Instituto Politécnico Nacional

Escuela Superior de Cómputo

DISEÑO DE SOLUCIONES CON ALGORITMOS DE EMPATE DE CADENAS

Ejercicio 09

Análisis de Algoritmos

M. en C. Edgardo Adrián Franco Martínez

Grupo: 3CM3

Fecha: 18 / Mayo / 2018

Alumno:

Calva Hernández José Manuel 2017630201

Índice

Perfect Cyclic String

Redacción del ejercicio

A perfect cyclic string is a string which can be represented as the repeated concatenation of one of its substrings. In fact, all strings can be formed in such way, and perhaps with more than one possible substring; substrings also can be the original string itself.

For example: The string "`cdabcdabcdab`" can be formed with the substring "`cdab`". The string "`abababab`" can be formed with the substrings "`ab`" or "`abab`", or even "`abababab`". And the string "`qwertyuiop`" can be formed only with the substring "`qwertyuiop`" (the string itself).

Given some string, you must find the size of the minimum substring which can be used to form the original given string.

Input

The first line contains an integer number T not greater than 10^3 representing the amount of strings to process. Each of the following T lines contains a string for processing of at most 10^3 lowercase letters of the English alphabet. You can safely assume that the sum of lengths of all given strings do not exceed $5 \cdot 10^5$.

Output

For each input string you must print an integer number with an end of line, representing the size of the minimum substring which can be used to form the original given string.

Sample Input

```
5
aaaa
abababab
cdabcdabcdab
qwertyuiopasdfg
abcabcabcabcxabcabcabcabcx
```

Sample Output

```
1
2
4
15
25
```

Captura de aceptación

21487804	12916 Perfect Cyclic String	Accepted	C++11	0.000	2018-06-19 02:15:54

Explicación de la solución

Usaremos como base la tabla de prefijos que genera el algoritmo KMP, esta nos dice si existen patrones de prefijos en determinada cadena y a partir de cuál debemos de empezar a buscar en caso de que existan subcadenas, analizando esa tabla podemos observar que en caso de que existan ciclos, se formará una secuencia ascendente a partir de donde se rompa la secuencia de 0's, por ello únicamente contaremos los 0's hasta encontrar el primer uno, a partir de él comenzaremos a contar ascendentemente y en caso de que se rompa esta secuencia, significa que no es un ciclo perfecto.

Código

```
#include < bits / stdc++.h >
using namespace std;
vector < int > prefixArray(string & pattern) {
    vector < int > prefixArr(pattern.size());
    for (int i = 0, j = 1; j < pattern.size();) {
        if (pattern[i] == pattern[j]) {
            i++;
            prefixArr[j] = i;
            j++;
        } else {
            if (i != 0) i = prefixArr[i - 1];
            else {
                prefixArr[j] = 0;
                j++;
            }
        }
    }
    return prefixArr;
}
int countSubstring(vector < int > & v) {
    bool count = true;
    int result = 0, index = 0;
    for (int i = 0; i < v.size(); ++i) {
        if (count) {
            if (v[i] == 0) {
                result++;
            } else {
                count = false;
                index = v[i];
            }
        } else {
            if (v[i] != ++index) {
                return v.size();
            }
        }
    }
    return result;
}
int main() {
    string s;
    cin >> s;
    vector < int > result;
    result = prefixArray(s);
    cout << countSubstring(result) << endl;
    return 0;
}
```

Análisis de complejidad

La complejidad es muy sencilla debido a que únicamente tenemos un dato importante que es la longitud de la cadena, a partir de ella generaremos la tabla de frecuencias por medio de una ventana deslizante, lo que implica una complejidad de O(n), a continuación analizaremos esa tabla de frecuencias por medio de la función countSubstring, que es una simple iteración por esa tabla que tiene una longitud de n, por lo tanto también se tendrá una complejidad de n, una vez aclarado esto, en general la complejidad será de:

$$f_t(n) = O(n) + O(n) = O(2n) = O(n)$$