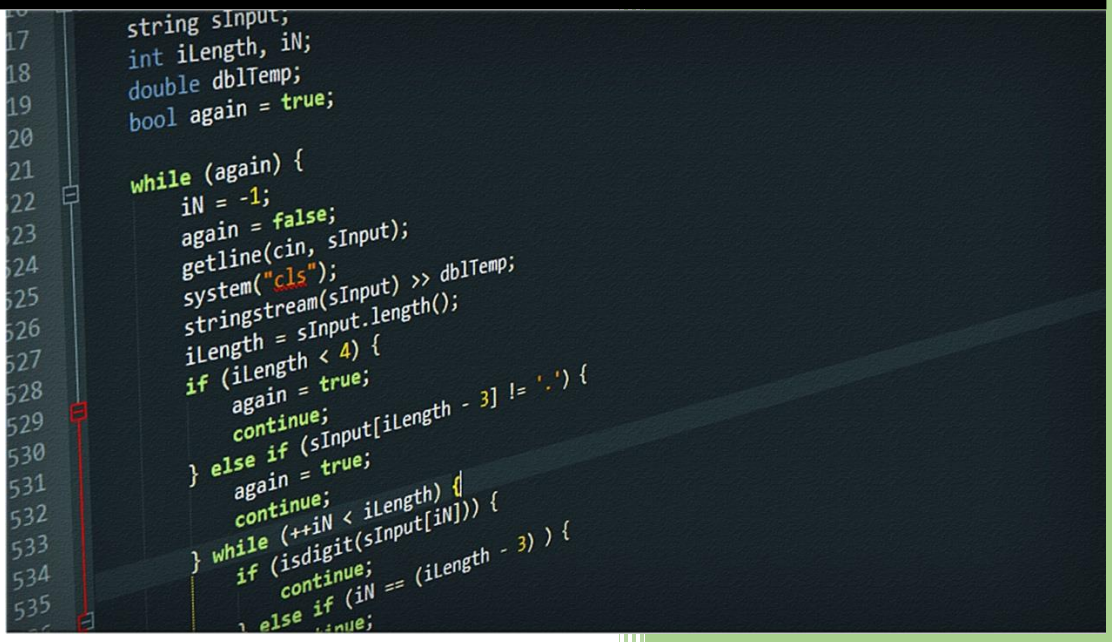
Análisis de algoritmos



Erick Efrain Vargas Romero

Prof. Franco Martínez Edgardo Adrián

Empate de cadenas

3CM2

Very Easy

Description

Given two strings a and b we define a * b to be their concatenation. For example, if a = 'abc' and b = 'def' then a*b = 'abcdef'. If we think of concatenation as multiplication, exponentiation by a non-negative integer is defined in the normal way: $a^0 = ''$ (the empty string) and $a^{n+1} = a * (a^n)$

Input

Each test case is a line of input representing s, a string of printable characters. The length of s will be at least 1 and will not exceed 1 million characters. A line containing a period follows the last test case.

Output

For each s you should print the largest n such that $s = a^n$ for some string a.

Example

Input	Output
abcd	1
aaaa	4
ababab	3

Código

```
#include <bits/stdc++.h>

using namespace std;
string s;

int buildKMPTBL(){
    vector<int> v(s.size());
    int j = 0, i = 1;
    while( i < s.size()){
        if(s[i] == s[j]){
            v[i++] = ++j;
        } else {
            if(j == 0)
                v[i++] = 0;
            else
                j = v[j - 1];
        }
    }
    return v.back();
}

int main(){

    while(cin >> s && s != "."){
        int l = buildKMPTBL();
        int n = s.size();
        if(n % (n - l) == 0)
            cout << n / (n - l) << endl;
        else
            cout << 1 << endl;
```

```
    }
    return 0;
}
```

Captura

#	Problem	Verdict	Language	Run Time	Submission Date
22379311	10298 Power Strings	Accepted	C++11	0.100	2018-12-01 05:01:23

Solución

La solución se ha realizado usando KMP, tal y como lo hemos realizado en clase. Una ves dicho lo anterior podemos encontrar la solución a este ejercicio de forma sencilla, si la longitud de la cadena módulo la longitud de la cadena menos uno es igual con cero vamos a imprimir la longitud de la cadena entre la longitud de la cadena menos uno. Si no se cumplió la condición solo encontramos una coincidencia, por tanto imprimimos uno.

Complejidad

La complejidad es prácticamente la longitud de la cadena eso nos cuesta construir la tabla usando KMP, O(|s|) y el resto del algoritmo es una simple fórmula, lo cual podemos decir que es constante, eso quiere decir que la complejidad de este algoritmo es O(|s|)

Password

Description

Asterix, Obelix and their temporary buddies Suffix and Prefix has finally found the Harmony temple. However, its doors were firmly locked and even Obelix had no luck opening them.

A little later they found a string s, carved on a rock below the temple's gates. Asterix supposed that that's the password that opens the temple and read the string aloud. However, nothing happened. Then Asterix supposed that a password is some substring t of the string s.

Prefix supposed that the substring t is the beginning of the string s; Suffix supposed that the substring t should be the end of the string s; and Obelix supposed that t should be located somewhere inside the string s, that is, t is neither its beginning, nor its end.

Asterix chose the substring t so as to please all his companions. Besides, from all acceptable variants Asterix chose the longest one (as Asterix loves long strings). When Asterix read the substring t aloud, the temple doors opened.

You know the string s. Find the substring t or determine that such substring does not exist and all that's been written above is just a nice legend.

Input

You are given the string s whose length can vary from 1 to 10^6 (inclusive), consisting of small Latin letters

Output

Print the string t. If a suitable t string does not exist, then print "Just a legend" without the quotes

Example

Input	Output
fixprefixsuffix	fix
abcdabc	Just a legend

Código

```
#include <bits/stdc++.h>

using namespace std;

int TBL[1000002];
void buildKMstrTBL(string &str)
{
    int i = 0, j = -1, m = str.size();
    TBL[0] = -1;
    while (i < m)
    {
        while (j >= 0 && str[i] != str[j])
            j = TBL[j];
        TBL[++i] = ++j;
    }
}
```

```
int main()
{
    string s;
    cin >> s;
    buildKMstrTBL(s);
    int ans = TBL[s.size()];
    bool b = false;
    while (ans > 0)
    {
        for (int i = s.size() - 1; i > ans && !b; i--)
            b = TBL[i] >= ans;
        if (b)
            break;
        ans = TBL[ans];
    }
    if (b)
        cout << s.substr(0, ans);
    else
        cout << "Just a legend";
    cout << endl;
}
```

Solución

Para solucionar este ejercicio se ha hecho uso del algoritmo de KMP, básicamente debemos construir la tabla, tal y como se ha visto en clase, vamos a iterar sobre nuestra tabla construida con el algoritmo de KMP, hasta que encontremos un elemento que sea mayor que el que contiene nuestra variable ans. Si se encontró algo que es mayor ahora nuestra variable booleana b tendrá un valor de true, eso quiere decir que encontramos la respuesta, por tanto, rompemos el ciclo. Si no se ha encontrado nuestra variable ans ahora tomará el valor de lo que tenga la tabla construida con KMP en su posición ans

Si salimos del ciclo y nuestra variable booleana es verdadera vamos a extraer una subcadena de la cadena original, la cual estará en el rango de cero a ans. Si terminó el ciclo pero la variable booleana tiene un valor de false imprimimos Just a legend debido a que no se encontró respuesta alguna

Complejidad

Como sabemos el construir la tabla usando KMP es de orden O(|s|) y el ejecutar el resto del algoritmo es prácticamente lineal, debido a que recorremos toda la tabla construida. Por tanto La complejidad es O(|s|)