DOCUMENTACIÓN PRÁCTICA DIVIDE Y VENCERÁS – EJERCICIO 10

Alberto Gálvez Gálvez 49448576W Subgrupo 2.3
Jaime Sánchez Sánchez 48754213D Subgrupo 2.3

Universidad de Murcia, Ingeniería en Informática, Análisis y Estructuras de Datos II
2020-2021

ÍNDICE

Pseudocódigo de la solución

```
function esSubcadena(cadena : string; subcadenasValidas : tabla[0,2] de string): boolean
    return cadena = subcadenasValidas[0]
        || cadena = subcadenasValidas[1]
        || cadena = subcadenasValidas[2];


function solucionCasoBase(cadena : string; indInicio : integer; subcadenasValidas : tabla[0,2] de string): lista de integer
    var
        solucion : lista de integer;
    begin
        if(esSubcadena(trocear(cadena, indInicio, 3)), subcadenasValidas)
        && esSubcadena(trocear(cadena, indInicio + 3, 3), subcadenasValidas)))
            insertar(solucion, indInicio);
    return solucion;
    end.


function solucionDirecta(cadena : string; indIzq, indDer : integer; subcadenasValidas : tabla[0,2] de string): lista de integer
    var
        solucion : lista de integer;
        i : integer;
    begin
        i := indIzq;
        while(i + 5 <= indDer)
            begin
                solucion := juntar(solucion, solucionCasoBase(cadena, i, subcadenasValidas));
                i++;
            end;
        return solucion;
    end.


function combinar(cadena : string; indCorte : integer; var solucionIzq : lista de integer; solucionDer : lista de integer;
subcadenasValidas : tabla[0,2] de string): lista de integer
    var
        solucion : lista de integer;
        i : integer;
    begin
        solucion := juntar(solucionIzq, solucionDer);
        for i := 0 to 4 do
            solucion := juntar(solucion, solucionCasoBase(cadena, indCorte - i, subcadenasValidas));
        return solucion;
    end.


function dividir(indIzq, indDer : integer): integer
    begin
        return (indDer+indIzq)/2 + 1;
    end.


function DyV(cadena : string; indIzq, indDer : integer; subcadenasValidas : tabla[0,2] de string): lista de integer
    var
        indCorte : integer;
        resultadoIzq, resultadoDer : lista de integer;
    begin
        if((indDer - indIzq + 1) >= 12)
            begin
                indCorte := dividir(indIzq, indDer)
                resultadoIzq := DyV(cadena, indIzq, indCorte, subcadenasValidas);
                resultadoDer := DyV(cadena, indCorte + 1, indDer, subcadenasValidas);
                return combinar(cadena, indCorte, solucionIzq, solucionDer, subcadenasValidas);
            end;
        else
            return solucionDirecta(cadena, indIzq, indDer, subcadenasValidas);
    end.
```

Justificaciones del diseño

El algoritmo consta de varias funciones esenciales en un ejercicio resuelto por la técnica de Divide y Vencerás.

El método principal (*DyV*) hace el análisis de casos para saber si debemos dividir el problema en dos subproblemas mediante el cálculo del punto medio de la cadena original (en caso de la longitud de la cadena ser par, una de las subcadenas tendrá una longitud en una unidad mayor que la otra), resolver los subproblemas y combinar las soluciones de los subproblemas; o resolver por un método directo si nos encontramos con un caso lo suficientemente pequeño. El método directo (*solucionDirecta*) consiste en una forma iterativa de resolver el problema.

En cuanto a la elección del caso base, tomamos como tal aquellas cadenas de longitud menor que doce, ya que a partir de una cadena de longitud doce obtenemos como mínimo una cadena de longitud seis en cada mitad de la cadena original, mientras que con longitudes menores que doce, las cadenas de los subproblemas tienen una longitud menor que seis.

Otro método esencial es el que nos permite combinar las soluciones de los dos subproblemas (*combina*), que no solo junta las soluciones, sino que también busca en las fronteras donde se han realizado las divisiones de la cadena original para saber si ahí hay soluciones. La búsqueda en esta frontera siempre comprueba cinco subcadenas en total, que serían las que no son comprobables al dividir el problema original.

Con respecto a la forma de guardar las soluciones es algo más trivial pues hay muchas maneras de hacerlo. Nosotros optamos por usar una lista enlazada, sin embargo, esta decisión da lugar a cuestiones que hay que tener en cuenta (esto se detalla en los resultados del estudio experimental). Casi todos los métodos mencionados devuelven una lista, que soluciones a problemas. Estas listas serían usadas por el método *combina* para combinar las soluciones de los subproblemas y, además, añadir las posibles soluciones de las fronteras.

En resumen, los métodos implementados usan la técnica de *Divide y Vencerás* de la manera correcta. Un subproblema únicamente calcula sus propias soluciones y el método usado para resolver ambos subproblemas es exactamente el mismo. Además de los métodos mencionados, encontramos otros métodos auxiliares que tienen tareas como encontrar el punto medio de una cadena o determinar si una subcadena de longitud seis es una solución que deba añadirse.

Análisis algorítmico Divide y Vencerás

El algoritmo está formado por varias funciones.

Función esSubcadena

Se trata de una función que recibe una cadena de longitud 3 y comprueba si esta cadena es igual a algunas de las subcadenas válidas. Dependiendo de la entrada se ejecutará solo el primer OR, el segundo o el tercero. Por tanto, tenemos:

- Mejor caso: solo se ejecuta el primer OR.
  t(n) = 1
- Peor caso: se ejecuta el tercer OR
  t(n) = 3
- Caso promedio: sería la probabilidad de que se pare en cualquiera de los 3 OR.
  t(n) = cte → 3>=cte>=1

Función soluciónCasoBase

Se trata de una función que recibe una cadena de longitud 6 y se encarga de decidir si es solución o no. Para esto se apoya de la función *esSubcadena*.

- Mejor caso: depende del mejor caso de la función *esSubcadena*. En el mejor caso de *esSubcadena* solo se ejecuta el primer OR, lo que quiere decir que devuelve true.
  t(n) = 1+2*esSubcadenaMejorCaso+pif*1+1=1+2+1+1 = 5
- Peor caso: depende del peor caso de la función *esSucadena*. En el peor caso de *esSubcadena* ejecuta el tercer OR, y además devolvería true
  t(n) = 1+2*esSubcadenaPeorCaso+1*1+1=1+2*3+1+1=9
- Caso promedio: depende del caso promedio de la función *esSucadena*.
  t(n) =1+2*esSubcadenaCasoPromedio+pif*1+1 = cte, 9>=cte>=5

Función soluciónDirecta

Se encarga de recorrer una cadena de longitud menor que 12 y añadir las soluciones que hay en ella. Se apoya en la función *soluciónCasoBase*.

- Mejor caso: depende del mejor caso de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta n-5 de(1+soluciónCasoBaseMejorCaso))+1 = 1+6n-30+1 → n
- Peor caso: depende del peor caso de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta n-5 de(1+soluciónCasoBasePeorCaso))+1 = 1+((10n)-(50)+1 → n
- Caso promedio: depende del caso promedio de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta n-5 de(1+soluciónCasoBaseCasoPromedio))+1 = 1+((1+cte)*n)- ((1+cte)*5)+1 → n

Función combina

Se encarga de combinar las soluciones de los dos subproblemas generados con la técnica de DyV.

- Mejor caso: depende del mejor caso de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta 5 de(1+soluciónCasoBaseMejorCaso))+1 = 1+30+1 = 32
- Peor caso: depende del peor caso de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta 5 de(1+soluciónCasoBasePeorCaso)) +1 = 1+5*(10)+1 = 52
- Caso promedio: depende del caso promedio de la función *soluciónCasoBase*
  t(n) = 1+sum(desde 1 hasta 5 de(1+soluciónCasoBaseCasoPromedio))+1 = 1+5*cte+1

Función dividir

Se trata de una función que únicamente debe realizar una operación aritmética para determinar el punto medio entre dos puntos dados.
t(n) = 2

Función DyV

Se trata de la función recursiva que aplica la técnica de Divide y Vencerás. Por tanto, tenemos dos casos posibles:

t(n) = SolucionDirecta, si n < 12
t(n) = 2*t(n/2) + combina + dividir si n >= 12

Dependiendo de qué tiempo analicemos las constantes serán unas u otras. Pero estas constantes no son definitivas para obtener la solución final.

t(n) = n, si n < 12
t(n) = 2*t(n/2) + cte + cte, si n >= 12

Si resolvemos la ecuación:

t(n) – 2*t(n/2) = cte, n =2^k-> k = $\log_2$ n, t'(k) = t(2^k)

t'(k) – 2*t'(k-1) = cte;

(x-2)(x-1) = 0, R1 = 2, m1 = 1, R2 = 1, m2 = 1

t'(k) = c1*2^k + c2 *1^k

t(n) = c1*n +c2 ≈ n

Podemos concluir que el algoritmo va a tener orden *n*, da igual el caso.

Experimentos de validación del algoritmo

Los experimentos de validación han sido muchos y de distinto tipo, siendo algunas cadenas empleadas para la validación generadas manualmente por nosotros, mientras que otras generadas de forma automática.

Inicialmente, comprobamos la validez de nuestro algoritmo mediante diversas técnicas manuales:

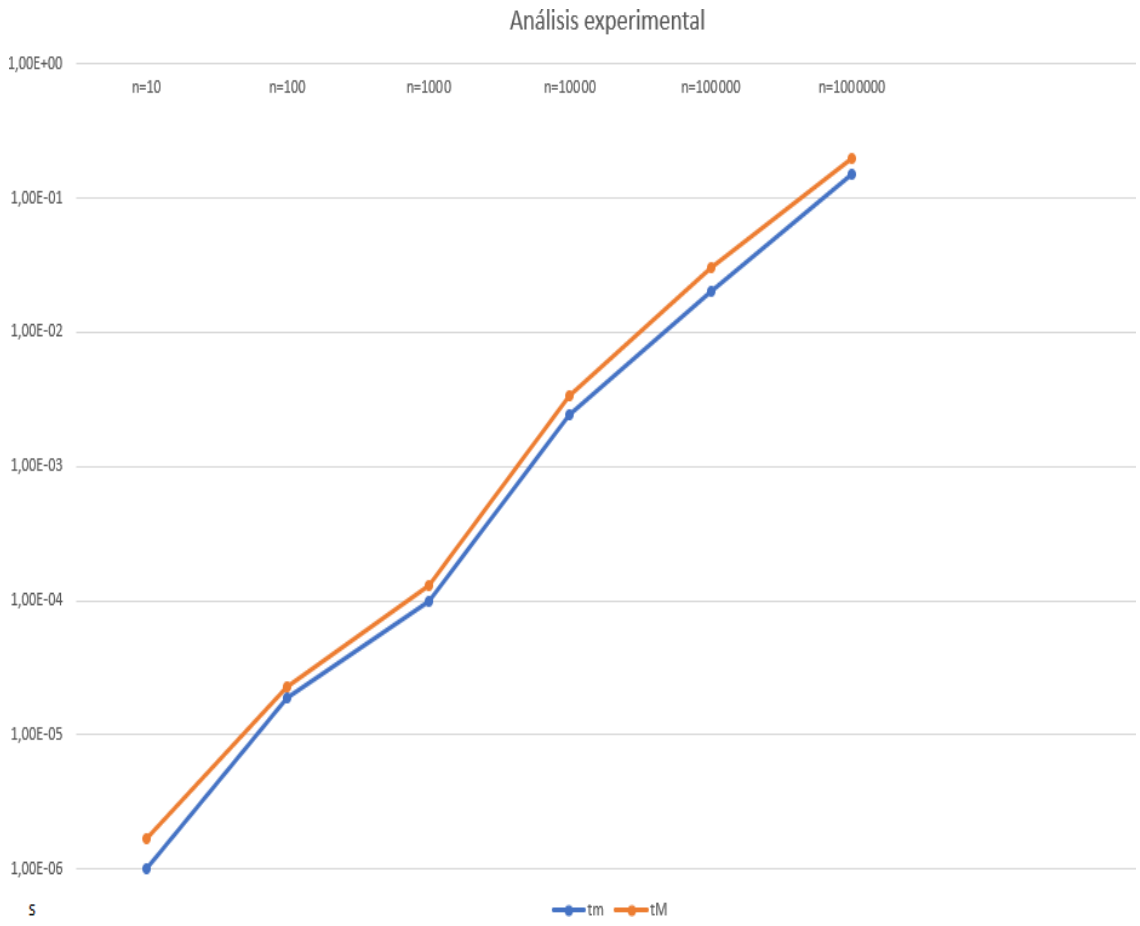
- Imprimir por pantalla de las soluciones que calculaba nuestro algoritmo, comparándolas con las obtenidas a mano por nosotros
- Proporcionar cadenas de igual y menor tamaño que la cota superior del caso base
- Ver los puntos de corte que separan las cadenas mediante impresiones por pantalla
- Poner cadenas de longitud par e impar
- Poner cadenas de gran longitud (n=90000000)
- Probar cadenas que representen el tiempo peor, promedio y mejor

Una vez maduros nuestro generador de pruebas y el método iterativo para generar el problema, nuestros experimentos de validación pasaron a estar automatizados mediante nuestro programa *validador*[1]. Este programa validador hace lo siguiente:

1. Genera numerosos casos de prueba para cada tamaño, cada caso de prueba tiene sus subcadenas válidas (generadas aleatoriamente), así como una cadena que representa el tiempo peor, otra el promedio y otra el mejor
2. Obtiene la solución para cada caso, tanto por el método iterativo como por aquel que emplea la técnica Divide y Vencerás
3. Comprueba que las soluciones sean iguales para ambos métodos
4. Si todas las soluciones para todos los casos de prueba son iguales para ambos métodos y no surgen errores, todo funciona perfectamente

1 Véase: téngase en cuenta que se hacen resuelven casos de tamaños desde muy pequeños, 9 carácteres (1 numSubcadenaL9), hasta tamaños muy grandes 90000 carácteres (10000 numSubcadenaL9). Así como también, para cada uno de estos tamaños, se generan varios casos de prueba y cada uno de ellos tiene un caso peor, promedio y mejor. Por tanto, el proceso validación tarda un tiempo considerable (3 min aprox.) debido al grandísimo volumen de los tests.

Resultados del estudio experimental



La obtención de los tiempos de CPU [2] se ha realizado mediante la función clock de la librería time.h

2 Véase: cabe destacar que debido a que las soluciones se guardan en una lista, al llamar a la función *combina*, se produce una unión de listas. Este proceso de juntar dos listas es proporcional a la suma del tamaño de las listas que se unen, por tanto, cuanta más soluciones haya más va a tardar esta unión. Esto hace que las diferencias entre el tiempo mejor y peor sea aún más pequeña o prácticamente inexistente (al nuestro tiempo mejor ser aquel en el que hay el máximo de soluciones posible, pero viéndose parte de este número menor de instrucciones frente al tiempo peor, contrarrestado por la unión de listas). De todas formas, no hemos tenido esto en cuenta al hacer el análisis asintótico y hemos contado el método que une las dos listas como una sola instrucción, además, si elegimos otro método para guardar las soluciones (como podría ser guardarlas en un fichero), este problema desaparecería.

Contraste entre los resultados empíricos y el estudio teórico realizado

Según el estudio teórico tanto el tiempo mejor, promedio y peor tienen exactamente el mismo orden (n). Las diferencias entre el caso mejor y el peor se reducen a una diferencia totalmente despreciable al tratarse de constantes. Estas diferencias se encuentran en los métodos *combina*, *esSucadena*, *solucionDirecta* y *solucionCasoBase*.

El mejor caso es aquel en el que haya más soluciones, de esta manera en el método *esSubcadena* solo ejecuta el primer *or*. El peor caso es aquel en el que la sentencia *if* del método *soluciónCasoBase* es verdadera y las dos ejecuciones del método *esSubcadena* devuelven true, pero habiendo ejecutado hasta el tercer *or*. El caso promedio se acerca bastante al peor caso, pues por lo general ejecutará hasta el tercer *or* de *esSubcadena* pero la probabilidad de que alguna *subcadena* de longitud tres sea alguna de las *subcadenas* válidas es $3/26^3$, además esta *subcadena* de longitud tres debería estas antes o después de otra *subcadena* que cumpla lo mismo, por lo que en el caso promedio casi nunca tenemos soluciones y el cuerpo del *if* casi nunca se ejecutará. Debido a todo esto, los tiempos suelen salir los mismos porque todos los tiempos del algoritmo tienen un orden exacto *n*.

Conclusiones

Como conclusión, en líneas generales la actividad ha sido bastante de nuestro agrado en comparación a otras de AED I. Sin embargo, la naturaleza del problema a resolver no ha sido demasiado de nuestro gusto.

Esto último debido al hecho de que el problema a resolver no era ni más eficiente ni más lógico resolverlo con la técnica *Divide y Vencerás* frente a una forma *clásica (iterativa).* Por ello, el diseño de un algoritmo con la técnica *DyV* para resolver el problema fue algo que nos costó bastante inicialmente. Sin embargo, una vez diseñado el algoritmo para resolver el problema, la implementación de este fue bastante de nuestro agrado. También, fue de gran agrado la implementación de un generador de pruebas y de un proceso de validación para comprobar la validez de la solución diseñada e implementada por nosotros, debido a la semejanza que puede tener esto con controles de calidad o tests que tengamos que efectuar en el futuro como ingenieros informáticos que desarrollen soluciones de calidad.

Con respecto a la técnica Divide y Vencerás, es una técnica o método de programación que nos ha gustado y sorprendido mucho. Esto, debido por una parte a lo esquemático que es, y por otra, a lo útil que puede ser en algunos tipos de problemas. Creo que el aprender una técnica como esta es una de las diferencias que luego son notorias entre un programador sin capacidad de análisis (por ej: ordenando conjuntos de elementos por fuerza bruta) y otro con capacidad para ello.