***Reporte Proyecto parte 1 Dalgo***

***Andres Fernando Galvis***

***Josue Rivera***

***Luimarco Daniel Carrascal***

***1.Algoritmo:***

Debido que no pudimos desarrollar una implementación DP que funcionara correctamente, decidimos usar un algoritmo Greedy, que va a funcionar para la mayoría de los casos.

Mediante la examinación de los casos ejemplos pudimos identificar que los cambios generados al inicio y al final de la cadena son los afectan más el valor de salida.

Teniendo esto en cuenta todos los algoritmos consecuentes que se desarrollaron tuvieron 2 partes transformación y conteo. La cuenta de las subcadenas realmente no se cambió en las diferentes iteraciones probadas.

Todas las iteraciones fueron probadas un mínimo de 20 veces, con los archivos de prueba P1_cases.OUT.

En la primera iteración se realizó la transformación de tal manera que se haría un recorrido en la cadena X, en un ciclo que ocurre m veces, de manera intercalada hacia adelante o hacia atrás dependiendo de numero de posiciones *m,* recorriendo la cadena hasta que el carácter donde está el índice no sea igual a uno de los caracteres de la subcadena, evaluando el primero en el recorrido para adelante (m es par) y el ultimo en el recorrido hacia atrás (m es impar). Esta iteración tuvo un porcentaje de aciertos del 70,41% respecto a los resultados del archivo de prueba.

La segunda iteración se buscó cambiar el recorrido de tal manera que, en vez de tener 2 recorridos distintos a realizar, dependiendo de una sola evaluación, se tendría un solo recorrido con 2 índices diferentes uno al inicio y uno al final de la cadena, de tal manera que se evaluaba con dichos índices si estos eran iguales a uno de los caracteres de la subcadena se pasaba comparar al otro índice en el otro extremo del rango consecutivamente reduciéndolo, esto se haría m veces. Esta iteración tuvo un porcentaje de aciertos del 56,23% respecto a los resultados del archivo de prueba.

La tercera iteración, que es la actual, en un ciclo que ocurre m veces, se preguntan la cantidad de veces que aparecen los caracteres de la subcadena en la cadena y se guardan como variables. Luego se evalúa cual es mayor y correspondientemente se realiza un recorrido desde el inicio o el fin de la cadena donde se va a buscar el primer carácter que no sea el primer o segundo carácter de la subcadena y se remplaza por este respectivamente. Esta iteración tiene un porcentaje de éxito del 76,47%.

Puedo establecer el siguiente pre y pos-condición:

$$Pre: \{m >= 0\}$$

$$Pos\{f(X, Y, m) = \ máxima\ cantidad\ de\ subcadena\ Y\ en\ cadena\ X\}$$

**2.Análisis de complejidades espacial y temporal:**

- ***Complejidad Espacial:***

La complejidad espacial del algoritmo Greedy de solución si ignoramos la lectura de los datos y su salida, tomando *n* como la longitud de la cadena, es dada por las siguientes ecuaciones:

$$f(X,Y,m) = transformacion(X,Y,m) + conteo(X,Y)$$

$$transformacion(X,Y,m) = m(n(k))$$

$$conteo(X,Y) = n^2$$

$$f(X,Y,m) = n * m + n^2$$

Entonces la complejidad dependería si m es mayor o menor a n:

$$Si\ n\ >= m\ , entonces\ será\ O(n\text{^}2)$$

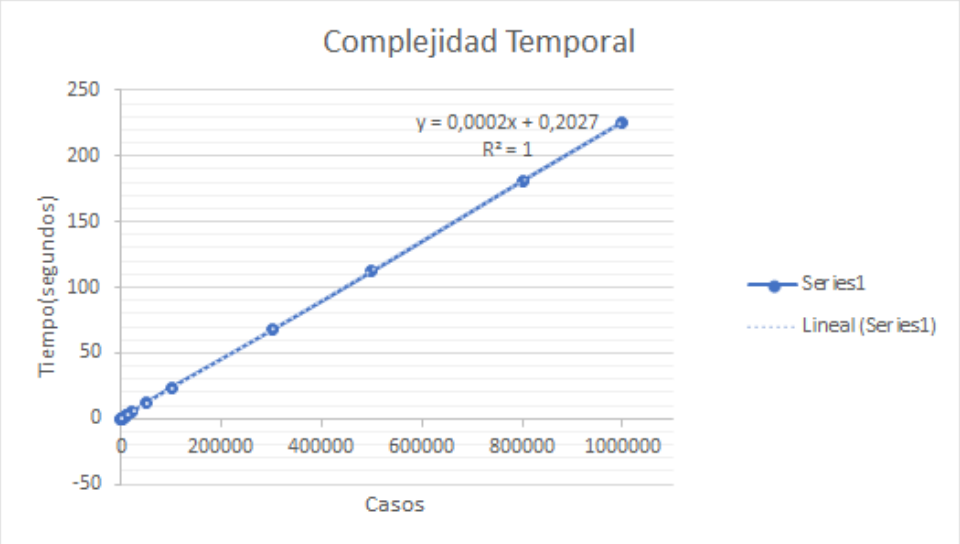
$$Si\ n < m\ , entonces\ sera\ O(n * m)$$

Y como dicho algoritmo se usará para resolver G casos, la complejidad total del programa seria la complejidad anterior de acuerdo con el valor de $m$, multiplicada por G casos.

- ***Complejidad temporal:***

Para establecer una complejidad temporal se realizaron varias pruebas con casos generados al azar, de tal manera que la longitud de las cadenas estuviera entre 20 y 400 caracteres, la subcadena debe ser formada por caracteres existentes en la cadena y los movimientos está en un rango entre 10 y 100.

Se realizaron 25 ejecuciones para cada prueba de 10 a un millón de casos y con el promedio de los tiempos dados se graficó el crecimiento de tiempo respecto a la cantidad de casos dados como entrada.




| Casos Generados | Tiempo(segundos) |
|---|---|
| 10 | 0,001 |
| 100 | 0,0254 |
| 1000 | 0,246 |
| 5000 | 1,19 |
| 8000 | 1,98 |

| | |
|---|---|
| 10000 | 2,45 |
| 20000 | 4,8 |
| 50000 | 12,18 |
| 100000 | 23,65 |
| 300000 | 67,32 |
| 500000 | 112,04 |
| 800000 | 181,37 |
| 1000000 | 226 |

Con los datos obtenidos de nuestras ejecuciones hemos podido establecer que, al buscar una tendencia de estos, podemos identificar una tendencia líneal en el caso de ejecución promedio del algoritmo.

***3.Escenarios de Comprensión:***

***ESCENARIO 1: Suponga ahora que un movimiento no aplica solo a una posición, sino a todo . Es decir, un movimiento cambiaría toda ocurrencia de un carácter  en . Por ejemplo, para  = " " un posible movimiento seria cambiar el carácter "" por el carácter "", dando como resultado  =" ".***

***(i) que nuevos retos presupone este nuevo escenario si aplica-?***

Este nuevo escenario propone retos como que las apariciones de la subcadena en la cadena que maximiza el número de ocurrencias puede que solo permita un reemplazo por ejemplo si tenemos en cuenta, aaaata como la cadena X y la subcadena Y xt, nos damos cuenta de que una vez reemplazamos la primera posición de la subcadena en la cadena nos da como resultado xxxxtx y con un solo reemplazo llega a un buen número de ocurrencias de la subcadena en la cadena más grande, sin embargo podría haber una mayor cantidad de ocurrencias si pudiéramos por ejemplo hacer 4 movimientos donde queramos y quedaría xxxatt donde encontramos que hay 6 ocurrencias un numero de ocurrencias que nunca se alcanzaría en este nuevo escenario dada esa cadena X. Lo que quiere decir que en este nuevo escenario puede haber cadenas que no maximicen el número de apariciones de la subcadena que se podrían obtener con el escenario base del proyecto, hay un número máximo de apariciones de la subcadena en la cadena X que sin importar el número de reemplazos que me den no me cambian el número de apariciones contrario al caso base del proyecto.

***(ii) que cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?***

La solución planteada entonces debería ya no insertar el primer elemento de la subcadena en las posiciones que van desde el inicio de la cadena X y el segundo elemento en las posiciones desde el final de la cadena X, cuyo orden dependerá de una comparación de la cantidad de los caracteres individuales de la subcadena en la cadena X.

En lugar de esto, se debería buscar en que posiciones de la subcadena se encuentra el elemento que va a ser reemplazado por el elemento a reemplazar y realizar el reemplazo, el contador tal como fue

definido en el código no cambiaría. Ya se tiene implementado que se comienza en la primera posición de la cadena y se revisa si es igual a la primera posición de la subcadena si no es así se reemplaza, en este caso además de esto se necesita también revisar si aparece más de una vez si aparece más de una vez vale más la pena realizar el reemplazo y utilizar un movimiento en ello que si aparece una sola vez, esto mismo habría que revisarlo para el final de la cadena X y la última posición de la subcadena Y.

***ESCENARIO 2: Ya no hay límite de movimientos.***

***(i) que nuevos retos presupone este nuevo escenario si aplica-?***

Teniendo en cuenta que ya no hay restricción de movimientos consideramos que el reto se encuentra en hacer el algoritmo lo más eficiente posible esto quiere decir que el número de movimientos sea el mínimo posible para obtener el mayor número de ocurrencias del substring en la cadena X. Es decir, se impone un reto de determinar cuál es ese mínimo número de movimientos que maximizan las apariciones de la subcadena en la cadena.

***(ii) que cambios -si aplica- le tendría que realizar a su solución para que se adapte a este nuevo escenario?***

Ya no tendríamos la entrada del número de movimientos máximos que podemos hacer, pero consideramos que como está planteado el algoritmo también podría funcionar sin tener el número de movimientos disponibles que tenemos. Seguiríamos reemplazando hasta tener el string de tamaño Length(X) por ejemplo xxxttt de esa forma maximizando el número de ocurrencias del substring Y xt. Habría que cambiar el metodo de maximizar las ocurrencias del substring para que ya no recibiera el parámetro m de numero de reemplazos máximos permitidos y en lugar de esto se podría llevar un contador con el número de reemplazos realizados.