UWCS Mini-Challenge

A leaderboard-based programming contest

Introduction

Welcome to the Term 3 programming mini-challenge! This is an online, leaderboard-based event that will run from Thursday 28th April - Tuesday 3rd May.

We'll start off with an introduction to the submission tools, then build up to complete an introductory problem (which you can check your solution for). This will be followed by the main problem, which you'll be trying to get the highest score on!

The winner will recieve a **£20 Amazon Voucher**! This is for current students only.

For those of you that are doing the personal development plan (PDP), we'd be happy to sign off on this event - as long as you submit a solution achieving a reasonable score.

Setup

To take part, you'll need to download a program we made to interact with your script. It will send input data, recieve the output data, and give back a mark or score.

Donwload the ProgcompCLI here: `https://github.com/Jlobblet/ProgcompMarker/releases`

Select the download that's appropriate to your operating system and architecture. On a DCS machine, this would be `linux-x64`. On most Windows machines, this is likely `win-64`. You can choose whether to download it as a zip file (or tarball), which you'll need to extract.

In the extracted folder, you'll find an exectuable called `ProgcompCli`. This will have the `.exe` extension if you're on windows. Run it from the command line with no arguments.

You should now be prompted for first-time setup. Enter `https://progcomp.uwcs.co.uk` for the endpoint, and pick a username which you think will identify you.

Using the ProgcompCLI

To submit a solution, you'll need to provide at least two arguments: the problem number in this document such as (0), and the path to the executable script you made to solve it.

By default, data will be sent from the server via standard input (`stdin`), and your standard output (`stdout`) will be redirected to the server to be marked. However, this can be changed! Check out the command options for more details.

If your language can directly output an executable, then you should be able to pass it as the argument. However, most languages do not do this by default.

```
./ProgcompCli 0 Solution.exe
```

If you're using an interpreted language and a UNIX-based system (Mac, Linux), you might want to add a *shebang* as the first line of your script, allowing it to be executed without any arguments. For Python, this is likely to be `#!/usr/bin/env python`. Otherwise, use the `executable-args` flag to pass in multiple arguments. Here is a Windows Java example:

```
.\ProgcompCli.exe 0 java --executable-args Solution
```

⓪ Example Problem - Anagram

A Rag Man.

This problem is intended to get you familiar with how submission works.

A pair of strings are called anagrams of each other if they share the same letters, but in a different order. Your task: determine whether given pairs of strings are anagrams.

Input

The first line of the input contains an integer t, denoting the number of test cases.

The next t lines of the input consist of two space-separated strings w_1 and w_2.

Constraints

- $1 \leq t \leq 10$
- $1 \leq |w_1| = |w_2| \leq 13$
- w_1 and w_2 will be uppercase.

Output

t lines each consisting of either "Yes" if the strings are anagrams, or "No" if they're not.

Example

Input	Output
3 BABA ABBA ORANGE CHERRY MOUNTAINEERS ENUMERATIONS	 Yes No Yes

Note that we have one output line for each of the test cases.

Feel free to give the problem a try yourself! However, the main goal of this is for you to understand the submission system, so I've given some example solutions on the next page, and ways to submit them. These are given for Python and Java, to show the differences.

Once you feel comfortable, let's move onto the first real problem!

Solutions and Submission

```python
#!/usr/bin/env python

t = int(input()) # Get number of test cases

for _ in range(t):
   # Get the two words
   w1, w2 = input().split()

   # Convert to sorted character arrays
   s1 = sorted(w1)
   s2 = sorted(w2)

   # Compare the arrays
   if s1 == s2:
      print("Yes")
   else:
      print("No")
```

`./ProgcompCli 0 sol.py` on Mac or Linux, with a shebang.
`.\ProgcompCli.exe 0 python.exe --executable-args sol.py` on Windows (remove `.exe` otherwise).

```java
import java.io.*;
import java.util.*;

public class Solution {
   public static void main(String[] args) {
      // Allows us to read from stdin
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

      // Get number of test cases
      int t = 0;
      try {
         t = Integer.parseInt(br.readLine());
      } catch (IOException ex) {}

      String word1, word2;
      for (int i = 0; i < t; i++) { // Repeat for number of test cases
         // Get words
         try {
            String line = br.readLine();
            String[] split = line.split(" ");
            word1 = split[0];
            word2 = split[1];
         } catch (IOException ex) { continue; }

         // Sort arrays
         char[] arr1 = word1.toCharArray();
         Arrays.sort(arr1);
         char[] arr2 = word2.toCharArray();
         Arrays.sort(arr2);

         // Compare arrays
         if (Arrays.equals(arr1, arr2)) {
            System.out.println("Yes");
         } else {
            System.out.println("No");
         }
      }
   }
}
```

`.\ProgcompCli.exe 0 java --executable-args Solution` on Windows (remove `.exe` otherwise).
Make sure to compile it first with `javac`!

① Introductory Problem - Shift

I'm sure they'll figure out a use for this

For computers to automatically check spelling, they need to guess which word we were attempting to write. One method is to find a notion of 'distance' between what was typed against a word dictionary, and suggest the lowest-scoring options. These notions of distance are known as metrics, with the most popular being the Levenshtein word metric.

Let's make our own word metric! It's called `SHIFT`[1], and consists of two operations:

1. Shift a letter a position up or down the alphabet (with wraparound)
2. Shift a letter on either edge of a word to the other side.

The `SHIFT` distance is defined as the minimum number of operations we need to make to get from one word to another.

Given two words of the same length, what is the `SHIFT` distance between them?

Input

The first line of the input contains an integer t, denoting the number of test cases.

The next t lines of the input consist of two space-separated strings w_1 and w_2.

Constraints

- $1 \leq t \leq 10000$
- $1 \leq |w_1| = |w_2| \leq 13$
- w_1 and w_2 will be uppercase.

Output

t lines each consisting of a single integer, the `SHIFT` distance between w_1 and w_2.

Example

Input	Output
5 READ REED ZERO HERO PAID FEED WORDS SWORD REDDIT CRINGE	 4 8 18 1 31

Let's walk through these 5 test cases:

1. We shift the A in READ down by 4 to get to REED.
2. We shift the Z in ZERO down by 8 to get to HERO.
3. We shift P up by 10, A down by 4, and I up by 4 to get to FEED.
4. We shift WORDS to the right by 1, and it immediately becomes SWORD.
5. We shift to the left by 3 to get DITRED, then perform 28 further up/down shifts.

[1] in conversation, note that the 'F' is pronounced silently

② Main Problem - Shifty

What do you mean, "it's not a real metric"?

Okay so, remember that word metric from the introductory problem? As it turns out, it wasn't very useful - most pairs of words tend not to have the same length.

That's why I've come up with the new and improved `SHIFT` Two (`SHIFT-T` for short)! It adds two more operations to the mix:

3. Duplicate the left-most letter
4. Delete the left-most letter

These new rules can be applied with the others in any order. See? It's useful now! Now we just need someone to calculate the `SHIFT-T` distance between two words...

Input

The first line of the input contains an integer t, denoting the number of test cases.
The next t lines of the input consist of two space-separated strings w_1 and w_2.

Constraints

- $1 \leq t \leq 1000$
- $1 \leq |w_1|, |w_2| \leq 30$
- w_1 and w_2 will be uppercase.

Output

t lines, each consisting of a sequence of s space separated strings.

This sequence must start with w_1, end with w_2, and describe all the intermediate strings resulting from the application of valid `SHIFT-T` operations.

Although there may be multiple valid sequences of words, the length of your sequence should not need to exceed $m = 13 \cdot \max(|w_1|, |w_2|)$.

Scoring

- -100 Points for an invalid step sequence.
- $\max(0, m - s + 1)$ Points for a valid step sequence.

Example

Input	Output
5 READ REED MADE MAD MAD MADE A BABY PAID FEED	 READ REBD RECD REDD REED MADE EMAD MAD MAD DMA EMA EEMA DEMA ADEM MADE A AA AAA AAAA BAAA BABA BABZ BABY PAID AID ID D DD ED EED EEED FEED

This would return a score of 233. Can you do better?

To verify your submission for the contest, or serve as proof you've taken part, DM your username, score, and the script you used to our shared discord account - **UWCS#3568**.
`Good luck, and have fun!`