Plagiarism Checking

A checking tool is used to detect code plagiarism of certain assessment. It provides a list of matching pairs. However, two submissions may not be directly detected in a separate pair. Instead, they can be related via a chain of matching pairs.

Given a list of matching pairs and a query pair, find the min number of connections between the nodes of the given pair (if any)?

# Function to Implement

int CheckPlagiarism(Tuple<string, string>[] matches, Tuple<string, string> query)

PlagiarismChecking.cs includes this method.

* " matches ": array of matching pairs (where **Item1: ID1, Item2: ID2**)
* "query": a query pair to be checked

<returns> min number of connections between the nodes of the query pair (if any)

# Example

matches[0] = new Tuple<string, string>("1", "4");

matches[1] = new Tuple<string, string>("4", "5");

matches[2] = new Tuple<string, string>("2", "3");

query11 = new Tuple<string, string>("1", "3");

int expected11 = 0;

query12 = new Tuple<string, string>("5", "4");

expected12 = 1;

matches [0] = new Tuple<string, string>("1", "2");

matches [1] = new Tuple<string, string>("2", "3");

matches [2] = new Tuple<string, string>("5", "4");

matches [3] = new Tuple<string, string>("5", "6");

matches [4] = new Tuple<string, string>("3", "5");

matches [5] = new Tuple<string, string>("4", "2");

query21 = new Tuple<string, string>("1", "5");

int expected21 = 3;

# 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

1. myList1.Count 🡺 get actual number of items in the list
2. myList1.Sort()🡺 Sort the elements in the list (ascending)
3. myList1[index]🡺 Get/Set the elements at the specified index
4. myList1.Add(“myString1”)🡺 Add new element to the list
5. myList1.Remove(“myStr1”)🡺 Remove the 1st occurrence of this element from list
6. myList1.RemoveAt(index)🡺 Remove the element at the given index from the list
7. 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);