ALGORITMOS Y ESTRUCTURAS DE DATOS II

Práctica 1

José Miguel Sánchez Almagro:
josemiguel.sancheza@um.es

Francisco Sánchez Gambín:
franciscojose.gambin@um.es

Grupo 1.2

Pseudocódigo

El pseudocódigo se encuentra escrito y explicado en el archivo pseudocodigo.txt en la carpeta del proyecto.

Para almacenar las soluciones del problema en los distintos métodos empleados, se ha optado por el uso de un array de enteros de 6 posiciones, ya que para la solución del problema mediante el método de combinar hacemos uso de la primera aparición de la cadena b en a con su correspondiente número de repeticiones, y de la última aparición de la cadena b y su correspondiente número de repeticiones.

- **Algoritmo directo**: En este procedimiento se encuentra el algoritmo que encuentra la solución directa del problema para el mayor número de repeticiones y posición de la aparición de una cadena_b en una cadena_a.
- **Función comprobar**: Esta función se encarga de comprobar si las cadenas a y b forman entre ellas un caso especial que no puede ser resuelto en el combinar.
- **Función pequenio**: Esta función devolverá un booleano indicando, a partir de la cadena_a y la cadena_b, si la cadena_a es suficientemente pequeña como para analizarla con el algoritmo directo en lugar de con divide y vencerás, o si por el contrario tenemos que dividirla en dos partes.
- **Función dividir**: Esta función devuelve un entero indicando la mitad por la que se debe partir la cadena_a, para analizar cada parte por separado.
- **Procedimiento combinar**: Este procedimiento será el encargado de combinar dos cadenas que se han resuelto de manera separada. En primer lugar, analizaremos la cadena_a y cadena_b para ver si se trata de un caso especial y tiene que ser resuelto de manera independiente, si se trata de un caso especial se aplicara la solución directa para este caso. Si no se trata de un caso especial comprobaremos si hemos entre el final de la aparición de la cadena_b y el principio de la aparición de otra cadena_b. Si no se corresponde con ninguno de los dos casos anteriores tomaremos el tamaño de la cadena_b menos unos caracteres a la izquierda de la división y a la derecha para analizar esa cadena intermedia y comprobar si la combinación de esta cadena intermedia con la parte izquierda y derecha produce un mayor número de repeticiones de la cadena_b.
- **Procedimiento divide**: este procedimiento comprobara si la cadena_a es suficientemente pequeña para aplicar directamente el algoritmo de resolución directa o en caso contrario dividir la cadena en dos partes y aplicar el procedimiento de combinar para encontrar la solución.

Estudio teórico del algoritmo

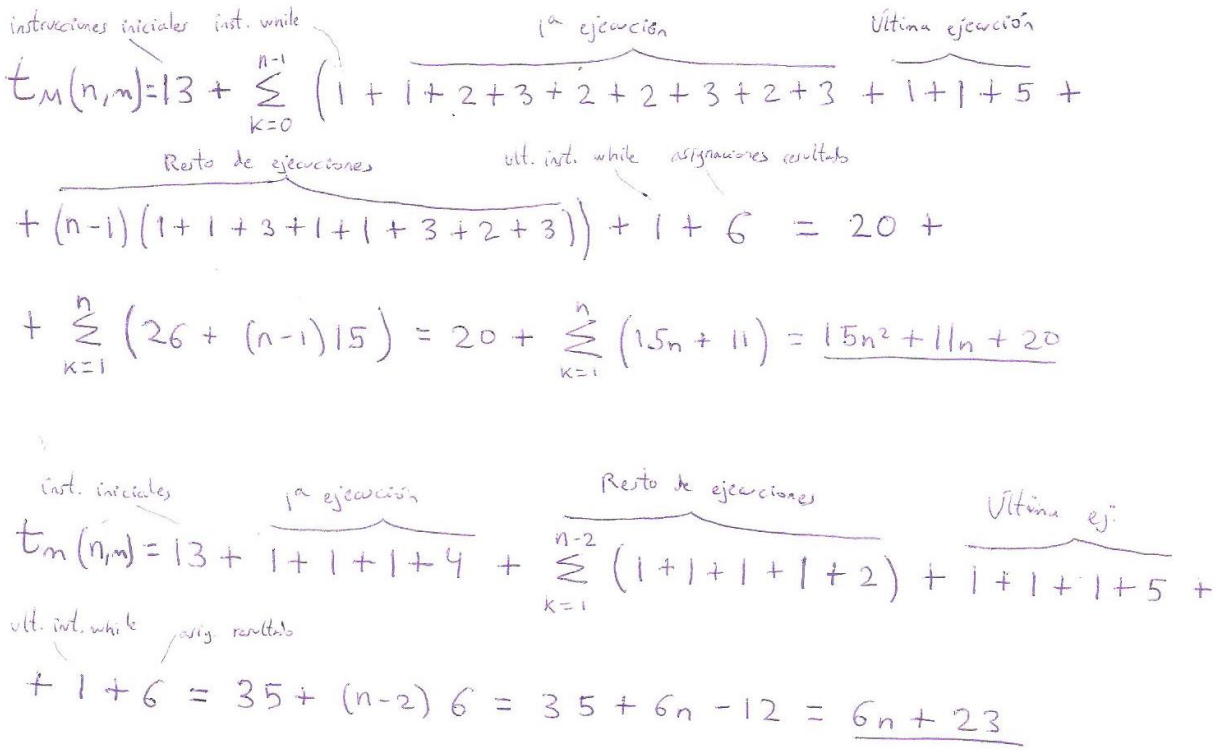
Los valores de n y m son el tamaño de la cadena_a y el tamaño de la cadena_b, respectivamente.

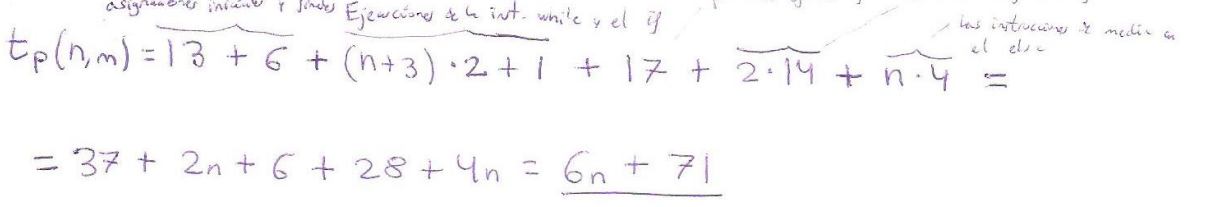
Algoritmo directo

- El **mejor caso** se dará cuando en el *while* siempre entremos al *else*, y dentro del *else* al *if*. El último *if* siempre será falso exceptuando la primera ejecución, que será verdadero, y por tanto ejecutará la instrucción que se encuentra en su interior. Además, en la última ejecución entraremos a los dos últimos *else*.
- En el **peor caso** siempre entra al *if* del *while*, y entrará en los 4º, 5º y 6º *if*s, el resto siempre serán falso, exceptuando la primera ejecución, que entrará a todos los *if*. Por último, como ocurría anteriormente, en la última ejecución siempre entraremos en los dos últimos *else*.
- Por último, en la estimación del **tiempo promedio** mediante una serie de ejecuciones experimentales, se ha llegado a la conclusión de que el número de veces que entra en el *while* estará en 3 aproximadamente, y las veces que entra al *else*, en el tamaño de cadena_a, es decir, n.

Debemos tener en cuenta que, en el **peor caso**, cada ciclo del *while* se ejecutará n+1 veces.

Una vez analizados los distintos casos, procedemos a realizar el conteo y obtener el orden.





$t_M(n,m) \in O(n^2)$

$t_m(n,m) \in \Omega(n)$ > No existe Θ

$t_p(n,m) \in O(n)$

Llegamos a la conclusión de que el peor caso empeora mucho la eficiencia respecto a los otros dos casos, pero afortunadamente no ocurrirá a menudo. Además, el tiempo promedio y el tiempo mínimo son muy parecidos, manteniendo los dos el mismo orden, por lo que el algoritmo se comportará bien en la mayoría de las situaciones.

Comprobar

Esta es una función auxiliar que se utiliza, dadas una cadena_a y una cadena_b, de detectar los casos especiales que no han podido ser resueltos de manera eficiente con el combinar, como se ha explicado anteriormente.

- El **mejor caso** será aquel en el que la cadena_b sea de tamaño 1, y por tanto no es caso especial y sale de la función de manera casi inmediata.
- El **peor caso** es aquel en el que la cadena_b sea de tamaño mayor que 1, y que además sea impar. Esto obliga a realizar dos *while*, pero además en este **peor caso** se comprobará que la cadena es especial en la segunda pasada del bucle *for*, lo cual empeora mucho más las cosas. Este será el talón de Aquiles de nuestro algoritmo.
- El **caso promedio** se comporta de manera parecida al peor caso, pero con una constante multiplicativa menor.

asignaciones iniciales 3 return

$$t_m(n,m) = 2 + 1 + 1 = 4$$

if for asignaciones dtc dtc while asignaciones dtc if dtc if 2 dtc if 3 aux=6

$$t_M(n,m) = 2 + 1 + 1 + 3 + \sum_{i=0}^{n} 3 + 2 + 3 + 1 + 1 + 1 + 1 + 1 + 1 + 1 +$$

for while inicio ult. while inicio2=fin ult. while inicio2=fin

$$+ \sum_{x=1}^{2} \left(i + 3 + \sum_{inicio}^{fin-1} (1 + 1 + 2) + 1 + 1 + 3 + \sum_{inicio2}^{fin2-1} (1 + 3) + 1 + 1 + 7 \right) -$$

ult. asig return

$$- 7 + 1 = 7 + \sum_{i=1}^{m+1} 3 + 11 + \sum_{x=1}^{2} \left(4 + \sum_{inicio}^{fin-1} 4 + 5 + \sum_{inicio2}^{fin2-1} 4 + 9 \right) - 6 =$$

$$= 12 + 3(m+1) + \sum_{x=1}^{2} \left(18 + \sum_{i=1}^{fin-inicio} 4 + \sum_{j=1}^{fin2-inicio2} 4 \right) = 12 + 3m + 3 +$$

$$+ 2 \cdot 18 + 2 \sum_{i=1}^{fin-inicio} 4 + 2 \sum_{j=1}^{fin2-inicio2} 4 = 51 + 3m + 2 \cdot \frac{m}{3} \cdot 4 + 2 \cdot \frac{2m}{3} \cdot 4 =$$

$$= 3m + 8m + 51 = 11m + 51$$

$t_m(n,m) \in \Omega(1)$

$t_m(n,m) \in O(m)$ $\Rightarrow \nexists\ \Theta$

$$t_p(n,m) = 2 + 1 + 1 + 2 + \sum_{i=1}^{m/2} 3 + 1 + \sum_{i=m/2+1}^{m} 4 + 1 + 1 + 3 + 1 =$$

(tam_caracter, iff, return 0)

$$= 13 + \frac{m}{2}3 + \frac{m}{2}4 = \frac{7}{2}m + 13$$

$t_p(n,m) \in O(m)$

Esta función no devuelve altos tiempos de ejecución, sin embargo, el tiempo promedio y el peor caso son bastante similares, lo cual tiene dos puntos de vista: el tiempo de ejecución nunca tendrá picos altos de tiempo, y en pocas ocasiones disfrutamos de tiempos de ejecución bajos como ocurre con el mejor caso, que su tiempo es irrisorio.

Combinar

Este es el procedimiento por excelencia de nuestro algoritmo. Se encarga de calcular la solución dadas una cadena_izquierda, una cadena_derecha, y una cadena_b. Esta solución será obtenida con la información de la cadena_izquierda y la cadena_derecha y el estudio de una cadena intermedia.

Sin embargo, esta función se basa en el supuesto de que esta cadena intermedia solo podrá obtener como máximo una cadena válida, y eso nos lleva a un problema, y es que existe un caso especial en el que no se ha encontrado manera de resolver la cadena intermedia con este supuesto.

El caso especial se da en las cadenas de más de un carácter: antes o después de la solución intermedia (de la cadena intermedia más si engancha con la parte izquierda o derecha) encontramos caracteres que pertenecen a la cadena_b, y si unimos estos caracteres del principio con los que hay a continuación de la cadena intermedia podemos formar otra cadena válida.

- No se ha encontrado solución a este tipo de casos con el planteamiento actual del combinar. Simplemente detectamos que se trata de un caso de este tipo o no, y lo resolvemos con el algoritmo directo, siendo este el **peor caso** del combinar.
- El **mejor caso** es aquel en el que no será necesario analizar ninguna cadena intermedia, porque la solución de la parte izquierda termina cuando empieza la solución de la parte derecha. Simplemente habrá que realizar algunos cálculos para ajustar la solución y listo.
- El **caso promedio** es precisamente el mas común: Si no se cumple ninguna de las condiciones anteriores debemos escoger una cadena intermedia de la cadena_a, resolverla con el algoritmo directo y realizar los ajustes necesarios para devolver la solución correcta.

El conteo quedaría así:

$$t_m(n,m) = 6 + (t_m(\text{COMPROBAR}); + 1 + 1 + 1 + 3 + 3 + 2 + 2 + 6 =$$

(asig. iniciales; apcad = 0; if; asig.; if y ++ int.; asig. finales)

$$= 25 + (4) = 29$$

$t_M(n,m) = 6 + t_M(\text{COMPROBAR}) + 1 + 1 + t_M(\text{DIRECTO}) = 8 + 11m + 51 +$ (especial > 0)

$+ 15n^2 + 11n + 20 = 15n^2 + 11n + 11m + 79$

$t_P(n,m) = 6 + t_P(\text{COMPROBAR}) + 1 + 2 + 6 + \sum_{i=\frac{n}{2}-m-1}^{\frac{n}{2}-1} 3 + 1 + \sum_{j=0}^{m-1} 3 + 1 +$ (j's)

$+ t_P(\text{DIRECTO}) + 6 + 2 + 1 + 4 + 3 + 5 + 6 = 44 + \frac{7}{2}m + 13 +$ (paridad n==0)

$+ (m+1)3 + m3 + 6(m+1+m) + 71 = 128 + \frac{7}{2}n + 3m + 3 + 3m +$

$+ 6m + 6 + 6m = 21'5m + 137$

$t_m(n,m) \in \Omega(1)$

$t_M(n,m) \in O(n^2)$ $\Rightarrow \nexists\ \Theta(n^2)$

$t_P(n,m) \in O(m)$

El mejor caso es tremendamente rápido, y afortunadamente no es poco común. El caso promedio también mantiene un orden de ejecución bajo, por lo que combinar gozará usualmente de unos tiempos de ejecución bajos. Conviene recordar que n pertenece al tamaño de la cadena_a y m al tamaño de la cadena_b, por lo que los órdenes de ejecución con m son mucho menores que los que tienen n.

Sin embargo, nos encontramos con un problema tremendamente grande, y ese es el peor caso. Cuando tenemos unas cadenas que cumplen la condición de caso especial, este se tiene que resolver con el algoritmo directo y devuelve un orden de ejecución muy elevado, ya que este se basa en n. Además, en el peor caso también hemos tenido en cuenta el peor caso del algoritmo directo, por lo que el problema se acentúa.

Directo vs Combinar

Una vez analizados los casos Directo y Combinar, podemos obtener conclusiones de cuando escoger cada uno. Estimando una relación en la que por cada cadena de tamaño n tenemos una cadena m de tamaño 1/10*m (promedio), estudiamos el caso en el que conviene ejecutar combinar en lugar del algoritmo directo:

$t_P(n,m)[\text{DIRECTO}] = t_P(n,m)[\text{COMBINAR}]; \quad 6n + 71 = 21'5m + 137;$

$6 \cdot 10m + 71 = 21'5m + 137; \quad 60m - 21'5m = 137 - 71; \quad 38'5m = 66;$

$m = 1'71 \simeq 2 \quad y \quad n = 17$

Por lo tanto, vemos que cuando la cadena_a tiene un tamaño mayor a 17 caracteres, y la cadena_b tiene un tamaño mayor o igual a 2 caracteres, es preferible dividir y luego aplicar combinar.

Divide y vencerás

Una vez obtenidos los órdenes anteriores podemos obtener al análisis del tiempo de ejecución del algoritmo completo Divide y Vencerás:

$t_m(n,m) = 1 + (t_m(n,m)\,[\text{DIRECTO}]) = 1 + (6n+23) = 6n+24$

dividir

$t_M(n,m) = 1+2+1+1+\sum_{i=0}^{n-1}(3)+1+\sum_{i=m}^{n}(3)+1+2+2\cdot$

$\cdot(t_M(n,m)\,[\text{DIRECTO}]) + 3 + (t_M(n,m)\,[\text{COMBINAR}]) + 4 = 16+3m+(n-m+1)3+$

$+2\cdot((15n^2+11n+20)+(11m+51)) = 67+14m+3n-3m+3+$

$+30n^2+22n+40 = 110+11m+25n+30n^2 = 30n^2+25n+11n+110$

$t_p(n,m) = 1+2+1+1+3m+1+3(n-m+1)+1+2+$

$+2\cdot(t_p(n,m)\,[\text{DIRECTO}]) + 3 + (t_p(n,m)\,[\text{COMBINAR}]) + 4 = 16+3m+$

$+3n-3m+3+2\cdot((6n+71)+(\frac{7}{2}n+13)) = 32+\frac{7}{2}m+3n+$

$12n+142 = 15n+\frac{7}{2}n+174$

$t_m(n,m) \in \Omega(n)$

$t_M(n,m) \in O(n^2)$

$t_p(n,m) \in O(n)$

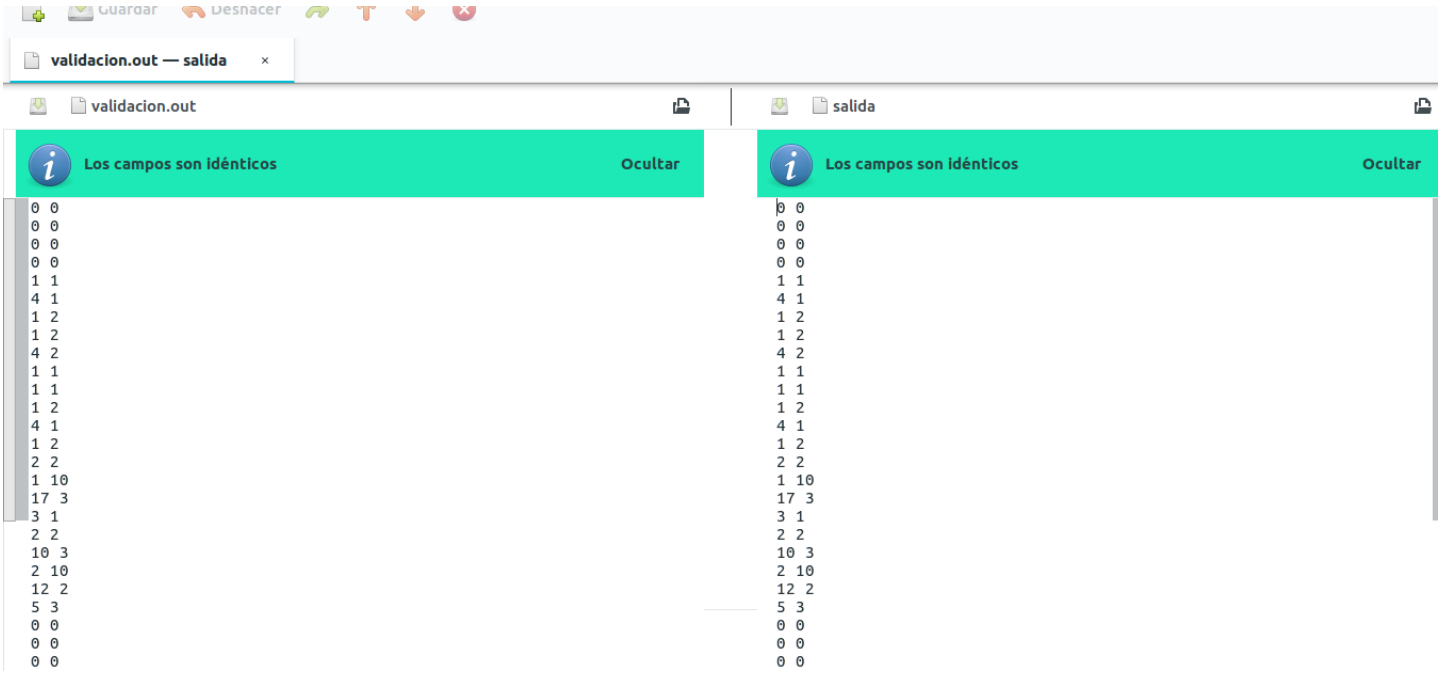
Podemos concluir este apartado afirmando que tenemos un programa eficiente y equilibrado, pero con un serio problema en algunos tipos de cadenas, que no suelen ser recurrentes, pero empeoran mucho la eficiencia.

Validación del algoritmo

La validación del algoritmo se lleva a cabo a través de la realización de dos experimentos.

El primer experimento consiste en la generación de un fichero, de forma manual por nuestra parte, de aquellos casos que consideramos de especial importancia para nuestro algoritmo de combinar, ya sea por la especial forma de la cadena_b por ejemplo abaa o por la división de la cadena_a de tal forma que parte la cadena_b de diversas formas. Estos casos se encuentran en el fichero “validacion.in” con sus correspondientes soluciones en el fichero “validacion.out”. Para comprobar el correcto funcionamiento de nuestro algoritmo ejecutaremos el programa introduciendo como fichero de entrada el fichero “validacion.in” y generando un fichero con las soluciones realizadas por nuestro algoritmo. Este fichero será comparado con el fichero de validacion.out mediante el comando meld, que nos dirá si hay alguna diferencia entre los dos ficheros de soluciones.

El segundo experimento consistirá en la ejecución del programa generar. Este programa genera aleatoriamente cadenas a y b, las cuales almacena en el fichero “generacion.in” y aplica sobre estas nuestro algoritmo de resolución directa guardando las soluciones en el fichero “generacion.out”. Una vez tengamos estos dos ficheros procederemos como en el experimento anterior, introduciremos como fichero de entrada para nuestro programa combinar el fichero generacion.in y generaremos un fichero con las soluciones que genere este programa. Una vez tengamos el fichero anterior pasaremos a comprobar si hay alguna diferencia con el fichero generado con nuestro método de divide y vencerás y el fichero “generacion.out”, empleando el comando meld.



Estudio experimental

Vamos a realizar diversas pruebas generadas con el programa generar.c, cuyo código se adjunta en la entrega. Este programa se encarga de generar cadenas_a y cadenas_b. Ademas crea dos tipos de cadenas:

- Cadenas_a y cadenas_b completamente **aleatorias**.
- Cadenas_a aleatorias y cadenas_b que son **subcadenas** de su cadena_a.

Las cadenas **aleatorias** tendrán muchas menos coincidencias que las **subcadenas**.

Vamos a realizar ejecuciones y medir su tiempo en diferentes tamaños de cadenas. Generaremos 10 cadenas_a y sus correspondientes 10 cadenas_b. A continuación, exponemos los resultados con cada tamaño:

Cadenas de tamaño 10

```
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ generar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ ./a.out
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ generar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ ./a.out
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ combinar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,004s
user    0m0,004s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,004s
user    0m0,004s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,004s
user    0m0,000s
sys     0m0,004s
```

Los resultados son prácticamente iguales en los tres intentos, 4 ms. Es un tiempo bajo que se corresponde con el tamaño de las cadenas.

Cadenas de tamaño 50

```
real    0m0,006s
user    0m0,006s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,005s
user    0m0,005s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,005s
user    0m0,000s
sys     0m0,005s
```

El tiempo aumenta muy poco respecto al experimento anterior, por lo que podemos estar satisfechos del rendimiento en estos tamaños.

Cadenas de tamaño 100

```
real    0m0,006s
user    0m0,003s
sys     0m0,003s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,007s
user    0m0,007s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,006s
user    0m0,003s
sys     0m0,003s
```

Continuamos observando que a pesar de volver a doblar el tamaño de las cadenas el tiempo se mantiene muy bajo y aumentando muy lentamente.

Cadenas de tamaño 200

```
real    0m0,010s
user    0m0,007s
sys     0m0,003s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,011s
user    0m0,011s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,011s
user    0m0,011s
sys     0m0,000s
```

Ahora volvemos a doblar el tamaño, pero el tiempo no avanza de la misma manera, el tiempo ahora se eleva más rápido y perdemos la gran eficiencia de las cadenas pequeñas.

Cadenas de tamaño 500

```
real    0m0,030s
user    0m0,027s
sys     0m0,004s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,028s
user    0m0,028s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,028s
user    0m0,024s
sys     0m0,004s
```

Ahora aumentamos el tamaño en un factor de 2’5, y el tiempo se eleva justo en ese valor. Podemos ver que el tiempo de analizar estas cadenas es 2’5 veces mayor al anterior.

Cadenas de tamaño 1000

Aquí hemos realizado 3 experimentos con distintas entradas, porque en cadenas tan grandes los tiempos pueden tener grandes variaciones, y así ha sido:

```
real    0m0,130s
user    0m0,130s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,122s
user    0m0,122s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,123s
user    0m0,119s
sys     0m0,004s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ generar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ ./a.out
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ combinar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,190s
user    0m0,190s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,192s
user    0m0,188s
sys     0m0,004s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,192s
user    0m0,184s
sys     0m0,008s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ generar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ ./a.out
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ g++ combinar.c
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,153s
user    0m0,149s
sys     0m0,004s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,142s
user    0m0,143s
sys     0m0,000s
mrchemi@MrChemi-Linux:~/0Universidad/0-ALGORITMOS_Y_ESTRUCTURAS_DE_DATOS_II/Practica/Tema_1_y_2/Version_1.0$ time ./a.out < generacion.in > salida

real    0m0,161s
user    0m0,161s
sys     0m0,000s
```

En las ejecuciones del primer archivo, los tiempos son los más bajos que vamos a obtener respecto a las otras pruebas, y sin embargo ya podemos observar que es tiempo muy elevado respecto al que se obtiene con las cadenas de 500 caracteres. El tiempo de ejecución para cadenas de estos tamaños ya se antoja muy elevado. En el segundo archivo el tiempo es tremendamente lento, casi 2 décimas de segundo. En la última prueba los tiempos son intermedios a los otros dos, por lo que tomaremos este como referencia (150 ms).

El tamaño de las cadenas ha aumentado en un factor de 2, y el tiempo de ejecución en un factor de 5. Una diferencia muy elevada, por lo tanto, concluimos que nuestro algoritmo se muestra muy eficiente para cadenas de hasta 500 o 600 caracteres, pero a partir de ese punto se eleva mucho el tiempo.

Contraste del estudio teórico y experimental

El estudio teórico realizado anteriormente deja como conclusión que el programa tiene un tiempo promedio de ejecución de orden n, elevándose en los peores casos a un orden de $n^{2,}$ debido en gran parte al empeoramiento en el tiempo de ejecución de los dos métodos principales (divide y vencerás y combinar) cuyo, pero tiempo es de orden n^2.

El estudio experimental deja patente claramente que el tiempo de ejecución es bastante estable y rápido para casos en el que el tamaño de la cadena inferior a 500 caracteres).

Ambos estudios dejan entrever que el programa es bastante estable en tiempo promedio, ya que su orden es n, y rápido para tamaños de cadena inferiores a 500.

El empeoramiento en los tiempos de ejecución a partir de cadenas de tamaño 500 puede explicarse debido a dos causas principalmente.

La primera de ellas se debe al aumento de la probabilidad de la aparición de los peores casos para nuestro problema, lo que deja un tiempo de ejecución de orden n^2, y ralentiza enormemente el tiempo de ejecución llegando a multiplicar por 5 los tiempos de ejecución, con respecto a cadenas de tamaño inferior.

La segunda causa de este empeoramiento puede explicarse debido a factores externos a los algoritmos empleados, como puede ser el equipo en el que se ejecuta el programa.

En conclusión, tanto estudio teórico como experimental nos dan a entender que se trata de un programa rápido en tiempo promedio para cadenas de un tamaño no demasiado excesivo, pese a la dificultad del problema planteado.

Conclusiones personales

Sabemos que quizá no se pedía este apartado en la memoria del proyecto, pero queríamos incluirlo, ya que durante el proceso de desarrollo del trabajo nos hemos encontrado con muchos detalles dignos de mencionar.

La resolución del algoritmo directo tuvo una dificultad media, como el resto de los elementos del ejercicio, exceptuando el combinar del algoritmo divide y vencerás. Desde el principio nos encontramos con diversos problemas en el planteamiento del ejercicio y su resolución, tanto en pseudocódigo como en la implementación. Una vez creíamos que teníamos una versión definitiva nos encontrábamos con nuevos problemas, y cuando resolvíamos estos con otros que no teníamos antes, y así sucesivamente… Finalmente tuvimos que plantear un procedimiento sencillo y eficiente, que resolviera un número de casos de manera rápida, y crear el denominado ‘caso especial’. Este caso especial se detecta antes del combinar con una función auxiliar, y si se trata de un caso especial no tuvimos otro remedio que resolverlo con el algoritmo directo para intentar tener el menor número de errores posibles. A pesar de esto sabemos que aún hay casos especiales que no se detectan y producen errores.

Esperamos que esto no influya mucho en la nota, si hubiéramos tenido más tiempo lo hubiéramos intentado solucionar, pero habíamos dedicado muchos días a realizar el trabajo y no podíamos continuar. Esperamos que este trabajo sea valorado y no se valore solamente el resultado.

Además, nos gustaría añadir, sin ánimo de atacar ni menospreciar a ningún compañero, que hay problemas más sencillos que otros y ello supone una ventaja para aquellos compañeros a los que les haya tocado por azar.

Esperamos que esto pueda ser solucionado más adelante.

Un saludo.