**Small World Phenomenon project**

**Team Number: T167**

**Members:**

Raafat Medhat Ramzy Eskander – 20191700241- CS

Rehab Mahmoud Youssef Ali – 20191700246 - IS

Romany Moner Sameh Abd Allah – 20191700257 - CS

**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\*(Actor^2))

ReadQueries: to read the test queries. 🡪 O(queries\*(AdjList^2))

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>(); //O(1)

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

{

string filename = @"C:\Users\green\Desktop\SmallWorldPhenomenon\Algorithms-Project-main\small\Case1\Movies193.txt"; //O(1)

var lines = File.ReadLines(filename); //O(1)

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

foreach (var line in lines) //O(Movies\*(actors^2)) //Lines

{

string fileLine = (string)line; //O(1)

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

movie = subs[0]; //O(1)

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( 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);

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

string stest = actors[i] + actors[j];

string stest2 = actors[j] + actors[i];

if(sharedMovies.ContainsKey(stest) && sharedMovies.ContainsKey(stest2))

{

sharedMovies[stest]++;

sharedMovies[stest2]++;

}else

{

sharedMovies.Add(stest, 1);

sharedMovies.Add(stest2, 1);

}

}

}

}

actors = new List<string>(); //O(1)

}

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

if (option == 3)

{

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

BG.Bonuse();

}

}

public void ReadQueries(int opt) //O(queries)

{

string filename = @"C:\Users\green\Desktop\SmallWorldPhenomenon\Algorithms-Project-main\small\Case1\queries110.txt"; //O(1)

var lines = File.ReadLines(filename); //O(1)

string actor1, actor2; //O(1)

Console.WriteLine("Query \t Degree \t RS \t Chain"); //O(1)

foreach (var line in lines) //O(queries) //lines

{

string fileLine = (string)line; //O(1)

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

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

actor1 = subs[0]; //O(1)

actor2 = subs[1]; //O(1)

try

{

List<ActorsEdges> t1 = adj[actor1]; //O(1)

List<ActorsEdges> t2 = adj[actor2]; //O(1)

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

BG.CalculateDeg(actor1, actor2,opt, sharedMovies);

}catch

{

Console.WriteLine("The Entered Actor1 neither Actor2 Doesn't Exist!! "); //O(1)

}

}

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;

VertexInfo = new Dictionary<string, KeyValuePair<int, int>>();

InfoMatrix = new Dictionary<string, KeyValuePair<string, string>>();

visited = new Dictionary<KeyValuePair<string, string>, bool>();

}

**Function CalculateDeg() 🡪 O(AdjList^2)**

Calls

Dijkstra() 🡪 O(AdjList^2)

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)

SHAREDMOVIES = sharedMovies; //O(1)

KeyValuePair<int, int> res = Dijkstra(actor1, actor2, opt); //O(AdjList^2)

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

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

BuildChain(actor1, actor2); //O(AdjList)

}

**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 Dijkstra() 🡪 O(AdjList^2)**

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

public KeyValuePair<int, int> Dijkstra(string actor1, string actor2, int opt) //O(AdjList^2)

{

VertexInfo.Add(actor1, new KeyValuePair<int, int>(0, 0)); //O(1)

prioqueue pq = new prioqueue(); //O(1)

pq.Enqueue(new ActorsEdges("", actor1, ""), 0); //O(1)

while(!pq.IsEmpty()) //O(AdjList)

{

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

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

string to = edge.to; //O(1)

string from = edge.from; //O(1)

KeyValuePair<string, string> k1 = new KeyValuePair<string, string>(to, from); //O(1)

KeyValuePair<string, string> k2 = new KeyValuePair<string, string>(from, to); //O(1)

if (visited.ContainsKey(k1) && visited.ContainsKey(k2)) //O(1)

{

continue;

}

else //O(1)

{

visited.Add(new KeyValuePair<string, string>(to, from), true);

visited.Add(new KeyValuePair<string, string>(from, to), true);

}

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

{

ActorsEdges neighbour = AdjList[edge.to][i];

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

{

VertexInfo.Add(neighbour.to, new KeyValuePair<int, int>(int.MaxValue, -int.MaxValue));

}

if (VertexInfo[edge.to].Key + neighbour.Edgecost < VertexInfo[neighbour.to].Key) //O(1)

{

int moviesCount = 0;

string s = edge.to + neighbour.to;

moviesCount = SHAREDMOVIES[s] / 2;

VertexInfo[neighbour.to] = new KeyValuePair<int, int>(VertexInfo[edge.to].Key + neighbour.Edgecost , VertexInfo[edge.to].Value + moviesCount );

if(InfoMatrix.ContainsKey(neighbour.to))

{

InfoMatrix[neighbour.to] = new KeyValuePair<string, string>(neighbour.from, neighbour.movie);

}else

{

InfoMatrix.Add(neighbour.to, new KeyValuePair<string, string>(neighbour.from, neighbour.movie));

}

}else if(VertexInfo[edge.to].Key + neighbour.Edgecost == VertexInfo[neighbour.to].Key)

{

int moviesCount = 0;

string s = edge.to + neighbour.to;

moviesCount = SHAREDMOVIES[s] / 2;

if (VertexInfo[edge.to].Value + moviesCount > VertexInfo[neighbour.to].Value)

{

VertexInfo[neighbour.to] = new KeyValuePair<int, int>(VertexInfo[neighbour.to].Key, VertexInfo[edge.to].Value + moviesCount);

if (InfoMatrix.ContainsKey(neighbour.to))

{

InfoMatrix[neighbour.to] = new KeyValuePair<string, string>(neighbour.from, neighbour.movie);

}

else

{

InfoMatrix.Add(neighbour.to, new KeyValuePair<string, string>(neighbour.from, neighbour.movie));

}

}

}

pq.Enqueue(neighbour, VertexInfo[neighbour.to].Key); //O(1)

}

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

{

return VertexInfo[actor2];

}

}

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

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

}

**Function Bonus() 🡪 O(AdjList^2)**

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)

this.Dijkstra(src, "", 3); //O(AdjList^2)

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

{

var item = VertexInfo.ElementAt(index); //<string , <int , int >>

var actor = item.Key; //string

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

var rs = item.Value.Value; //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();

}

Total 🡪 O(queries\*(AdjList^2))