## Source code

### Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SmallWorld

{

    internal class Program

    {

        private static void Main(string[] args)

        {

            Console.ForegroundColor = ConsoleColor.Cyan;

            Console.WriteLine("Welcome to Small World!");

            Console.ResetColor();

            while (true)

            {

                Console.WriteLine("1. Normal");

                Console.WriteLine("2. Optimization");

                Console.ForegroundColor = ConsoleColor.Red;

                Console.WriteLine("3. Exit");

                Console.ForegroundColor = ConsoleColor.DarkYellow;

                Console.Write("Choice: ");

                string mode = Console.ReadLine();

                Console.ResetColor();

                if (mode == "1")

                    Normal.Run();

                else if (mode == "2")

                    Optimization.Run();

                else if (mode == "3")

                    break;

                else

                {

                    Console.ForegroundColor = ConsoleColor.Red;

                    Console.WriteLine("Invalid mode");

                    Console.ResetColor();

                }

            }

        }

    }

}

### SmallWorld.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

using System.IO;

namespace SmallWorld

{

    class Optimization

    {

        private static string moviesPath = "", queriesPath = "", solutionPath = "";

        private static List<string> queries = new List<string>();

        private static Queue<string> answers = new Queue<string>();

        private static Dictionary<int, Dictionary<int, int>> common = new Dictionary<int, Dictionary<int, int>>();

        private static Dictionary<string, int> index = new Dictionary<string, int>();

        private static List<string> moviesNames = new List<string>();

        private static List<List<int>> movies = new List<List<int>>();

        private static List<string> actorsName = new List<string>();

        private static List<HashSet<int>> adjs = new List<HashSet<int>>();

        private static List<List<int>> actorsMovies = new List<List<int>>();

        static bool sample;

        private static int MAX\_DEGREE\_FOUND = 0;

        private static int[] dosFrequency;

        public static void Run()

        {

            sample = false;

            SelectTestCase();

            Console.WriteLine("Test case selected");

            Console.WriteLine("Parsing...");

            ParseMovies();

            Console.WriteLine("Parsing done");

            ChooseOpeartion();

            ClearAll();

            Console.ReadLine();

        }

        private static void ChooseOpeartion()

        {

            string Choice;

            do

            {

                Console.ForegroundColor = ConsoleColor.Green;

                Console.WriteLine("1.Find Shortest Path (Completed)");

                Console.ForegroundColor = ConsoleColor.Green;

                Console.WriteLine("2.Find Strongest Path (Completed)");

                Console.ForegroundColor = ConsoleColor.Green;

                Console.WriteLine("3.Find Degree of Seperation Frequency (Completed)");

                Console.ForegroundColor = ConsoleColor.DarkYellow;

                Console.WriteLine("4.Find Minimum Movies that link all actors/actresses (Pending)");

                Console.ResetColor();

                Console.Write("Enter your choice: ");

                Choice = Console.ReadLine();

                if ((Choice == "1" || Choice == "2" || Choice == "3" || Choice == "4"))

                    break;

                else

                {

                    Console.ForegroundColor = ConsoleColor.Red;

                    Console.WriteLine("Invalid Choice!");

                    Console.ResetColor();

                }

            } while (true);

            switch (Choice)

            {

                case "1":

                    ParseQueries();

                    ParseSolutions();

                    RunTestCase();

                    break;

                case "2":

                    FindStrongestPath();

                    break;

                case "3":

                    CalculateFrequency();

                    break;

                case "4":

                    Console.ForegroundColor = ConsoleColor.DarkYellow;

                    Console.WriteLine("Not implemented yet");

                    Console.ResetColor();

                    break;

            }

        }

        private static void ClearAll()

        {

            queries.Clear();

            answers.Clear();

            index.Clear();

            movies.Clear();

            moviesNames.Clear();

            actorsName.Clear();

            adjs.Clear();

            actorsMovies.Clear();

            common.Clear();

        }

        private static void SelectTestCase()

        {

            string path = @"..\..\..\Testcases\";

            string choice;

            do

            {

                Console.WriteLine("0.Sample");

                Console.WriteLine("-Complete");

                Console.WriteLine(" -small");

                Console.WriteLine("\t1-Case1");

                Console.WriteLine("\t2-Case2");

                Console.WriteLine(" -medium");

                Console.WriteLine("  -Case1");

                Console.WriteLine("\t3.queries85");

                Console.WriteLine("\t4.queries4000");

                Console.WriteLine("  -Case2");

                Console.WriteLine("\t5.queries110");

                Console.WriteLine("\t6.queries2000");

                Console.WriteLine(" -large");

                Console.WriteLine("\t7.queries26");

                Console.WriteLine("\t8.queries600");

                Console.WriteLine(" -extreme");

                Console.WriteLine("\t9.queries22");

                Console.WriteLine("\t10.queries200");

                Console.Write("Select Test Case: ");

                choice = Console.ReadLine();

                if (!(choice == "0" || choice == "1" || choice == "2" || choice == "3" || choice == "4" || choice == "5" || choice == "6" || choice == "7" || choice == "8" || choice == "9" || choice == "10"))

                {

                    Console.ForegroundColor = ConsoleColor.Red;

                    Console.WriteLine("Invalid Choice");

                    Console.ResetColor();

                }

                else

                    break;

            }

            while (true);

            switch (choice)

            {

                case "0":

                    path += @"Sample\";

                    moviesPath = path + "movies1.txt";

                    queriesPath = path + "queries1.txt";

                    solutionPath = path + "queries1 - Solution.txt";

                    sample = true;

                    break;

                case "1":

                    path += @"Complete\small\Case1\";

                    moviesPath = path + "Movies193.txt";

                    queriesPath = path + "queries110.txt";

                    solutionPath = path + @"\Solution\queries110 - Solution.txt";

                    break;

                case "2":

                    path += @"Complete\small\Case2\";

                    moviesPath = path + "Movies187.txt";

                    queriesPath = path + "queries50.txt";

                    solutionPath = path + @"\Solution\queries50 - Solution.txt";

                    break;

                case "3":

                    path += @"Complete\medium\Case1\";

                    moviesPath = path + "Movies967.txt";

                    queriesPath = path + "queries85.txt";

                    solutionPath = path + @"\Solutions\queries85 - Solution.txt";

                    break;

                case "4":

                    path += @"Complete\medium\Case1\";

                    moviesPath = path + "Movies967.txt";

                    queriesPath = path + "queries4000.txt";

                    solutionPath = path + @"\Solutions\queries4000 - Solution.txt";

                    break;

                case "5":

                    path += @"Complete\medium\Case2\";

                    moviesPath = path + "Movies4736.txt";

                    queriesPath = path + "queries110.txt";

                    solutionPath = path + @"\Solutions\queries110 - Solution.txt";

                    break;

                case "6":

                    path += @"Complete\medium\Case2\";

                    moviesPath = path + "Movies4736.txt";

                    queriesPath = path + "queries2000.txt";

                    solutionPath = path + @"\Solutions\queries2000 - Solution.txt";

                    break;

                case "7":

                    path += @"Complete\large\";

                    moviesPath = path + "Movies14129.txt";

                    queriesPath = path + "queries26.txt";

                    solutionPath = path + @"\Solutions\queries26 - Solution.txt";

                    break;

                case "8":

                    path += @"Complete\large\";

                    moviesPath = path + "Movies14129.txt";

                    queriesPath = path + "queries600.txt";

                    solutionPath = path + @"\Solutions\queries600 - Solution.txt";

                    break;

                case "9":

                    path += @"Complete\extreme\";

                    moviesPath = path + "Movies122806.txt";

                    queriesPath = path + "queries22.txt";

                    solutionPath = path + @"\Solutions\queries22 - Solution.txt";

                    break;

                case "10":

                    path += @"Complete\extreme\";

                    moviesPath = path + "Movies122806.txt";

                    queriesPath = path + "queries200.txt";

                    solutionPath = path + @"\Solutions\queries200 - Solution.txt";

                    break;

            }

        }

        private static void ParseMovies()

        {

            StreamReader reader = new StreamReader(moviesPath);

            while (true)

            {

                string line = reader.ReadLine();

                if (line == null)

                    break;

                string movie\_name = line.Substring(0, line.IndexOf('/'));

                List<int> movieActors = new List<int>();

                while (true)

                {

                    line = line.Remove(0, (line.IndexOf('/') != -1) ? line.IndexOf('/') + 1 : line.Length);

                    if (line.Length == 0)

                        break;

                    string actor\_name = (line.IndexOf('/') != -1) ? line.Substring(0, line.IndexOf('/')) : line;

                    if (!index.ContainsKey(actor\_name))

                    {

                        int newActorIndex = actorsName.Count();

                        index[actor\_name] = newActorIndex;

                        movieActors.Add(newActorIndex);

                        actorsMovies.Add(new List<int>());

                        adjs.Add(new HashSet<int>());

                        actorsMovies[newActorIndex].Add(moviesNames.Count());

                        actorsName.Add(actor\_name);

                    }

                    else

                    {

                        int prevActorIndex = index[actor\_name];

                        movieActors.Add(prevActorIndex);

                        actorsMovies[prevActorIndex].Add(moviesNames.Count());

                    }

                }

                movies.Add(new List<int>());

                movies[moviesNames.Count()].AddRange(movieActors);

                moviesNames.Add(movie\_name);

            }

            foreach (var movie in movies)

            {

                foreach (var actor in movie)

                {

                    if (!common.ContainsKey(actor))

                        common[actor] = new Dictionary<int, int>();

                    foreach (var adj in movie)

                    {

                        if (adj != actor)

                        {

                            if (!common[actor].ContainsKey(adj))

                                common[actor][adj] = 0;

                            adjs[actor].Add(adj);

                            common[actor][adj]++;

                        }

                    }

                }

            }

            dosFrequency = new int[index.Count()];

            reader.Close();

        }

        private static void ParseQueries()

        {

            StreamReader reader = new StreamReader(queriesPath);

            while (true)

            {

                string line = reader.ReadLine();

                if (line == null)

                    break;

                queries.Add(line);

            }

        }

        private static void ParseSolutions()

        {

            StreamReader reader = new StreamReader(solutionPath);

            if (!sample)

            {

                List<string> answer = new List<string>();

                while (true)

                {

                    string line = reader.ReadLine();

                    if (line == null)

                        break;

                    if (line == "")

                    {

                        string ans;

                        ans = answer[0] + answer[1] + answer[2] + answer[3];

                        answers.Enqueue(ans);

                        answer.Clear();

                    }

                    else

                    {

                        answer.Add(line + "\n");

                    }

                }

            }

            else

            {

                while (true)

                {

                    string line = reader.ReadLine();

                    if (line == null)

                        break;

                    answers.Enqueue(line);

                }

            }

        }

        private static void RunTestCase()

        {

            int current = 1;

            int passed = 0;

            if (sample)

                Console.WriteLine(answers.Dequeue());

            Stopwatch stopwatch = new Stopwatch();

            stopwatch.Start();

            foreach (string query in queries)

            {

                string first = query.Substring(0, query.IndexOf('/'));

                string second = query.Substring(query.IndexOf('/') + 1);

                string answer = Solve(first, second);

                if (answers.Count() == 0)

                {

                    Console.WriteLine(answer);

                }

                else

                {

                    Console.ForegroundColor = ConsoleColor.DarkCyan;

                    Console.Write("Query {0}/{1} ", current, queries.Count);

                    Console.ResetColor();

                    string actual = answers.Dequeue();

                    if (answer != actual)

                    {

                        Console.WriteLine("+=======================+");

                        Console.WriteLine("Query {0} failed: {1}/{2}", current, first, second);

                        Console.WriteLine("Expected:\n {0}", actual);

                        Console.WriteLine("Actual:\n {0}", answer);

                        Console.ForegroundColor = ConsoleColor.Red;

                        Console.Write($" Failed\n");

                        Console.ResetColor();

                    }

                    else

                    {

                        passed++;

                        Console.ForegroundColor = ConsoleColor.Green;

                        Console.Write($" Passed\n");

                        Console.ResetColor();

                        Console.WriteLine(answer);

                    }

                    current++;

                }

            }

            stopwatch.Stop();

            Console.ForegroundColor = ConsoleColor.DarkYellow;

            Console.Write($"Optimized Time taken:  {stopwatch.ElapsedMilliseconds / 1000.0} Seconds | {stopwatch.ElapsedMilliseconds} Milliseconds");

            Console.ResetColor();

        }

        private static string Solve(string src, string dest)

        {

            int iFirst = index[src];

            int iSecond = index[dest];

            List<int> shortestPath = FindShortestPath(iFirst, iSecond);

            if (shortestPath != null)

                return GenerateAnswer(shortestPath);

            else

                return "";

        }

        private static string GenerateAnswer(List<int> path)

        {

            int degree = 0, relation = 0;

            Stack<string> chainOfActors = new Stack<string>();

            Stack<string> chainOfMovies = new Stack<string>();

            string answer = "";

            string commonMovies = "";

            if (!sample) answer += $"{actorsName[path.Last()]}/{actorsName[path.First()]}\n";

            if (path.Count() != 1)

            {

                for (int i = 0; i < path.Count() - 1; i++)

                {

                    List<int> movies = GetIntersection(actorsMovies[path[i]], (actorsMovies[path[i + 1]]));

                    chainOfActors.Push(actorsName[path[i]]);

                    if (!sample)

                        chainOfMovies.Push(moviesNames[movies[0]]);

                    else

                    {

                        commonMovies = "";

                        foreach (var m in movies)

                        {

                            if (commonMovies == "")

                                commonMovies += moviesNames[m];

                            else

                                commonMovies += moviesNames[m].Substring(moviesNames[m].Count() - 1);

                            if (m != movies.Last())

                                commonMovies += " or ";

                        }

                        chainOfMovies.Push(commonMovies);

                    }

                    relation += movies.Count;

                    degree++;

                }

            }

            else

            {

                relation += actorsMovies[path[0]].Count();

            }

            if (!sample)

            {

                answer += $"DoS = {degree}, RS = {relation}\n";

                answer += $"CHAIN OF ACTORS: {actorsName[path.Last()]} -> ";

                while (chainOfActors.Count() != 0)

                {

                    answer += chainOfActors.Pop();

                    if (chainOfActors.Count() != 0)

                        answer += " -> ";

                }

                answer += "\n";

                answer += "CHAIN OF MOVIES:  =>";

                while (chainOfMovies.Count() != 0)

                {

                    answer += " " + chainOfMovies.Pop();

                    answer += " =>";

                }

                answer += "\n";

            }

            else

            {

                answer += $"{actorsName[path.Last()]}/{actorsName[path.First()]}\t\t{degree}\t\t{relation}\t\t";

                while (chainOfMovies.Count() != 0)

                {

                    answer += chainOfMovies.Pop();

                    if (chainOfMovies.Count() != 0)

                        answer += " => ";

                }

            }

            return answer;

        }

        private static List<int> FindShortestPath(int start, int end)

        {

            int[] parent = new int[index.Count()];

            int[] weight = new int[index.Count()];

            bool[] visited = new bool[index.Count()];

            int maxDist = int.MaxValue;

            int[] dist = new int[index.Count()];

            bool found = false;

            if (start == end)

            {

                parent[end] = -1;

                return ConstructPath(parent, end);

            }

            else

            {

                Queue<int> q = new Queue<int>();

                q.Enqueue(start);

                parent[start] = -1;

                weight[start] = 0;

                dist[start] = 0;

                while (q.Count != 0)

                {

                    int u = q.Dequeue();

                    if (visited[u])

                        continue;

                    if (dist[u] + 1 > maxDist)

                        break;

                    foreach (int v in adjs[u])

                    {

                        if (v == start)

                            continue;

                        if (visited[v])

                            continue;

                        if (dist[v] == 0 || dist[v] > dist[u] + 1)

                        {

                            int commonWithU = common[v][u] + weight[u];

                            dist[v] = dist[u] + 1;

                            q.Enqueue(v);

                            parent[v] = u; weight[v] = commonWithU;

                        }

                        else if (dist[v] == dist[u] + 1)

                        {

                            int commonWithU = common[v][u] + weight[u];

                            int commonWithParent = common[v][parent[v]] + weight[parent[v]];

                            if (commonWithU > commonWithParent)

                            {

                                parent[v] = u;

                                weight[v] = commonWithU;

                            }

                        }

                        if (v == end)

                        {

                            found = true;

                            maxDist = dist[v];

                        }

                    }

                    visited[u] = true;

                }

                if (found == false)

                    return null;

                return ConstructPath(parent, end);

            }

        }

        private static void GetKnown(int src)

        {

            dosFrequency[0] = 1;

            int[] parent = new int[index.Count()];

            int[] weight = new int[index.Count()];

            bool[] visited = new bool[index.Count()];

            int maxDist = int.MaxValue;

            int[] dist = new int[index.Count()];

            dist = Enumerable.Repeat(int.MaxValue, index.Count()).ToArray();

            Queue<int> q = new Queue<int>();

            q.Enqueue(src);

            parent[src] = -1;

            weight[src] = 0;

            dist[src] = 0;

            while (q.Count != 0)

            {

                int u = q.Dequeue();

                if (visited[u])

                    continue;

                if (dist[u] + 1 > maxDist)

                    break;

                foreach (int v in adjs[u])

                {

                    if (dist[v] > dist[u] + 1)

                    {

                        int commonWithU = common[v][u] + weight[u];

                        if (dist[v] != int.MaxValue && dosFrequency[dist[v]] != 0)

                            dosFrequency[dist[v]]--;

                        dist[v] = dist[u] + 1; dosFrequency[dist[v]]++;

                        if (MAX\_DEGREE\_FOUND < dist[v])

                            MAX\_DEGREE\_FOUND = dist[v];

                        q.Enqueue(v);

                        weight[v] = 0;

                        parent[v] = u;

                        weight[v] = commonWithU;

                    }

                    else if (dist[v] == dist[u] + 1)

                    {

                        int commonWithU = common[v][u] + weight[u];

                        int commonWithParent = common[v][parent[v]] + weight[parent[v]];

                        if (commonWithU > commonWithParent)

                        {

                            parent[v] = u; weight[v] = commonWithU;

                        }

                    }

                }

                visited[u] = true;

            }

        }

        private static List<int> GetIntersection(List<int> first, List<int> second)

        {

            int fIter = 0;

            int sIter = 0;

            var result = new List<int>();

            while (fIter < first.Count && sIter < second.Count)

            {

                if (first[fIter] < second[sIter])

                {

                    fIter++;

                }

                else if (first[fIter] > second[sIter])

                {

                    sIter++;

                }

                else

                {

                    result.Add(first[fIter]); fIter++; sIter++;

                }

            }

            return result;

        }

        private static List<int> ConstructPath(int[] parent, int dest)

        {

            List<int> path = new List<int>();

            int i = dest;

            while (i != -1)

            {

                path.Add(i); i = parent[i];

            }

            return path;

        }

        private static void CalculateFrequency()

        {

            int src; string actorName; do

            {

                Console.Write("Enter Actor Name: ");

                actorName = Console.ReadLine();

                if (index.ContainsKey(actorName))

                {

                    src = index[actorName]; break;

                }

                else

                {

                    Console.WriteLine("Invalid Actor/Acteress Name");

                }

            } while (true);

            MAX\_DEGREE\_FOUND = 0; GetKnown(src);

            Console.WriteLine("Deg.of Separ.{0,2}Frequency", "");

            for (int i = 0; i <= MAX\_DEGREE\_FOUND; i++)

            {

                Console.WriteLine("{0,12}{1,12}", i, dosFrequency[i]);

            }

        }

        private static void FindStrongestPath()

        {

            int src = -1, dest = -1;

            do

            {

                Console.Write("Enter Source Actor Name: ");

                string srcName = Console.ReadLine();

                if (index.ContainsKey(srcName))

                {

                    src = index[srcName];

                    Console.Write("Enter Destination Actor Name: ");

                    string destName = Console.ReadLine();

                    if (index.ContainsKey(destName))

                    {

                        dest = index[destName];

                    }

                    else

                    {

                        Console.WriteLine("Invalid Actor/Acteress Name");

                    }

                }

                else

                {

                    Console.WriteLine("Invalid Actor/Acteress Name");

                }

            } while (src == -1 || dest == -1);

            bool isReachable = false;

            isReachable = FindShortestPath(src, dest) != null;

            if (isReachable)

            {

                bool[] discovered = new bool[index.Count()];

                Stack<int> path = new Stack<int>();

                List<List<int>> paths = new List<List<int>>();

                path.Push(src);

                Visit(src, discovered, dest, path, paths);

                Console.WriteLine(GenerateAnswer(paths[max\_relation\_index]));

            }

        }

        static int max\_relation = 0;

        static int max\_relation\_index = 0;

        private static void Visit(int u, bool[] discovered, int dest, Stack<int> path, List<List<int>> paths, int w = 0)

        {

            discovered[u] = true;

            foreach (var adj in adjs[u])

            {

                if (adj == dest)

                {

                    path.Push(adj);

                    w += common[u][adj];

                    paths.Add(path.ToList());

                    if (w > max\_relation)

                    {

                        max\_relation = w;

                        max\_relation\_index = paths.Count() - 1;

                    }

                    path.Pop();

                    w -= common[u][adj];

                    continue;

                }

                if (!discovered[adj])

                {

                    path.Push(adj);

                    Visit(adj, discovered, dest, path, paths, w + common[u][adj]);

                    path.Pop();

                }

            }

            discovered[u] = false;

        }

    }

    class Normal

    {

        private static string moviesPath, queriesPath, solutionPath;

        private static List<KeyValuePair<string, string>> queries = new List<KeyValuePair<string, string>>();

        private static Queue<string> answers = new Queue<string>();

        private static Dictionary<string, HashSet<string>> actorsInMovie = new Dictionary<string, HashSet<string>>();

        private static Dictionary<string, HashSet<string>> moviesOfActor = new Dictionary<string, HashSet<string>>();

        static bool sample;

        public static void Run()

        {

            sample = false;

            SelectTestCase();

            Console.WriteLine("Parsing Test Case ...");

            ParseMovies();

            ParseQueries();

            ParseSolution();

            Console.WriteLine("Parsing Is Done");

            RunTestCase();

            ClearAll();

            Console.ReadLine();

        }

        private static void ClearAll()

        {

            actorsInMovie.Clear();

            moviesOfActor.Clear();

            queries.Clear();

            answers.Clear();

        }

        private static void ParseSolution()

        {

            StreamReader reader = new StreamReader(solutionPath);

            if (!sample)

            {

                List<string> answer = new List<string>();

                while (true)

                {

                    string line = reader.ReadLine();

                    if (line == null)

                        break;

                    if (line == "")

                    {

                        string ans;

                        ans = answer[0] + answer[1] + answer[2] + answer[3];

                        answers.Enqueue(ans);

                        answer.Clear();

                    }

                    else

                    {

                        answer.Add(line + "\n");

                    }

                }

            }

            else

            {

                while (true)

                {

                    string line = reader.ReadLine();

                    if (line == null)

                        break;

                    answers.Enqueue(line);

                }

            }

        }

        public static void ParseMovies()

        {

            StreamReader reader = new StreamReader(moviesPath);

            while (true)

            {

                string line = reader.ReadLine();

                if (line == null)

                    break;

                string movie\_name = line.Substring(0, line.IndexOf('/'));

                actorsInMovie[movie\_name] = new HashSet<string>();

                Movie movie = new Movie(movie\_name);

                while (true)

                {

                    line = line.Remove(0, (line.IndexOf('/') != -1) ? line.IndexOf('/') + 1 : line.Length);

                    if (line.Length == 0)

                        break;

                    string actor\_name = (line.IndexOf('/') != -1) ? line.Substring(0, line.IndexOf('/')) : line;

                    actorsInMovie[movie\_name].Add(actor\_name);

                    if (!moviesOfActor.ContainsKey(actor\_name))

                        moviesOfActor[actor\_name] = new HashSet<string>();

                    moviesOfActor[actor\_name].Add(movie\_name);

                    movie.actors.Add(actor\_name);

                }

            }

        }

        public static void ParseQueries()

        {

            StreamReader reader = new StreamReader(queriesPath);

            while (true)

            {

                string line = reader.ReadLine();

                if (line == null)

                    break;

                string key = line.Substring(0, line.IndexOf('/'));

                line = line.Remove(0, key.Length);

                string value = line.Substring(1);

                queries.Add(new KeyValuePair<string, string>(key, value));

            }

        }

        private static void SelectTestCase()

        {

            string path = @"..\..\..\Testcases\";

            string choice;

            do

            {

                Console.WriteLine("0.Sample");

                Console.WriteLine("-Complete");

                Console.WriteLine(" -small");

                Console.WriteLine("\t1-Case1");

                Console.WriteLine("\t2-Case2");

                Console.WriteLine(" -medium");

                Console.WriteLine("  -Case1");

                Console.WriteLine("\t3.queries85");

                Console.WriteLine("\t4.queries4000");

                Console.WriteLine("  -Case2");

                Console.WriteLine("\t5.queries110");

                Console.WriteLine("\t6.queries2000");

                Console.WriteLine(" -large");

                Console.WriteLine("\t7.queries26");

                Console.WriteLine("\t8.queries600");

                Console.WriteLine(" -extreme");

                Console.WriteLine("\t9.queries22");

                Console.WriteLine("\t10.queries200");

                Console.Write("Select Test Case: ");

                choice = Console.ReadLine();

                if (!(choice == "0" || choice == "1" || choice == "2" || choice == "3" || choice == "4" || choice == "5" || choice == "6" || choice == "7" || choice == "8" || choice == "9" || choice == "10"))

                {

                    Console.ForegroundColor = ConsoleColor.Red;

                    Console.WriteLine("Invalid Choice");

                    Console.ResetColor();

                }

                else

                    break;

            }

            while (true);

            switch (choice)

            {

                case "0":

                    path += @"Sample\";

                    moviesPath = path + "movies1.txt";

                    queriesPath = path + "queries1.txt";

                    solutionPath = path + "queries1 - Solution.txt";

                    sample = true;

                    break;

                case "1":

                    path += @"Complete\small\Case1\";

                    moviesPath = path + "Movies193.txt";

                    queriesPath = path + "queries110.txt";

                    solutionPath = path + @"\Solution\queries110 - Solution.txt";

                    break;

                case "2":

                    path += @"Complete\small\Case2\";

                    moviesPath = path + "Movies187.txt";

                    queriesPath = path + "queries50.txt";

                    solutionPath = path + @"\Solution\queries50 - Solution.txt";

                    break;

                case "3":

                    path += @"Complete\medium\Case1\";

                    moviesPath = path + "Movies967.txt";

                    queriesPath = path + "queries85.txt";

                    solutionPath = path + @"\Solutions\queries85 - Solution.txt";

                    break;

                case "4":

                    path += @"Complete\medium\Case1\";

                    moviesPath = path + "Movies967.txt";

                    queriesPath = path + "queries4000.txt";

                    solutionPath = path + @"\Solutions\queries4000 - Solution.txt";

                    break;

                case "5":

                    path += @"Complete\medium\Case2\";

                    moviesPath = path + "Movies4736.txt";

                    queriesPath = path + "queries110.txt";

                    solutionPath = path + @"\Solutions\queries110 - Solution.txt";

                    break;

                case "6":

                    path += @"Complete\medium\Case2\";

                    moviesPath = path + "Movies4736.txt";

                    queriesPath = path + "queries2000.txt";

                    solutionPath = path + @"\Solutions\queries2000 - Solution.txt";

                    break;

                case "7":

                    path += @"Complete\large\";

                    moviesPath = path + "Movies14129.txt";

                    queriesPath = path + "queries26.txt";

                    solutionPath = path + @"\Solutions\queries26 - Solution.txt";

                    break;

                case "8":

                    path += @"Complete\large\";

                    moviesPath = path + "Movies14129.txt";

                    queriesPath = path + "queries600.txt";

                    solutionPath = path + @"\Solutions\queries600 - Solution.txt";

                    break;

                case "9":

                    path += @"Complete\extreme\";

                    moviesPath = path + "Movies122806.txt";

                    queriesPath = path + "queries22.txt";

                    solutionPath = path + @"\Solutions\queries22 - Solution.txt";

                    break;

                case "10":

                    path += @"Complete\extreme\";

                    moviesPath = path + "Movies122806.txt";

                    queriesPath = path + "queries200.txt";

                    solutionPath = path + @"\Solutions\queries200 - Solution.txt";

                    break;

            }

        }

        public static void RunTestCase()

        {

            int current = 1;

            int passed = 0;

            if (sample)

                Console.WriteLine(answers.Dequeue());

            Stopwatch stopwatch = new Stopwatch();

            stopwatch.Start();

            foreach (KeyValuePair<string, string> kvp in queries)

            {

                string answer = Solve(kvp.Key, kvp.Value);

                Console.ForegroundColor = ConsoleColor.DarkCyan;

                Console.Write("Query {0}/{1} ", current, queries.Count);

                Console.ResetColor();

                string actual = answers.Dequeue();

                if (answer != actual)

                {

                    Console.WriteLine("+=======================+");

                    Console.WriteLine("Query {0} failed: {1}/{2}", current, kvp.Key, kvp.Value);

                    Console.WriteLine("Expected:\n {0}", actual);

                    Console.WriteLine("Actual:\n {0}", answer);

                    Console.ForegroundColor = ConsoleColor.Red;

                    Console.Write($" Failed\n");

                    Console.ResetColor();

                }

                else

                {

                    passed++;

                    Console.ForegroundColor = ConsoleColor.Green;

                    Console.Write($" Passed\n");

                    Console.ResetColor();

                    Console.WriteLine(answer);

                }

                current++;

            }

            stopwatch.Stop();

            Console.ForegroundColor = ConsoleColor.DarkYellow;

            Console.Write($"Time taken:  {stopwatch.ElapsedMilliseconds / 1000.0} Seconds | {stopwatch.ElapsedMilliseconds} Milliseconds");

            Console.ResetColor();

        }

        public static string Solve(string who, string whom)

        {

            int degree = 1, answerDegree = int.MaxValue, maxRelation = 0;

            List<Answer> answers = new List<Answer>();

            Dictionary<string, Actor> dictActors = new Dictionary<string, Actor>();

            Queue<Actor> actorsToAsk = new Queue<Actor>();

            HashSet<string> lvlActors = new HashSet<string>();

            HashSet<string> nextLvlActors = new HashSet<string>();

            HashSet<string> lvlAskedActors = new HashSet<string>();

            dictActors[who] = new Actor(who, null);

            actorsToAsk.Enqueue(dictActors[who]);

            lvlActors.Add(who);

            while (actorsToAsk.Count() != 0)

            {

                Actor actor = actorsToAsk.Dequeue();

                List<string> movies = moviesOfActor[actor.name].ToList();

                foreach (string movie in movies)

                {

                    if (actor.askedMovies.ContainsKey(movie) && actor.askedMovies[movie] == true)

                        continue;

                    else

                        actor.askedMovies[movie] = false;

                    List<string> actors = actorsInMovie[movie].ToList();

                    foreach (string actorInMovie in actors)

                    {

                        Actor movieActor = new Actor(actorInMovie, actor);

                        if (dictActors.ContainsKey(actorInMovie))

                        {

                            movieActor = dictActors[actorInMovie];

                        }

                        else

                        {

                            if (actorInMovie != actor.name)

                            {

                                List<string> commonMovies = moviesOfActor[actorInMovie].Intersect(moviesOfActor[actor.name]).ToList();

                                movieActor.commonMovies = commonMovies;

                                movieActor.weight = movieActor.parent.weight + movieActor.commonMovies.Count();

                            }

                            dictActors[actorInMovie] = movieActor;

                        }

                        if (!movieActor.asked && actorInMovie != actor.name && actorInMovie != whom)

                        {

                            if (!movieActor.toSkip)

                            {

                                actorsToAsk.Enqueue(movieActor);

                                movieActor.toSkip = true;

                            }

                            if (lvlActors.Contains(actor.name) && !lvlActors.Contains(actorInMovie))

                            {

                                List<string> commonMovies = moviesOfActor[actorInMovie].Intersect(moviesOfActor[actor.name]).ToList();

                                nextLvlActors.Add(actorInMovie);

                                if (actor.weight + commonMovies.Count() > movieActor.parent.weight + movieActor.commonMovies.Count())

                                {

                                    movieActor.parent = actor;

                                    movieActor.commonMovies = commonMovies;

                                    movieActor.weight = movieActor.parent.weight + movieActor.commonMovies.Count();

                                }

                            }

                        }

                        if (actorInMovie == whom)

                        {

                            actor.askedMovies[movie] = true;

                            Answer answer = new Answer();

                            Actor whomActor = new Actor(whom, actor);

                            Stack<string> moviesList = new Stack<string>();

                            Stack<string> actorList = new Stack<string>();

                            Actor curr = new Actor(whom, actor);

                            int relation = 0;

                            do

                            {

                                if (curr == null || curr.parent == null)

                                    break;

                                List<string> commonMoviesList = moviesOfActor[curr.name].Intersect(moviesOfActor[curr.parent.name]).ToList();

                                string commonMovies = "";

                                if (!sample)

                                    commonMovies = commonMoviesList[0];

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

                                {

                                    relation++;

                                    if (sample)

                                    {

                                        if (commonMovies == "")

                                            commonMovies += commonMoviesList[i];

                                        else

                                            commonMovies += commonMoviesList[i].Substring(commonMoviesList[i].Count() - 1);

                                        if (i != commonMoviesList.Count() - 1)

                                            commonMovies += " or ";

                                    }

                                    string elm = commonMoviesList[i];

                                    actor.askedMovies[elm] = true;

                                }

                                moviesList.Push(commonMovies);

                                actorList.Push(curr.name);

                                curr = curr.parent;

                            } while (true);

                            maxRelation = (maxRelation < relation) ? relation : maxRelation;

                            answer.moviesList = moviesList;

                            answer.actorList = actorList;

                            answer.degree = degree;

                            answer.relation = relation;

                            answerDegree = degree;

                            answers.Add(answer);

                        }

                    }

                }

                actor.asked = true;

                lvlAskedActors.Add(actor.name);

                if (lvlActors.Count() == lvlAskedActors.Count())

                {

                    degree++;

                    lvlActors.Clear();

                    lvlAskedActors.Clear();

                    lvlActors = nextLvlActors;

                    nextLvlActors = new HashSet<string>();

                }

                if (answerDegree < degree)

                    break;

            }

            foreach (Answer answer in answers)

            {

                if (answer.relation == maxRelation)

                {

                    return answer.print(who, whom);

                }

            }

            return "";

        }

        public class Actor

        {

            public int weight = 0;

            public string name;

            public Actor parent;

            public bool asked = false;

            public bool toSkip = false;

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

            public Dictionary<string, bool> askedMovies = new Dictionary<string, bool>();

            public Actor(string name, Actor parent)

            {

                this.name = name;

                this.parent = parent;

            }

            public Actor(int weight, string name, Actor parent, bool asked, bool toSkip, List<string> commonMovies, Dictionary<string, bool> askedMovies)

            {

                this.weight = weight;

                this.name = name;

                this.parent = parent;

                this.asked = asked;

                this.toSkip = toSkip;

                this.commonMovies = commonMovies;

                this.askedMovies = askedMovies;

            }

            public Actor copy()

            {

                return new Actor(weight, name, parent, asked, toSkip, commonMovies, askedMovies);

            }

            public void setWeight()

            {

                Actor dummy = copy();

                weight = 0;

                while (dummy != null)

                {

                    weight += dummy.commonMovies.Count();

                    dummy = dummy.parent;

                }

            }

        }

        public class Movie

        {

            public string name;

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

            public Movie(string name)

            {

                this.name = name;

            }

        }

        public class Answer

        {

            public int degree;

            public int relation;

            public Stack<string> moviesList = new Stack<string>();

            public Stack<string> actorList = new Stack<string>();

            public string print(string who, string whom)

            {

                string answer = "";

                if (!sample)

                {

                    answer += String.Format("{0}/{1}\n", who, whom);

                    answer += String.Format("DoS = {0}, RS = {1}\n", degree, relation);

                    answer += String.Format("CHAIN OF ACTORS: {0} -> ", who);

                    while (actorList.Count() != 0)

                    {

                        answer += actorList.Pop();

                        if (actorList.Count() != 0)

                            answer += " -> ";

                    }

                    answer += "\n";

                    answer += "CHAIN OF MOVIES:  =>";

                    while (moviesList.Count() != 0)

                    {

                        answer += " " + moviesList.Pop();

                        answer += " =>";

                    }

                    answer += "\n";

                }

                else

                {

                    answer += $"{who}/{whom}\t\t{degree}\t\t{relation}\t\t";

                    while (moviesList.Count() != 0)

                    {

                        answer += moviesList.Pop();

                        if (moviesList.Count() != 0)

                            answer += " => ";

                    }

                }

                return answer;

            }

        }

    }

}

## Analysis

// O(V + E)

        private static string Solve(string src, string dest)

        {

            int iFirst = index[src]; // O(1)

            int iSecond = index[dest]; // O(1)

            List<int> shortestPath = FindShortestPath(iFirst, iSecond); // O(V + E)

            if (shortestPath != null) //O(1)

                return GenerateAnswer(shortestPath); // O (V + E)

            else

                return ""; // O(1)

        }

        // O (V + E)

        private static string GenerateAnswer(List<int> path)

        {

            int degree = 0, relation = 0; // O(1)

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

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

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

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

            if (!sample) // O(1)

                answer += $"{actorsName[path.Last()]}/{actorsName[path.First()]}\n"; // O(1)

            if (path.Count() != 1) // O(1)

            {

                for (int i = 0; i < path.Count() - 1; i++) // O (V + E)

                {

                    List<int> movies = GetIntersection(actorsMovies[path[i]], (actorsMovies[path[i + 1]])); // O(N + M)

                    chainOfActors.Push(actorsName[path[i]]); // O(1)

                    if (!sample) // O(1)

                        chainOfMovies.Push(moviesNames[movies[0]]); // O(1)

                    else

                    {

                        commonMovies = ""; // O(1)

                        foreach (var m in movies) // O(N)

                        {

                            if (commonMovies == "") // O(1)

                                commonMovies += moviesNames[m]; // O(1)

                            else

                                commonMovies += moviesNames[m].Substring(moviesNames[m].Count() - 1); // O(1)

                            if (m != movies.Last()) // O(1)

                                commonMovies += " or "; // O(1)

                        }

                        chainOfMovies.Push(commonMovies); // O(1)

                    }

                    relation += movies.Count; // O(1)

                    degree++;// O(1)

                }

            }

            else

            {

                relation += actorsMovies[path[0]].Count(); // O(1)

            }

            if (!sample) // O(1)

            {

                answer += $"DoS = {degree}, RS = {relation}\n"; // O(1)

                answer += $"CHAIN OF ACTORS: {actorsName[path.Last()]} -> "; // O(1)

                while (chainOfActors.Count() != 0) // O(N)

                {

                    answer += chainOfActors.Pop(); // O(1)

                    if (chainOfActors.Count() != 0) // O(1)

                        answer += " -> "; // O(1)

                }

                answer += "\n"; // O(1)

                answer += "CHAIN OF MOVIES:  =>"; // O(1)

                while (chainOfMovies.Count() != 0) // O(N)

                {

                    answer += " " + chainOfMovies.Pop(); // O(1)

                    answer += " =>"; // O(1)

                }

                answer += "\n"; // O(1)

            }

            else

            {

                //"A/B\t\t1\t\t3\t\tMovie 1 or 2 or 7"

                answer += $"{actorsName[path.Last()]}/{actorsName[path.First()]}\t\t{degree}\t\t{relation}\t\t"; // O(1)

                while (chainOfMovies.Count() != 0) // O(N)

                {

                    answer += chainOfMovies.Pop();// O(1)

                    if (chainOfMovies.Count() != 0) // O(1)

                        answer += " => "; // O(1)

                }

            }

            return answer; // O(1)

        }

        // O(V + E)

        private static List<int> FindShortestPath(int start, int end)

        {

            int[] parent = new int[index.Count()]; // O(1)

            int[] weight = new int[index.Count()];// O(1)

            bool[] visited = new bool[index.Count()];// O(1)

            int maxDist = int.MaxValue; // O(1)

            int[] dist = new int[index.Count()]; // O(1)

            bool found = false; // O(1)

            if (start == end) // O(1)

            {

                parent[end] = -1; // O(1)

                return ConstructPath(parent, end); // O(V)

            }

            else

            {

                Queue<int> q = new Queue<int>(); // O(1)

                q.Enqueue(start); // O(1)

                parent[start] = -1; // O(1)

                weight[start] = 0; // O(1)

                dist[start] = 0; // O(1)

                while (q.Count != 0) // O(V)

                {

                    int u = q.Dequeue(); // O(1)

                    if (visited[u]) // O(1)

                        continue; // O(1)

                    if (dist[u] + 1 > maxDist) // O(1)

                        break; // O(1)

                    foreach (int v in adjs[u]) // O(E)

                    {

                        if (v == start) // O(1)

                            continue; // O(1)

                        if (visited[v]) // O(1)

                            continue; // O(1)

                        if (dist[v] == 0 || dist[v] > dist[u] + 1) // O(1)

                        {

                            int commonWithU = common[v][u] + weight[u]; // O(1)

                            dist[v] = dist[u] + 1; // O(1)

                            q.Enqueue(v); // O(1)

                            parent[v] = u; // O(1)

                            weight[v] = commonWithU; // O(1)

                        }

                        else if (dist[v] == dist[u] + 1) // O(1)

                        {

                            int commonWithU = common[v][u] + weight[u]; // O(1)

                            int commonWithParent = common[v][parent[v]] + weight[parent[v]]; // O(1)

                            if (commonWithU > commonWithParent) // O(1)

                            {

                                parent[v] = u; // O(1)

                                weight[v] = commonWithU; // O(1)

                            }

                        }

                        if (v == end) // O(1)

                        {

                            found = true; // O(1)

                            maxDist = dist[v]; // O(1)

                        }

                    }

                    visited[u] = true; // O(1)

                }

                if (found == false) // O(1)

                    return null; // O(1)

                return ConstructPath(parent, end); // O(V)

            }

        }

        // O(V + E)

        private static void GetKnown(int src)

        {

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

            int[] parent = new int[index.Count()]; // O(1)

            int[] weight = new int[index.Count()]; // O(1)

            bool[] visited = new bool[index.Count()]; // O(1)

            int maxDist = int.MaxValue; // O(1)

            int[] dist = new int[index.Count()]; //O(1)

            dist = Enumerable.Repeat(int.MaxValue, index.Count()).ToArray(); //O(V)

            Queue<int> q = new Queue<int>(); // O(1)

            q.Enqueue(src); // O(1)

            parent[src] = -1; // O(1)

            weight[src] = 0; // O(1)

            dist[src] = 0; // O(1)

            while (q.Count != 0) // O(V)

            {

                int u = q.Dequeue(); // O(1)

                if (visited[u]) // O(1)

                    continue; // O(1)

                if (dist[u] + 1 > maxDist) // O(1)

                    break; // O(1)

                foreach (int v in adjs[u]) // O(E)

                {

                    if (dist[v] > dist[u] + 1) // O(1)

                    {

                        int commonWithU = common[v][u] + weight[u]; // O(1)

                        if (dist[v] != int.MaxValue && dosFrequency[dist[v]] != 0) // O(1)

                            dosFrequency[dist[v]]--; // O(1)

                        dist[v] = dist[u] + 1; // O(1)

                        dosFrequency[dist[v]]++; // O(1)

                        if (MAX\_DEGREE\_FOUND < dist[v]) // O(1)

                            MAX\_DEGREE\_FOUND = dist[v]; // O(1)

                        q.Enqueue(v); // O(1)

                        weight[v] = 0; // O(1)

                        parent[v] = u; // O(1)

                        weight[v] = commonWithU; // O(1)

                    }

                    else if (dist[v] == dist[u] + 1) // O(1)

                    {

                        int commonWithU = common[v][u] + weight[u]; // O(1)

                        int commonWithParent = common[v][parent[v]] + weight[parent[v]]; // O(1)

                        if (commonWithU > commonWithParent) // O(1)

                        {

                            parent[v] = u; // O(1)

                            weight[v] = commonWithU; // O(1)

                        }

                    }

                }

                visited[u] = true; // O(1)

            }

        }

        // O(N + M)

        private static List<int> GetIntersection(List<int> first, List<int> second)

        {

            int fIter = 0; // O(1)

            int sIter = 0; // O(1)

            var result = new List<int>(); // O(1)

            while (fIter < first.Count && sIter < second.Count) // O(N + M)

            {

                if (first[fIter] < second[sIter]) // O(1)

                {

                    fIter++; // O(1)

                }

                else if (first[fIter] > second[sIter]) // O(1)

                {

                    sIter++; // O(1)

                }

                else

                {

                    result.Add(first[fIter]); // O(1)

                    fIter++; // O(1)

                    sIter++; // O(1)

                }

            }

            return result; // O(1)

        }

        // O(V)

        private static List<int> ConstructPath(int[] parent, int dest)

        {

            List<int> path = new List<int>(); // O(1)

            int i = dest; // O(1)

            while (i != -1) // O(V)

            {

                path.Add(i); // O(1)

                i = parent[i]; // O(1)

            }

            return path; // O(1)

        }

        // O(V + E)

        private static void CalculateFrequency()

        {

            int src; // O(1)

            string actorName; // O(1)

            do

            {

                Console.Write("Enter Actor Name: "); // O(1)

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

                if (index.ContainsKey(actorName)) //O(1)

                {

                    src = index[actorName]; //O(1)

                    break; //O(1)

                }

                else

                {

                    Console.WriteLine("Invalid Actor/Acteress Name"); //O(1)

                }

            } while (true);

            MAX\_DEGREE\_FOUND = 0; // O(1)

            GetKnown(src); // O(V + E)

            Console.WriteLine("Deg.of Separ.{0,2}Frequency", ""); //O(1)

            for (int i = 0; i <= MAX\_DEGREE\_FOUND; i++) // O(N)

            {

                Console.WriteLine("{0,12}{1,12}", i, dosFrequency[i]); // O(1)

            }

        }

        // O(V + E)

        private static void FindStrongestPath(int src, int dest)

        {

            bool isReachable = false; //O(1)

            isReachable = FindShortestPath(src, dest) != null; // O(V + E)

            if (isReachable) //O(1)

            {

                bool[] discovered = new bool[index.Count()]; // O(1)

                Stack<int> path = new Stack<int>(); // O(1)

                List<List<int>> paths = new List<List<int>>(); // O(1)

                path.Push(src); // O(1)

                Visit(src, discovered, dest, path, paths); // O(V)

                Console.WriteLine(GenerateAnswer(paths[max\_relation\_index])); // O (V + E)

            }

        }

        static int max\_relation = 0;

        static int max\_relation\_index = 0;

        // O(V)

        private static void Visit(int u, bool[] discovered, int dest, Stack<int> path, List<List<int>> paths, int w = 0)

        {

            discovered[u] = true;

            foreach (var adj in adjs[u])

            {

                if (adj == dest)

                {

                    path.Push(adj);

                    w += common[u][adj];

                    paths.Add(path.ToList());

                    if (w > max\_relation)

                    {

                        max\_relation = w;

                        max\_relation\_index = paths.Count() - 1;

                    }

                    path.Pop();

                    w -= common[u][adj];

                    continue;

                }

                if (!discovered[adj])

                {

                    path.Push(adj);

                    Visit(adj, discovered, dest, path, paths, w + common[u][adj]);

                    path.Pop();

                }

            }

            discovered[u] = false;

        }

## Execution Time

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Type** | **Test case** | **Movies** | **Actors** | **Queries** | **Normal (s)** | **Optimization (s)** | **Diff (s)** | **Improvement** |
| Small | 1 | 193 | 2513 | 110 | 1.482 | 0.3 | 1.182 | 494% |
| Small | 2 | 187 | 8264 | 50 | 5.729 | 0.6 | 5.129 | 955% |
| Medium | 3 | 967 | 13848 | 85 | 16.241 | 1.5 | 14.741 | 1083% |
| Medium | 4 | 967 | 13848 | 4000 | 262.766 | 26 | 236.766 | 1011% |
| Medium | 5 | 4735 | 43923 | 110 | 46.423 | 5 | 41.423 | 928% |
| Medium | 6 | 4735 | 43923 | 2000 | 175.366 | 22 | 153.366 | 797% |
| Large | 7 | 14129 | 170518 | 26 | 231.683 | 26 | 205.683 | 891% |
| Large | 8 | 14129 | 170518 | 600 | 24.732 | 4 | 20.732 | 618% |
| Extreme | 9 | 122806 | 418451 | 22 | 1651.905 | 85 | 1566.905 | 1943% |
| Extreme | 10 | 122806 | 418451 | 200 | 101.333 | 8 | 93.333 | 1267% |