] = dist2[node] + 1; //O(1)
                    }
                    if (dist[niebour.Key] > d + 1 ||
vis[niebour.Key] != id)
                    {
                        dist[niebour.Key] = d + 1; //O(1)
                        par[niebour.Key] = node; //O(1)
                        mov[niebour.Key] = niebour.Value; // to
store the name of movie //O(1)
                        vis[niebour.Key] = id; //O(1)
                        set.Add(new KeyValuePair<int, string>(d + 1,
niebour.Key)); //O(1)
```

```
                    }
                    // to updata relation
                    Tuple<string, string> ecc = new Tuple<string,
string>(node, niebour.Key);
                    Tuple<string, string> ecc2 = new Tuple<string,
string>(niebour.Key, node);
                    if (act_movie.ContainsKey(ecc))
                    {
                        numOfMovies = act_movie[ecc] +
strength[node]; //O(1)
                        if (numOfMovies > strength[niebour.Key] &&
dist2[node] < dist2[niebour.Key]) //O(1)
                            strength[niebour.Key] = numOfMovies;
//O(1)
                    }
                    else
                    {
                        numOfMovies = act_movie[ecc2] +
strength[node]; //O(1)
                        if (numOfMovies > strength[niebour.Key] &&
dist2[node] < dist2[niebour.Key]) //O(1)
                            strength[niebour.Key] = numOfMovies;
//O(1)
                    }
                }
            }


            //output-- deg.

            if (vis[actor2] == id)
                Console.Write(dist[actor2] + "\t");

            //rel
            Console.Write(" " + strength[actor2]);
            // chain
            List<string> shortestPath = new List<string>();
            string current = actor2;
            while (current != actor1)
            {
                shortestPath.Add(mov[current]);
                current = par[current];
            }

            //output--chain
            shortestPath.Reverse();

            Console.Write("\t" + shortestPath[0]);
            for (int i = 1; i < shortestPath.Count; i++)
                Console.Write(" => " + shortestPath[i]);

        } // O(V.E)
    }
}
```

Program. Cs

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


namespace small_word_final
{

    class Program
    {
        static String sample_case(string x)
        {
            if (x == "sample")
            {
                string moviepath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Sample\\mo
vies1.txt";
                return moviepath;
            }
            else
            {
                string querypath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Sample\\qu
eries1.txt";
                return querypath;
            }
        }
        static String complete_case(string size, string x)
        {
            if (size == "small")
            {
                if (x == "m")
                {
                    string moviepath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
small\\Case1\\Movies193.txt";
                    return moviepath;
                }
                else
                {
                    string querypath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
small\\Case1\\queries110.txt";
```

```
                    return querypath;
                }
            }
            else if (size == "medium")
            {
                if (x == "m")
                {
                    string moviepath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
medium\\Case1\\Movies967.txt";
                    return moviepath;
                }
                else
                {
                    string querypath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
medium\\Case1\\queries4000.txt";
                    return querypath;
                }
            }
            else if (size == "large")
            {
                if (x == "m")
                {
                    string moviepath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
large\\Movies14129.txt";
                    return moviepath;
                }
                else
                {
                    string querypath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
large\\queries26.txt";
                    return querypath;
                }
            }
            else if (size == "extreme")
            {
                if (x == "m")
                {
                    string moviepath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
extreme\\extreme\\Movies122806.txt";
                    return moviepath;
                }
```

```
                else
                {
                    string querypath =
"C:\\Users\\aaaaa\\source\\repos\\test5\\Testcases\\Complete\\
extreme\\queries22.txt";
                    return querypath;
                }
            }
            else
                return null;

        }
        static void Main(string[] args)
        {

            HashSet<string> vertHash = new HashSet<string>();
            List<KeyValuePair<string, KeyValuePair<string,
string>>> edges = new List<KeyValuePair<string,
KeyValuePair<string, string>>>();

            int t;
            string x = "sample";
            Console.Write("Enter 0 for exit, 1 for run the
code \n");
            t = Convert.ToInt32(Console.ReadLine());

            Console.Write("Enter Type of data
(sample/small/medium/large/extreme): \n");
            x = Console.ReadLine();

            string moviepath = "";
            string querypath = "";
            while (t != 0)
            {
                if (x == "sample")
                {
                    moviepath = sample_case(x);
                    querypath = sample_case("q");
                }
                if (x == "small")
                {
                    moviepath = complete_case(x, "m");
                    querypath = complete_case(x, "q");
                }
                if (x == "medium")
                {
                    moviepath = complete_case(x, "m");
                    querypath = complete_case(x, "q");
                }
```

```
                if (x == "large")
                {
                    moviepath = complete_case(x, "m");
                    querypath = complete_case(x, "q");
                }
                if (x == "extreme")
                {
                    moviepath = complete_case(x, "m");
                    querypath = complete_case(x, "q");
                }
                var watch = new
System.Diagnostics.Stopwatch();
                watch.Start();
                string[] lines = File.ReadAllLines(moviepath);
                foreach (string line in lines)//o(E)
                {
                    string[] arr = line.Split('/');
                    for (int i = 1; i < arr.Length; i++)
//O(1)
                    {
                        vertHash.Add(arr[i]);
                        for (int j = i + 1; j < arr.Length;
j++)
                        {
                            edges.Add(new KeyValuePair<string,
KeyValuePair<string, string>>(arr[0], new KeyValuePair<string,
string>(arr[i], arr[j])));

                        }
                    }
                }

                Graph graph = new Graph(new
List<string>(vertHash), edges);

////////////////////////////////////////////////////////////
/////
                string[] querylines =
File.ReadAllLines(querypath);
                Console.Write("Deg. \t Rel. \t Chain\n");
                foreach (string queryline in querylines)
//O(E)
                {
                    string[] actors = queryline.Split('/');
                    graph.es(new List<string>(vertHash));
                    graph.Dijkstra(actors[0], actors[1]);
                    Console.WriteLine();

                }
```

```
                Console.WriteLine($"Execution Time:
{watch.ElapsedMilliseconds} ms");


                Console.Write("Enter 0 for exit, 1 for Run
Code \n");
                t = Convert.ToInt32(Console.ReadLine());
                Console.Write("Enter Type of data
(sample/small/medium/large): \n");
                x = Console.ReadLine();
                watch = new System.Diagnostics.Stopwatch();
                watch.Start();
            }

        }
    }
}
```

My Comparer.cs

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

namespace small_word_final
{
    public class MyComparer : IComparer<KeyValuePair<int, string>>
    {
        public int Compare(KeyValuePair<int, string> fn1,
KeyValuePair<int, string> fn2)
        {


            return fn1.Key == fn2.Key ?
fn1.Value.CompareTo(fn2.Value) : fn1.Key.CompareTo(fn2.Key);

        }
    }
}
```

Function	Complexity
Dijkstra	O(V.E)
Es	O(V)
sample_case	O(1)
complete_case	O(1)

The complexity of all project = O(V.E)

Test case	Time
Sample	No time
Small	No time
Medium	32944ms = 0.54 min
Large	150000ms = 2.5 min
Extreme	174000ms = 2.9 min