## 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% |