Alumno:

Calva Hernández José Manuel 2017630201

Análisis de Algoritmos

M. en C. Edgardo Adrián Franco Martínez  
Grupo: 3CM3  
Fecha: 18 / Mayo / 2018

Diseño de soluciones con algoritmos de Empate

de CadenasEjercicio 09



Instituto Politécnico Nacional

Escuela Superior de Cómputo

Índice

[Perfect Cyclic String 2](#_Toc517119424)

[Redacción del ejercicio 2](#_Toc517119425)

[Captura de aceptación 2](#_Toc517119426)

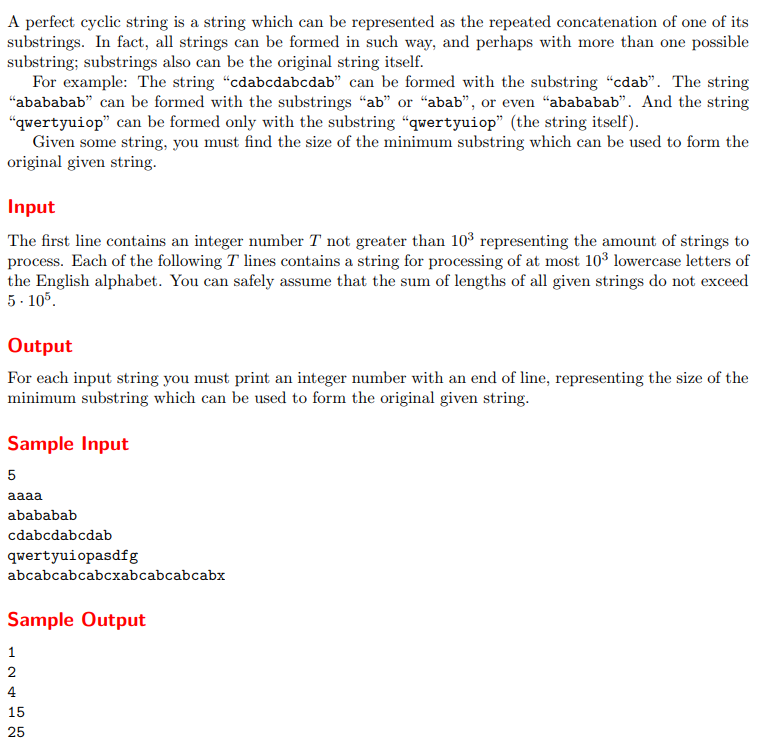
[Explicación de la solución 3](#_Toc517119427)

[Código 3](#_Toc517119428)

[Análisis de complejidad 4](#_Toc517119429)

# Perfect Cyclic String

Redacción del ejercicio



Captura de aceptación



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

1. #include < bits / stdc++.h >
2. using namespace std;
3. vector < int > prefixArray(string & pattern) {
4. vector < int > prefixArr(pattern.size());
5. for (int i = 0, j = 1; j < pattern.size();) {
6. if (pattern[i] == pattern[j]) {
7. i++;
8. prefixArr[j] = i;
9. j++;
10. } else {
11. if (i != 0) i = prefixArr[i - 1];
12. else {
13. prefixArr[j] = 0;
14. j++;
15. }
16. }
17. }
18. return prefixArr;
19. }
20. int countSubstring(vector < int > & v) {
21. bool count = true;
22. int result = 0, index = 0;
23. for (int i = 0; i < v.size(); ++i) {
24. if (count) {
25. if (v[i] == 0) {
26. result++;
27. } else {
28. count = false;
29. index = v[i];
30. }
31. } else {
32. if (v[i] != ++index) {
33. return v.size();
34. }
35. }
36. }
37. return result;
38. }
39. int main() {
40. string s;
41. cin >> s;
42. vector < int > result;
43. result = prefixArray(s);
44. cout << countSubstring(result) << endl;
45. return 0;
46. }

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: