**ActorEdge()**

Class ActorEdges to carry information about the edge that connects 2 actors together:

-From  the source actor.

-To  the destination actor.

-Movie  the movie that 2 actors appeared in.

-Edgecost  the edge cost.

class ActorsEdges

{

public string from; public string to; public string movie; public int Edgecost;

public ActorsEdges(string f, string t, string m)

{

from = f; to = t; movie = m;

Edgecost = 1;

}

}

# Class ReadData()

This class to read the data from the files

ReadSample: to read the movies data.  O(Movies\*(Line Actor^2)) ReadQueries: to read the test queries.  O(queries\*(AdjList))

class ReadData

{

public static Dictionary<string, List<ActorsEdges>> adj =

new Dictionary<string, List<ActorsEdges>>(); //O(1)

public static Dictionary<string, int> sharedMovies =

new Dictionary<string, int>(); //O(1)

public List<string> actors = new List<string>();

public void ReadSample(int option) //O(movies\*(actors^2))

{

string filename =

@"C:\Users\green\Desktop\SmallWorldPhenomenon\small\Case1\Movies193.txt"; //O(1)

string movie = ""; //O(1)

using (StreamReader sr = File.OpenText(filename))

{

string line = String.Empty;

while ((line = sr.ReadLine()) != null)

{

string[] subs = line.Split('/');

movie = subs[0];

for (int i = 1; i < subs.Length; i++) //O(subs.Length)

{

actors.Add(subs[i]); //O(1)

}

for (int i = 0; i < actors.Count; i++) //O( line->actors^2 )

{

if (!adj.ContainsKey(actors[i])) //O(1)

{

adj.Add(actors[i], new List<ActorsEdges>());

}

for (int j = 0; j < actors.Count; j++) //O(actors)

{

if (i != j) //O(1)

{

ActorsEdges AE =

new ActorsEdges(actors[i], actors[j], movie);//O(1)

adj[actors[i]].Add(AE);

string stest = actors[i] + actors[j];//O(1)

string stest2 = actors[j] + actors[i];//O(1)

if (sharedMovies.ContainsKey(stest) &&

sharedMovies.ContainsKey(stest2))

{

sharedMovies[stest]++;//O(1)

sharedMovies[stest2]++;//O(1)

}

else

{

sharedMovies.Add(stest, 1);

sharedMovies.Add(stest2, 1);

}

}

}

}

actors = new List<string>();

}

}

Console.WriteLine("Done Reading Movie File!"); //O(1)

if (option == 3)

{

BuildGraph BG = new BuildGraph(adj, sharedMovies); //O(1)

BG.Bonuse();

}

}

public void ReadQueries(int opt) //O(queries\*(AdjList^2))

{

string filename =

@"C:\Users\green\Desktop\SmallWorldPhenomenon\small\Case1\queries110.txt";//O(1)

using (StreamReader sr = File.OpenText(filename))

{

string line = String.Empty;

while ((line = sr.ReadLine()) != null)

{

string[] subs = line.Split('/'); //O(1)

Console.WriteLine(); //O(1)

BuildGraph BG = new BuildGraph(adj, sharedMovies); //O(1)

BG.CalculateDeg(subs[0], subs[1], opt);//O(AdjList)

}

}

Console.WriteLine("done reading queries"); //O(1)

}

}

}

# Class BuildGraph()

## Constructor for initializing :

AdjList, VertexInfo, InfoMatrix, visited

public BuildGraph(Dictionary<string, List<ActorsEdges>> adj) //O(1)

{

AdjList = adj;

NodeData = new Dictionary<string, NodeInfo>();//O(1)

SHAREDMOVIES = sharedMovies; //O(1)

visited = new int[5000, 5000];

}

# Function CalculateDeg()  O(AdjList^2)

Calls

BFS()  O(AdjList) BuildChain()  O(AdjList)

public void CalculateDeg(string actor1, string actor2, int opt, Dictionary<string, int> sharedMovies) //O(AdjList^2)

{

Console.Write(actor1 + "/" + actor2); //O(1)

NodeInfo res = BFS(actor1, actor2, opt); //O(AdjList)

Console.Write("\t " + res.deg + " \t \t "); //O(1)

Console.Write(res.rel + " \t"); //O(1)

BuildChain(actor1, actor2); //O(movieChain)}

# Function BuildChain()  O(AdjList)

Print the Chain between 2 Actors

public void BuildChain(string actor1, string actor2)

{

//O(AdjList)

Stack<string> movieChain = new Stack<string>(); //O(1)

string test = actor2; //O(1)

while (test != actor1)

{

//O(AdjList)

movieChain.Push(InfoMatrix[test].Value);

test = InfoMatrix[test].Key;

}

int i = 0;

foreach (var element in movieChain) //O(AdjList)

{

i++;

if (i == movieChain.Count)

{

Console.Write(element);

}

else

Console.Write(element + " -> ");

}

Console.WriteLine();

}

# Function BFS()  O(AdjList)

## Calculates the Degree Of Separation and Relation Strength of the destination actor and returns it.

public NodeInfo BFS(string actor1, string actor2, int opt)

{

NodeInfo ni = new NodeInfo(0, 0, " ", " ");

NodeData.Add(actor1, ni);

Queue<ActorsEdges> pq = new Queue<ActorsEdges>();

pq.Enqueue(new ActorsEdges("", actor1, "")); //O(1)

while (pq.Count != 0) //O(AdjList)

{

ActorsEdges edge = pq.Peek(); //O(1)

if (edge.to == actor2 && opt == 2) //O(1)

{

return NodeData[actor2];

}

pq.Dequeue(); //O(1)

int f = 0; int t = 0, temp = 0;

foreach (var c in edge.from) //O(actor1.Length)

{

temp = (int)c;//O(1)

char x1 = 'A', x2 = 'Z', x3 = 'a', x4 = 'z';//O(1)

if ((temp >= (int)x1 && temp <= (int)x2) || (temp >= (int)x3

&& temp <= (int)x4))

{

f += temp;//O(1)

}

}

foreach (var c in edge.to)//O(actor2.Length)

{

temp = (int)c;//O(1)

char x1 = 'A', x2 = 'Z', x3 = 'a', x4 = 'z';//O(1)

if ((temp >= (int)x1 && temp <= (int)x2) || (temp >= (int)x3

&& temp <= (int)x4))

{

t += temp;//O(1)

}

}

if (visited[f, t] == 1 || visited[t, f] == 1)//O(1)

{

continue;

}

else

{

visited[f, t] = 1;//O(1)

visited[t, f] = 1;//O(1)

}

for (int i = 0; i < AdjList[edge.to].Count; i++)//O(AdjList[edge.to])

{

ActorsEdges neighbour = AdjList[edge.to][i];//O(1)

if (!NodeData.ContainsKey(neighbour.to))

{

ni = new NodeInfo(int.MaxValue, -1, " ", " ");//O(1)

NodeData.Add(neighbour.to, ni);

}

if (NodeData[edge.to].deg + neighbour.Edgecost <

NodeData[neighbour.to].deg)

{

int moviesCount = 0;//O(1)

string s = edge.to + neighbour.to;//O(1)

moviesCount = SHAREDMOVIES[s] / 2;//O(1)

NodeData[neighbour.to] = new NodeInfo(NodeData[edge.to].deg +

neighbour.Edgecost, NodeData[edge.to].rel + moviesCount, neighbour.from,

neighbour.movie);//O(1)

}

else if (NodeData[edge.to].deg + neighbour.Edgecost ==

NodeData[neighbour.to].deg)

{

int moviesCount = 0;//O(1)

string s = edge.to + neighbour.to;//O(1)

moviesCount = SHAREDMOVIES[s] / 2;//O(1)

if (NodeData[edge.to].rel + moviesCount >

NodeData[neighbour.to].rel)

{

NodeData[neighbour.to] =

new NodeInfo(NodeData[neighbour.to].deg, NodeData[edge.to].rel + moviesCount,

neighbour.from, neighbour.movie);//O(1)

}

}

pq.Enqueue(neighbour);//O(1)

}

}

if (opt == 3) { return NodeData[actor1]; } //O(1)

return NodeData[actor2]; //O(1)

}

Total  O(queries\*(AdjList))

# Function Bonus()  O(AdjList)

## Calculate the distribution of the degree of separation between a given actor and all other actors.

Print the strongest path.

public void Bonuse() //O(AdjList^2)

{

string src, dest = ""; //O(1)

int maxrs = -1; //O(1)

int[] frequancy = new int[13]; //O(1)

frequancy[0] = 1; //O(1)

Console.WriteLine("Enter Actor name: "); //O(1)

src = Console.ReadLine(); //O(1)

BFS(src, "", 3); //O(AdjList^2)

for (int index = 0; index < NodeData.Count; index++) //O(VertexInfo.Count)

{

var item = NodeData.ElementAt(index); //<string , NodeInfo>

var actor = item.Key; //string

var deg = item.Value.deg; //int deg

var rs = item.Value.rel; //int rs

int dos = deg;

if (dos < 12) frequancy[dos]++;

else frequancy[12]++;

if (rs > maxrs)

{

maxrs = rs;

dest = actor;

}

}

Console.WriteLine("Deg. of Separ. \t Frequency.");

Console.WriteLine("--------------------------------------");

for (int i = 0; i < 13; i++) //O(1)

{

//print distribution of the degree of separation

if (i == 12) Console.WriteLine(">" + (i - 1) + " \t\t\t " +

(frequancy[i]));

else Console.WriteLine(i + "\t\t\t " + frequancy[i]);

}

//print The strongest path (based on the relation strength)

BuildChain(src, dest); //O(AdjList)

Console.WriteLine("The strongest path (based on the relation strength): " +

maxrs);

//Console.ReadLine();

}