Plagiarism Analysis

A checking tool is used to detect code plagiarism of certain assessment. It provides a list of matching pairs together with the corresponding similarity score (%) of each pair. However, unfortunately, the same solution can be found in different submissions.

Given a list of submission IDs and a list of matching pairs with their similarity scores (%), it's required to calculate the average similarity percentage of the component containing the given submission ID (i.e. startVertex)

Input:

- |V| = from 4000 to 8000
- |E| = sparse or dense
- # communities = from 1 to 100

Function to Implement

```
float AnalyzeMatchingScore(string[] vertices, Tuple<string, string, float>[] 
                                edges, string startVertex)
```

`PlagiarismAnalysis.cs includes this method.`

- "vertices": array of submission IDs
- "edges": array of matching pairs and their similarity score (where **`Item1: ID1, Item2: ID2, Item3: similarity score (%)`**)
- "startVertex": start vertex to analyze its component

<returns> average similarity score (%) of each component in the Graph

Example

```
vertices1 = { "19T021", "19T024", "19T025"};
edges1[0] = new Tuple<string, string, float>("19T021", "19T024", 10);
edges1[1] = new Tuple<string, string, float>("19T024", "19T025", 15);
startVertex ="19T024"
expected1 = 12.5;
```

```
vertices3 = { "A1", "A2", "A3", "A4", "A5", "A6" };
edges3[0] = new Tuple<string, string, float>("A1", "A2", 1);
edges3[1] = new Tuple<string, string, float>("A2", "A3",2);
edges3[2] = new Tuple<string, string, float>("A5", "A4",3);
edges3[3] = new Tuple<string, string, float>("A5", "A6",4);
edges3[4] = new Tuple<string, string, float>("A3", "A5",5);
edges3[5] = new Tuple<string, string, float>("A4", "A2",6);
startVertex = "A6"
expected3 = 3.5;
```

C# Help

Queues

Creation

To create a queue of a certain type (e.g. string)

```
Queue<string> myQ = new Queue<string>() //default initial size

Queue<string> myQ = new Queue<string>(initSize) //given initial size
```

Manipulation

1. `myQ.Count` ➔ get actual number of items in the queue
2. `myQ.Enqueue(“myString1”)` ➔ Add new element to the queue
3. `myQ.Dequeue()` ➔ return the top element of the queue (FIFO)

Lists

Creation

To create a list of a certain type (e.g. string)

```
List<string> myList1 = new List<string>() //default initial size

List<string> myList2 = new List<string>(initSize) //given initial size
```

Manipulation

4. `myList1.Count` ➔ get actual number of items in the list
5. `myList1.Sort()` ➔ Sort the elements in the list (ascending)
6. `myList1[index]` ➔ Get/Set the elements at the specified index
7. `myList1.Add(“myString1”)` ➔ Add new element to the list
8. `myList1.Remove(“myStr1”)` ➔ Remove the 1st occurrence of this element from list
9. `myList1.RemoveAt(index)` ➔ Remove the element at the given index from the list
10. `myList1.Contains(“myStr1”)` ➔ Check if the element exists in the list

Dictionary (Hash)

Creation

To create a dictionary of a certain key (e.g. string) and value (e.g. array of strings)

```
//default initial size
Dictionary<string, string[]> myDict1 = new  Dictionary<string, string[]>();
```

```
//given initial size
Dictionary<string, string[]> myDict2 = new  Dictionary<string, string[]>(size);
```

Manipulation

1. `myDict1.Count` → Get actual number of items in the dictionary
2. `myDict1[key]` → Get/Set the value associated with the given key in the dictionary
3. `myDict1.Add(key, value)` → Add the specified key and value to the dictionary
4. `myDict1.Remove(key)` → Remove the value with the specified key from the dictionary
5. `myDict1.ContainsKey(key)` → Check if the specified key exists in the dictionary

Creating 1D array

```
int [] array = new int [size]
```

Creating 2D array

```
int [,] array = new int [size1, size2]
```

Length of 1D array

```
int arrayLength = my1DArray.Length
```

Length of 2D array

```
int array1stDim = my2DArray.GetLength(0)

int array2ndDim = my2DArray.GetLength(1)
```

Sorting single array

Sort the given array in ascending order

```
Array.Sort(items);
```

Sorting parallel arrays

Sort the first array "master" and re-order the 2nd array "slave" according to this sorting

```
Array.Sort(master, slave);
```