



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Ejercicios 10: Diseño de soluciones de Empate de Cadenas

Unidad de aprendizaje: Análisis de Algoritmos

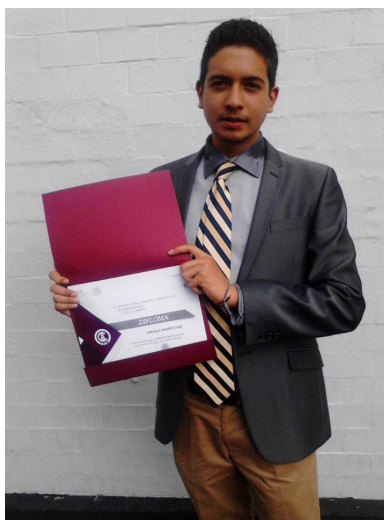
Grupo: 3CM3

Alumno:

Ramos Diaz Enrique

Profesor(a):

Franco Martínez Edgardo Adrián



4 de diciembre 2018

Índice

1	Shift the String	2
1.1	Problema	2
1.2	Análisis y solución	2
1.3	Código resultante	3
1.4	Validación del juez online	4
2	Power Strings	5
2.1	Problema	5
2.2	Análisis y solución	5
2.3	Código resultante	6
2.4	Validación del juez online	7

1. Shift the String

1.1. Problema

Nos dan dos cadenas **A** y **B** de la misma longitud. Cada cadena contiene **N** letras en minúsculas (de la 'a' a la 'z'). Una operación de cambio o "shift" eliminará el primer carácter de una cadena y agregará el mismo carácter al final de esa cadena. Por ejemplo, después de realizar una operación de shift en una cadena 'abcd', la nueva cadena será 'bcda'. Si realiza esta operación dos veces, la nueva cadena será 'cdab'. Necesitamos realizar algunas (quizá ninguna) operaciones shift en la cadena **B** para maximizar la longitud del prefijo común más largo entre **A** y **B**. Si se puede encontrar más de un resultado, debemos elegir el que utilice el menor número de operaciones shift.

Entrada

La primera línea de la entrada contiene un solo número entero **N**. La segunda y tercer línea contienen las cadenas **A** y **B** de mismo tamaño respectivamente.

Salida

Contiene un solo entero que es el mínimo número de operaciones de desplazamiento o shift.

1.2. Análisis y solución

En la cadena **A** no haremos ningún tipo de operación, la que nos interesa es la segunda cadena, la **B**.

Primero tenemos que recorrer el arreglo **B**, en busca de algún carácter que coincida con el primer carácter del arreglo **A**. Nos piden llegar al prefijo más largo entre ambos, recordemos que el prefijo de una cadena es la subcadena que ésta más a la izquierda de la cadena, y siempre inicia con el primer carácter.

Una vez que encontramos que hay un empate en la cadena **B** con respecto a la **A**, iniciamos el algoritmo KMP entre ambas cadenas desde ésta posición. Mientras el empate en la secuencia a partir de éste punto no se rompa (es decir, que los caracteres no coincidan en algún punto de la secuencia al momento), se irán haciendo las operaciones shift necesarias para ir "acomodando" este empate de cadenas al principio de la cadena **B**, e ir construyendo el prefijo común más largo.

Una vez llegamos al final de la cadena **B** y no quedan más caracteres por comparar, guardamos el tamaño del prefijo común encontrado al momento, y las operaciones shift realizadas.

El algoritmo KMP continuará hasta que no encuentre empate entre caracteres. En cada iteración, si la longitud del nuevo prefijo encontrado es mayor al que se tenía, se asigna éste al anterior y se actualiza el número de operaciones shift, que son las iteraciones que se han hecho hasta ese momento.

1.3. Código resultante

La complejidad del algoritmo es la misma que la del algoritmo KMP: $O(2n)$. Puede parecer que es cuadrática debido al while anidado, pero éste posee una complejidad logarítmica, debido al modulo que se aplica sobre su contador, por lo que la lineal del while más externo le gana.

```
1  #include <stdlib.h>
2  #include <string.h>
3  #include <stdio.h>
4
5  int main() {
6      int n;
7      scanf("%d", &n);
8      char *A = (char*)calloc(n+1, sizeof(char));
9      char *B = (char*)calloc(n+1, sizeof(char));
10     scanf("%s %s", A, B);
11     //Prefijo: subcadena a la izquierda
12     int prefix = 0, shift = 0;
13     int i = 0;
14     while(i < n){
15         //Revisar el primer caracter del prefijo de A
16         if(B[i] == A[0]){
17             //Revisar los siguientes caracteres del prefijo, si
18             //    existen (KMP)
19             if(B[(i + prefix) % n] == A[prefix]){
20                 int j = 0, aux = 0;
21                 //Hacemos las shift operations mientras el prefijo
22                 //    coincide
23                 while(B[i + j] == A[j]){
24                     aux++;
25                     j = (j + 1) % n;
26                 }
27                 /*Si la longitud del nuevo prefijo encontrado es mayor,
28                 //    actualizamos valores*/
29                 if(aux > prefix){
30                     prefix = aux;
31                     shift = i;
32                 }
33             }
34             i++;
35         }
36         printf("%d\n", shift);
37         return 0;
38     }
```

1.4. Validación del juez online

The first screenshot shows the CodeChef website with the user 'brokenerk' logged in. The page displays 'brokenerk's SUBMISSIONS FOR TASHIFT'. A red box highlights the following submission data:

ID	Date/Time	User	Result	Time	Mem	Lang	Solution
21749591	52 min ago	brokenerk	✓ 100 [100pts]	0.00	24.5M	C	View
21749584	56 min ago	brokenerk	✓ 100 [100pts]	0.01	24.5M	C	View
21749564	1 hours ago	brokenerk	✗	0.00	24.5M	C	View

The second screenshot shows the 'Successful Submission' page for the 'Shift The String' problem. A red box highlights the 'Correct Answer' message and the following table of sub-tasks:

Sub-Task	Task #	Score	Result (time)
1	0	NA	AC (0.000000)
1	1	NA	AC (0.000000)
1	2	NA	AC (0.000000)
1	10	NA	AC (0.000000)
Final Score - 30.000000			Result - AC
2	3	NA	AC (0.000000)
2	4	NA	AC (0.000000)
2	5	NA	AC (0.000000)
2	11	NA	AC (0.000000)

2. Power Strings

2.1. Problema

Dadas dos cadenas a y b , definimos $a*b$ como su concatenación. Por ejemplo: si $a = 'abc'$ y $b = 'def'$, entonces $a*b = 'abcdef'$. Si pensamos en la concatenación como una multiplicación, la exponenciación de un entero positivo es definida de forma normal como: $a^0 = ''$ (cadena vacía), y $a^{n+1} = a * (a^n)$.

Entrada

Cada caso de prueba es una línea de entrada representando s , una cadena de caracteres imprimibles. La longitud de s es al menos de 1 y no debe superar 1 millón de caracteres. Una línea que posea un punto va a al final del último caso de prueba.

Salida

Para cada caso de prueba s se debe imprimir la n más larga tal que $s = a^n$ para alguna subcadena a .

2.2. Análisis y solución

Para la solución de este problema, se implementó un algoritmo que es una combinación de fuerza bruta con el KMP. Nos piden hallar el número de veces en que una subcadena a se concatena, formando otra cadena s .

Para cada caso de entrada, se asume que la n más larga es 1, pues por definición: $s = s^1$. Ahora bien, vamos a tener un arreglo que va a iniciar en 2, esto quiere que asumimos que la cadena s es resultado de concatenar 2 cadenas a , es decir $s = a^2$.

Primero revisamos que la longitud de la cadena completa sea divisible entre el número de concatenaciones de a propuestas. Si no lo es, vamos aumentando las concatenaciones de la subcadena a hasta encontrar que sean divisibles.

Ya que encontramos un número de concatenación que es divisible, obtenemos el valor de la división de la longitud de la cadena completa entre el número de concatenaciones de ésta subcadena propuesta. Esta será la longitud supuesta de la subcadena a propuesta.

Posteriormente, aplicamos al algoritmo KMP a partir de éste valor obtenido. Nosotros vamos a asumir que siempre coincidirán el patrón de la cadena a en s con ayuda de una bandera **empata**. En caso contrario, negamos ésta bandera y rompemos el ciclo del KMP.

Ya al final revisamos el valor de la bandera. Si el KMP termino y se empataron las subcadenas como intuimos, le asignamos a la variable n , que son las veces en que se concatena ésta subcadena, el valor del contador de nuestro ciclo, que son las veces que intuimos la subcadena a se concatenaba, formando s .

Aquí es donde se encuentra la parte de fuerza bruta, pues vamos buscando todas las posibles combinaciones en que s podría ser el resultado de concatenar n veces varias subcadenas a de distintos tamaños, y ya solo validamos esto con el empate de cadenas. Cuando encontramos al menos carácter que no empata, rompemos todo el ciclo y probamos con otra n .

2.3. Código resultante

Si no tomamos en cuenta que se pueden evaluar m casos o cadenas (de hacerlo, la complejidad sería cúbica), la complejidad de este algoritmo para un caso es $O(\text{lenS} * \text{lenS})$, pues se va recorriendo el arreglo ingresado, y luego se le aplica el algoritmo KMP.

```

1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  char *cadena = (char*)calloc(1000000*5, sizeof(char*));
6  int main() {
7      while(scanf("%s", cadena)) {
8          /*Leemos y evaluamos n cadenas hasta hallar un punto*/
9          if(strcmp(cadena, ".") == 0) break;
10         int longitud = strlen(cadena);
11         //Es claro que toda cadena esta elevada a la 1 por defecto,
            ↳ asumimos que a es la misma cadena s
12         int power = 1;
13         /*Iniciamos desde 2*/
14         for(int i = 2; i <= longitud; i++){
15             /*Revisamos divisibilidad para nuestra n propuesta*/
16             if(longitud % i == 0){
17                 int L = longitud / i;
18                 //Asumimos que la subcadena a siempre empata con
                    ↳ las anteriores
19                 int empata = 1; //Bandera
20                 for(int j = L; j < longitud; j++){
21                     /*Algoritmo KMP*/
22                     if (cadena[j] != cadena[j % L]){
23                         /*En cuanto la subcadena no coincida con a,
                            ↳ acabamos. Probamos con otra n*/
24                         empata = 0; break;
25                     }
26                 }
27                 if(empata) {
28                     /*Vamos obteniendo el periodo o potencia de la
                            ↳ subcadena a si sí empató*/
29                     power = i;
30                 }
31             }
32         }
33         printf ("%d\n", power);
34     }
35     return 0;
36 }
```

2.4. Validación del juez online

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=9

UVa

Online Judge

Home > My Submissions

Google Custom Search Search

Main Menu

- Home
- My Account
- Contact Us
- ACM-ICPC Live Archive
- Logout

Online Judge

- Quick Submit
- Migrate submissions
- My Submissions

Google Custom Search Search

Main Menu

- Home
- My Account
- Contact Us
- ACM-ICPC Live Archive
- Logout

Online Judge

- Quick Submit
- Migrate submissions
- My Submissions
- My Statistics
- My uHunt with Virtual Contest Service
- Browse Problems
- Quick access, info and search
- Problemsetters' Credits
- Live Rankings
- Site Statistics
- Contests
- Electronic Board
- Additional Information
- Other Links

My Submissions


#	Problem	Verdict	Language	Run Time	Submission Date
22398930	10298 Power Strings	Accepted	C++	0.480	2018-12-05 00:51:14
22398917	10298 Power Strings	Accepted	C++	0.480	2018-12-05 00:43:53
22363643	10298 Power Strings	Accepted	C++	0.470	2018-11-28 03:14:39

<< Start < Prev 1 Next > End >>

Display # 30 Results 1 - 3 of 3

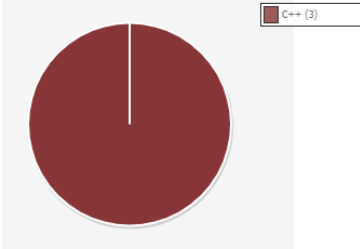
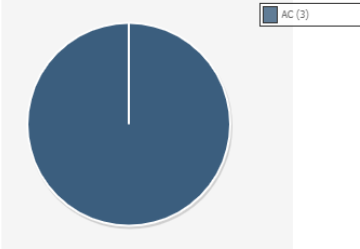
User Statistics

Enrique Ramos (brokenerk)



3 SUBMISSIONS 1 PROBLEMS TRIED 1 PROBLEMS SOLVED

2018-11-28 FIRST SUBMISSION 2018-12-05 LAST SUBMISSION



Solved problems

Problem	Ranking	Submission	Date	Run time
10298	3020	22363643	2018-11-28 03:14:39	0.470