**Integrantes**

Carolina Monsalve Vásquez - 367045

Juan Camilo Restrepo Vélez - 373886

Andres Felipe Diaz - 331011

**ESTRUCTURAS DE DATOS Y ALGORITMOS**

**TALLER 4**

**1.** Consultar alternativas para implementar la función de comparación de similaridad (considerar al menos dos alternativas). Para cada alternativa indicar:

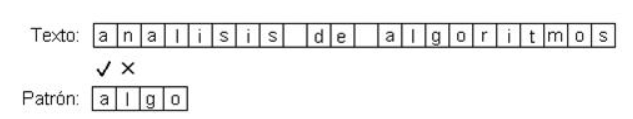
* Seudocódigo del algoritmo
* Tiempo necesario (usar como modelo de costo la comparación de caracteres)

**R\\ Nota:** los códigos y toda la teoría explicada a continuación son el producto de la investigación realizada durante la elaboración de la presente práctica y que además se encuentra citada al final del documento.

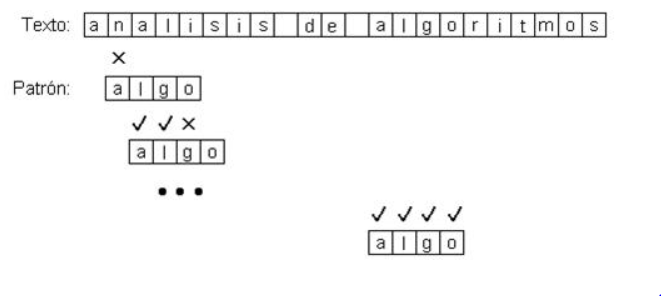
**Algoritmo de Fuerza Bruta**

* **Características:** se identifica principalmente por ser el algoritmo más simple posible. Consiste en probar todas las posibles posiciones del patrón en el texto, requiriendo un espacio constante y realizando siempre saltos de un carácter. Este compara de izquierda a derecha y la búsqueda del patrón se realiza en un tiempo O(mn), donde son hechas 2n comparaciones previstas de los caracteres del texto.
* **Lógica:** en su lógica se puede observar que se sitúa el patrón en la primera posición, y se compara carácter a carácter hasta encontrar un fallo o llegar al final del patrón; posteriormente, se pasa a la siguiente posición y se repite el proceso que finaliza al alcanzar el final del texto. Cabe resaltar que no existe un preprocesamiento del patrón ni un espacio extra constante además del espacio asignado al patrón y al texto.
* **Descripción:** la búsqueda consiste en la comparación de todas las posiciones del texto entre 0 y el n-m, si una ocurrencia del patrón corresponde o no. De encontrarse una no ocurrencia, o una ocurrencia total del patrón, salta de un carácter hacia la derecha.

**Por ejemplo:** se alinea la primera posición del patrón con la primera posición del texto, posteriormente se comparan los caracteres uno a uno hasta finalizar el patrón, esto es, se encontró una ocurrencia del patrón en el texto, o hasta que sea encontrada una discrepancia.



Si se detiene la búsqueda por una discrepancia, se mueve el patrón en una posición hacia la derecha y se intenta nuevamente.



**Así,**

public class Brute {

public static int **search1**(String pat, String txt) {

int m = pat.**length**();

int n = txt.**length**();

for (int i = 0; i <= n - m; i++) {

int j;

for (j = 0; j < m; j++) {

if (txt.**charAt**(i+j) != pat.**charAt**(j))

break;

}

if (j == m) return i;

}

return n;

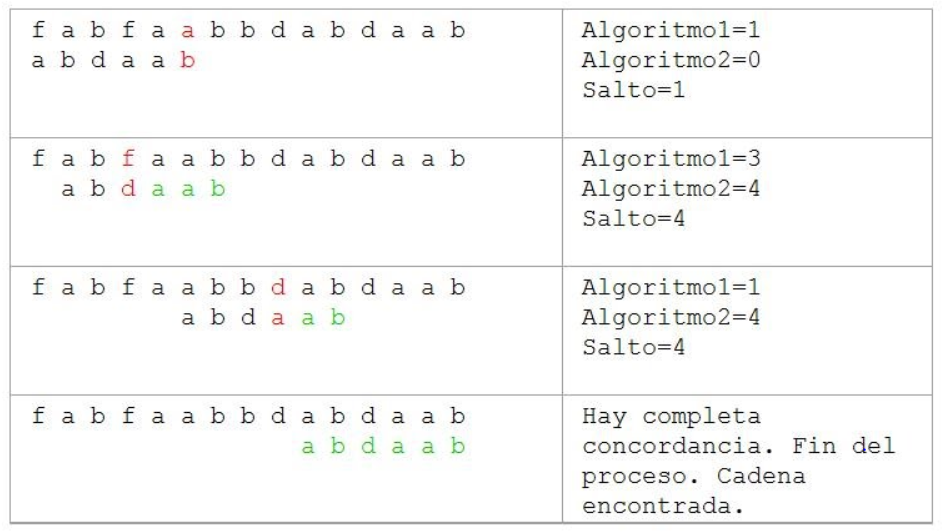
}

* **#Comparación de caracteres**: peor caso ~ nm

**Algoritmo Boyer Moore (BM)**

* **Características:** la comparación es realizada de derecha a izquierda y de haber una discrepancia en el último carácter del patrón y el carácter del texto no aparece en todo el patrón, entonces este se puede mover m posiciones sin hacer ninguna comparación extra. Dado que no es necesario la comparación de los primeros m-1 caracteres del texto, podría realizarse una búsqueda en el texto con menos de n comparaciones. Si el carácter discrepante del texto se encuentra dentro del patrón, este podría desplazarse a un número menor de espacios. Reconociendo además que el tiempo promedio del algoritmo es de O (n log(m/n)) y generando un tiempo de O(mn) en el peor caso.
* **Lógica:** presentada la discrepancia de un carácter, el carácter del texto es comparado con el patrón de búsqueda para determinar un salto hacia la derecha, pero de no existir el carácter en el patrón de búsqueda se salta la cadena completa. Si la discrepancia se produce posterior a repetidas coincidencias se salta en función de la repetición de patrones en la secuencia de búsqueda y se alinea de nuevo usando ese valor. Por último, cabe resaltar que se toma como salto el mayor de los valores.

**Por ejemplo:**



**Así,**

public int **search**(String txt) {

int m = pat.**length**();

int n = txt.**length**();

int skip;

for (int i = 0; i <= n - m; i += skip) {

skip = 0;

for (int j = m-1; j >= 0; j--) {

if (pat.**charAt**(j) != txt.**charAt**(i+j)) {

skip = Math.**max**(1, j - right[txt.**charAt**(i+j)]);

break;

}

}

if (skip == 0) return i;

}

return n;

}

* **#Comparación de caracteres:** peor caso ~ nm, caso promedio ~ nLog(m/n)

**3**. Estimar la complejidad algorítmica de la operación get, asumiendo una colección de N términos y uno de los algoritmos de comparación analizados en el punto 1.

**R\\** **Modelo de costo** #Comparaciones de caracteres

**public** Bag<String> get(String pat){

Bag<String> similares=**new** Bag<String>();

String txt;

pat = pat.toUpperCase();

**int** m = pat.length();

**for**(String t: coleccion) {

txt = t.toUpperCase();

**int** n = txt.length();

**for**(**int** i = 0; i <= n-m; i++) {

**int** j;

**for**(j = 0; j < m; j++)

**if**(pat.charAt(j) != txt.charAt(i+j))

**break**;

**if**(j==m) similares.add(t);

}

}

**return** similares;

}

|  |  |  |
| --- | --- | --- |
| **#Comparaciones de caracteres get()** | **#Comparaciones de caracteres BoyerMoore.search()** | **Orden de crecimiento** |
| Peor caso ~ N(nm) | Peor caso ~ N(nm) | Cúbico |

En ambos casos consta multiplicar el tamaño N por el tiempo que requiere el algoritmo que se utilizará, es decir Tamaño\_Coleccion \* Tiempo\_Base\_Algoritmo\_Utilizar

**Referencias**

BoyerMoore.java. (2019). Retrieved 29 October 2019, from https://algs4.cs.princeton.edu/53substring/BoyerMoore.java.html

Brute.java. (2019). Retrieved 2 October 2019, from <https://algs4.cs.princeton.edu/53substring/Brute.java.html>

FuerzaBrutaAlgorithm (2019). Retrieved 209 October 2019, from <https://sites.google.com/site/busquedasecuencialdetexto/algoritmo-fuerza-bruta>

BoyerMooreAlgorithm (2019). Retrieved 209 October 2019, from <https://sites.google.com/site/busquedasecuencialdetexto/algoritmo-boyer-moore>