2.3 Problem 2

Step 1

The Secondary Structure of RNA

In this problem you will need to submit your solution in any of the supported programming languages. The code will be tested on all the prepared tests.

This problem has two versions with different constraints. If your solution passes all the tests of the easy version you will get 350 points. If your solution passes all the tests of the hard version you will get 650 points. In total you can get maximum 1000 points. For this you need to submit solutions to **both** easy and hard versions.

The description of the problem follows in the next step. Constraints for each version are also written in corresponding steps.

Step 2

Note: The constraints written in this problem previously were incorrect. The length does not exceed 151 not 501.

Clarification: a bond between to consecutive complementary nucleotides is possible.

The Secondary Structure of RNA

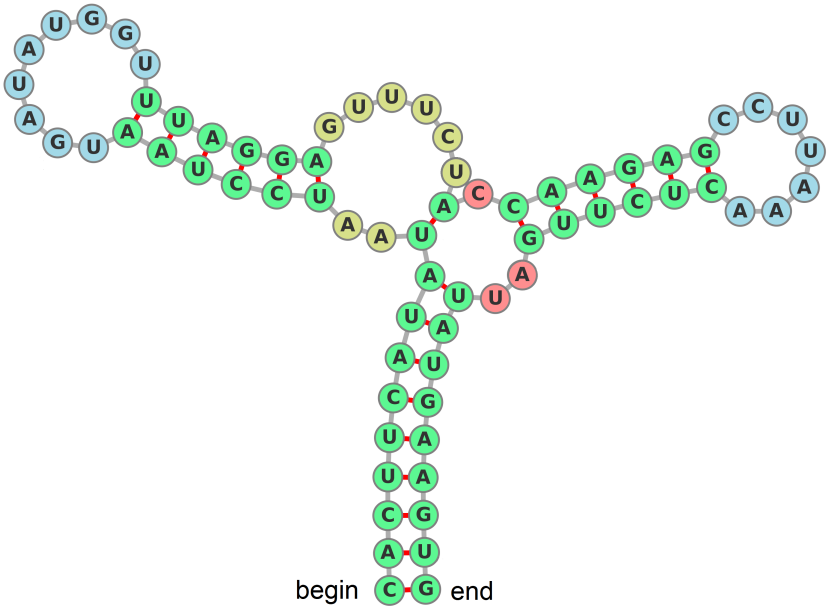
RNA consists of ribonucleotides of 4 types: adenine (A), cytosine (C), guanine (G), uracil (U). Therefore RNA can be represented as a string that consists of the characters A, C, G, U. It is known that the ribonucleotides A-U and C-G are complementary to each other.

RNA consists of one strand (chain of ribonucleotides) most of the time, and sometimes it can fold into the secondary structure. In this problem we consider a simpler secondary form than in real life. In this form, some pairs of complementary ribonucleotides build hydrogen bonds between them and each ribonucleotide can build at most one bond. Each bond can be represented as a pair of indices (i, j) $(i < j)$ of the string S that represents RNA. These indices indicate the positions of the symbols in S corresponding to these ribonucleotides.

Also, in the secondary structure one additional condition holds for each pair of hydrogen bonds (i_1, j_1), (i_2, j_2), such that $i_1 < i_2$:

- they do not intersect ($i_1 < j_1 < i_2 < j_2$);
- or the second bond is inside the first one ($i_1 < i_2 < j_2 < j_1$).

The picture below shows an example of the correct secondary structure of RNA. Grey lines connect the pairs of subsequent nucleotides and red lines represent hydrogen bonds.



The string *S* with even length is ***perfect*** if the corresponding RNA could have a secondary structure in which each nucleotide is contained in a bond.
The string *S* with odd length is ***almost perfect*** if we could get a perfect string by removing exactly one symbol.

Your task is to check if the given string *S* is **perfect**, **almost perfect** or **imperfect**.

For example, the picture shows **imperfect** RNA.

Input Format

The only line of the input contains string *S* of symbols A, G, C, U. The length of S is greater than 1.

Constraints for the easy version: the length of the string S does not exceed 151.
Constraints for the hard version: the length of the string S does not exceed $3 \cdot 10^5 + 1$.

Output Format

If the string *S* is perfect, the sole line of the output should contain the string «perfect» (without quotes). If the string S is almost perfect, the sole line of the output should contain the string «almost perfect» (without quotes). Otherwise, the sole line should contain «imperfect» (without quotes).

Examples

Sample Input 1

```
UGCA
```

Sample Output 1

```
perfect
```

Sample Input 2

```
AGUCU
```

Sample Output 2

```
almost perfect
```

Sample Input 3

```
CAGUU
```

Sample Output 3

```
imperfect
```

Limits

Memory limit per test: 256 Megabytes
Time limit per test: 2 seconds

Step 3

The Secondary Structure of RNA (Easy)

Constraints for the easy version: the length of the string S does not exceed 151.

Full testset could be downloaded by a link: https://stepik.org/media/attachments/lesson/39303/tests-easy.zip.

Sample Input 1:

```
UGCA
```

Sample Output 1:

```
perfect
```

Sample Input 2:

```
AGUCU
```

Sample Output 2:

```
almost perfect
```

Sample Input 3:

```
CAGUU
```

Sample Output 3:

```
imperfect
```

Step 4

The Secondary Structure of RNA (Hard)

Constraints for the hard version: the length of the string S does not exceed $3 \cdot 10^5 + 1$.

Full testset could be downloaded by a link: https://stepik.org/media/attachments/lesson/39303/tests-hard.zip.

Sample Input 1:

```
UGCA
```

Sample Output 1:

```
perfect
```

Sample Input 2:

```
AGUCU
```

Sample Output 2:

```
almost perfect
```

Sample Input 3:

```
CAGUU
```

Sample Output 3:

```
imperfect
```

To solve this problem please visit https://stepik.org/lesson/39303/step/4

Step 5

Editorial

Easy version

The easy problem, for someone who knows the basic algorithms in Bioinformatics, should resemble the classical problem of RNA folding but in simplified form. The solution is based on Dynamic Programming and works in $O(|S|^3)$ time, where $|S|$ is the length of the string.

Let $dp[i][j]$ be the minimal number of unpaired bases if we fold substring of S: $S[i:j]$. This is how it should be initialized and recalculated:

1. $dp[i][i] = 1$ and $dp[i][i+1] = 0$ if $S[i]$ and $S[i+1]$ are complementary to each other, or 2, otherwise.
2. $dp[i][j] = \min_{k=i+1 \ldots j+1} (dp[i][k] + dp[k+1][j])$.
3. $dp[i][j] = \min(dp[i][j], dp[i+1][j-1])$, if $S[i]$ and $S[j]$ are complementary to each other.

If $dp[1][|S|]$ is zero, then the string is *perfect*. If $dp[1][|S|]$ is one, then the string is *almost perfect*. Otherwise, the string is *imperfect*.

Hard version

One could notice that the previous Dynamic Programming solution resembles the solutions for problems with bracket sequences. We could see a perfect string as a correct bracket sequence and we check the correctness of bracket sequences using stack!

So, to check if the string is perfect we iterate through bases and maintain a stack:

1. If the current base is complementary to the base on top of the stack, then we pop from the stack.
2. Otherwise, we put the current base into the stack.

If there is no base in the stack, then the given string is perfect.

Now, how to understand that the string is almost perfect? The simplest solution will be to delete each symbol and check the resulting string to be perfect. Unfortunately, such solution works in $O(|S|^2)$ time and does not fit into timelimit.

But the way to deal with it is quite easy. Let us perform the solution as in the case of the perfect string. We state that if the string is almost perfect then the base to delete is the middle base of the stack. We remove this base and check that the stack collapses. (In other words, we check that the stack has odd length and it holds a palindrome)

In other cases, the string is imperfect.

Note. We do not provide the proof of the hard version. You could try to do that by yourself.