MEMORIA PRÁCTICA 1

CONVOCATORIA: ENERO

ALGORITMOS Y ESTRUCTURAS DE DATOS II

CHRISTIAN MATAS CONESA

christian.m.c@um.es

PROFESOR: JOAQUÍN CERVERA LÓPEZ

ENTREGA: 9/01/2024

1. Pseudocódigo y explicación del algoritmo

C: array de char

Función Pequeño(p, q):

Si (q - p + 1 <= 5) entonces

Retornar Verdadero

Sino

Retornar Falso

Fin Si

Fin Función

Función Dividir(p, q):

Retornar (p + q) / 2

Fin Función

Función esCadenaValida(p, q):

cadena := Subcadena(C, p, q)

Ordenar(cadena)

Si (cadena == "abcde") entonces

Retornar Verdadero

Sino

Retornar Falso

Fin Si

Fin Función

Función SolucionDirecta(p, q):

solucion := -1

Si (esCadenaValida(p, q)) entonces

solucion := p + 1

Fin Si

Retornar solucion

Fin Función

Función concatenarSet(s1, s2, s3):

```
    resultado := Vacío
    AgregarElementos(resultado, v1)
    AgregarElementos(resultado, v2)
    AgregarElementos(resultado, v3)
    Retornar resultado
Fin Función
```

Función Combinar(p, m, s1, s2):

```
    SolucionFrontera := Vacío
    Para i en el rango(Max(m - 3, p), m) hacer
        n := SolucionDirecta(i, i + 4)
        Si (n != -1) Entonces
            AgregarElemento(SolucionFrontera, n)
        Fin Si
    Fin Para
    Retornar concatenarSet(s1, s2, SolucionFrontera)
Fin Función
```

Función DyV(p, q):

```
Si (Pequeño(p, q)) entonces
    solucion := Vacío
    n := SolucionDirecta(p, q)
    Si (n != -1) entonces
        AgregarElemento(solucion, n)
    Fin Si
    Retornar solucion
Sino
    m := Dividir(p, q)
    s1 := DyV(p, m)
    s2 := DyV(m + 1, q)
    Retornar Combinar(p, m, s1, s2)
Fin Si
Fin Función
```

Procedimiento Principal:

```
resultado := DyV(0, Longitud(C) - 1)
Imprimir "Número total de subcadenas encontradas: ", Longitud(resultado)
Imprimir "Posiciones de las subcadenas encontradas: "
Para i en resultado hacer
    Imprimir i, " "
Fin Para
Fin Procedimiento
```

La función DyV comprueba si la cadena es lo suficientemente pequeña para devolver la solución directa, sino divide la cadena por la mitad y vuelve a llamar a DyV para cada mitad de la cadena. Cuando ya se consigue que el tamaño sea pequeño se llama a combinar que es la función que nos dará la solución. Lo que hace es extraer los 5 caracteres de la cadena que está analizando, los ordena y los compara con la cadena “abcde” para comprobar si es una solución o no, además también analiza las fronteras al unir dos cadenas por si hubiera alguna solución más.

2. Estudio teórico del tiempo de ejecución y conclusiones acerca de las órdenes

$$t(n) = \begin{cases} g(n) & \text{Si } n \leq 5 \text{ (caso base)} \\ 2 \cdot t(n/2) + f(n) & \text{En otro caso} \end{cases}$$

• t(n): tiempo de ejecución del algoritmo DyV.

• g(n): tiempo de calcular la solución para el caso base, algoritmo directo.

• f(n): tiempo de combinar los resultados.

Vamos a calcular g(n):

g(n) = 1+1+2+1+t(if)·1+1+t(if)·1+1 = 1+1+2+1+1/3125·1+1+1/3125·1+1 = 7.00064

t(if) = 1/5^5 = 1/3125

g(n) = 7.00064 instrucciones de media.

El conteo de instrucciones sería así: empezamos en la función DyV dentro del primer if, ya que es el caso base y se ha cumplido la condición de Pequeño, tiene 1 instrucción, luego entra dentro de SoluciónDirecta, ahí tiene otra instrucción, luego entra en esCadenaValida, ahí tiene 2 instrucciones y un if, si entra en el if como si no entra tiene una instrucción, después vuelve a SolucionDirecta, la probabilidad de este if es la misma que de encontrar una cadena aleatoria de 5 caracteres de la "a" a la "e" ordenados alfabéticamente, hay 5^5 posibles combinaciones de cadenas de 5 caracteres, por lo que la probabilidad de encontrar una ordenada alfabéticamente y entrar en el if es 1/5^5, lo cual es 1/3125. Si entra en el if tendrá una instrucción más, después del if tiene otra instrucción, después vuelve a DyV y tiene otro if y su probabilidad es la misma que la del anterior if, por último, tiene una instrucción más.

El peor caso sería que se cumpliera la condición de los if y el mejor caso sería que no entrara.

Mejor caso g(n): 1+1+2+1+ 1+1 = 7 instrucciones.

Peor caso g(n): 1+1+2+1+t(if)·1+1+t(if)·1+1, como la condición de los if se cumple, quedaría así: 1+1+2+1+1+1+1+1 = 9 instrucciones.

El caso base del algoritmo es cuando n es menor o igual que 5, la función Pequeño(p, q) verifica si la longitud de la subcadena (q – p + 1) es menor o igual a 5. Si es así, devuelve verdadero, indicando que la cadena es lo suficientemente pequeña y se puede resolver directamente. En caso contrario, devuelve falso, indicando que es necesario dividir el problema en subproblemas más pequeños.

Cuando $n \leq 5$, la recursión para y la función DyV devuelve el resultado directo para el caso base. La complejidad de tiempo para el caso base es constante O(1), ya que la resolución es directa y no depende del tamaño de la entrada.

La función SolucionDirecta utiliza la función esCadenaValida(p, q) para verificar si la subcadena es válida. La función esCadenaValida crea una subcadena y la ordena. Si la subcadena ordenada es igual a "abcde", entonces la subcadena original es válida. En caso afirmativo, se establece solucion como p + 1, de lo contrario, solucion permanece como -1.

Cuando la longitud de la subcadena es ≤ 5, el ordenamiento y la verificación de igualdad en esCadenaValida se realizan en un conjunto pequeño y constante de elementos (5 caracteres). El coste de estas operaciones no depende de la longitud total de la cadena de entrada, sino solo de la pequeña constante 5, por eso es O(1).

Vamos a calcular f(n):

Serían las funciones de Dividir() y de Combinar(), la función de Dividir() es muy simple y se trata de una simple instrucción que calcula la mitad de “n”, por eso en la fórmula anterior $2 \cdot t(n/2) + f(n)$ no se tiene en cuenta, además también sería de O(1). La función Combinar() analiza la frontera usando un bucle y reutilizando la función de SolucionDirecta().

Analicemos la función combinar para calcular f(n):

El tiempo de ejecución de esta función depende del rango de valores de “i” en el bucle. La función SolucionDirecta se llama para cada “i”, y dentro de SolucionDirecta se realiza la función esCadenaValida, que ordena una subcadena de longitud constante (5 en este caso).

El bucle itera sobre un rango de valores que depende de “m”, y el tamaño del rango está limitado por Max(m−3,p). Por lo tanto, f(n) se puede expresar en términos del tamaño del rango:

f(n) = O(Tamaño del rango)

La complejidad del bucle en Combinar es lineal en el tamaño del rango. Ahora bien, el tamaño del rango depende de “m” y “p”. Dado que “m” se calcula como Dividir(p,q) en la función DyV, y en cada nivel de la recursión “p” se divide a la mitad, la complejidad de f(n) se puede expresar como O(n).

Peor caso:

En el peor caso, el bucle itera sobre un rango de tamaño 4 Max(m−3,p) hasta m).

La función SolucionDirecta(i, i + 4) se llama para cada “i” en este rango, y dentro de SolucionDirecta se realiza la función esCadenaValida que ordena una subcadena de longitud constante 5.

En conclusión, t(n) es del orden O(n + log n), porque cuando la cadena se ha dividido lo suficiente para cumplir la condición Pequeño (≤ 5), esta se recorre entera con SolucionDirecta(), de ahí la "n" a lo que se le suma "log n" (siempre que se menciona logaritmo me refiero a logaritmo en base 2) que sería el coste de Combinar(), ya que en este algoritmo el coste de dividir es insignificante.

La cadena de caracteres se divide de la siguiente forma:

80 en dos de 40
40 en dos de 20
20 en dos de 10
10 en dos de 5

Esto sería la sucesión 5, 10, 20, 40, 80…, que se puede expresar como: $a_n = 5\ x\ 2^n$

Si despejamos la n, nos quedaría: $n = log_2\left(\frac{a_n}{5}\right)$.

Para una cadena de 10^4 caracteres se necesitarían 11 divisiones.

Para el mejor caso (n = 10^4): $10^4 + log_2(10^4)$ = 10013.29

Para el peor caso (n = 10^6): $10^6 + log_2(10^6)$ = 1000019.93

Dentro del tamaño también influye el número de cadenas que encuentre, cuantas menos cadenas encuentre será más rápido, por lo que dentro del mismo tamaño, si el set resultado es de tamaño 0, será más rápido que si es de gran tamaño.

3. Programación de algoritmo

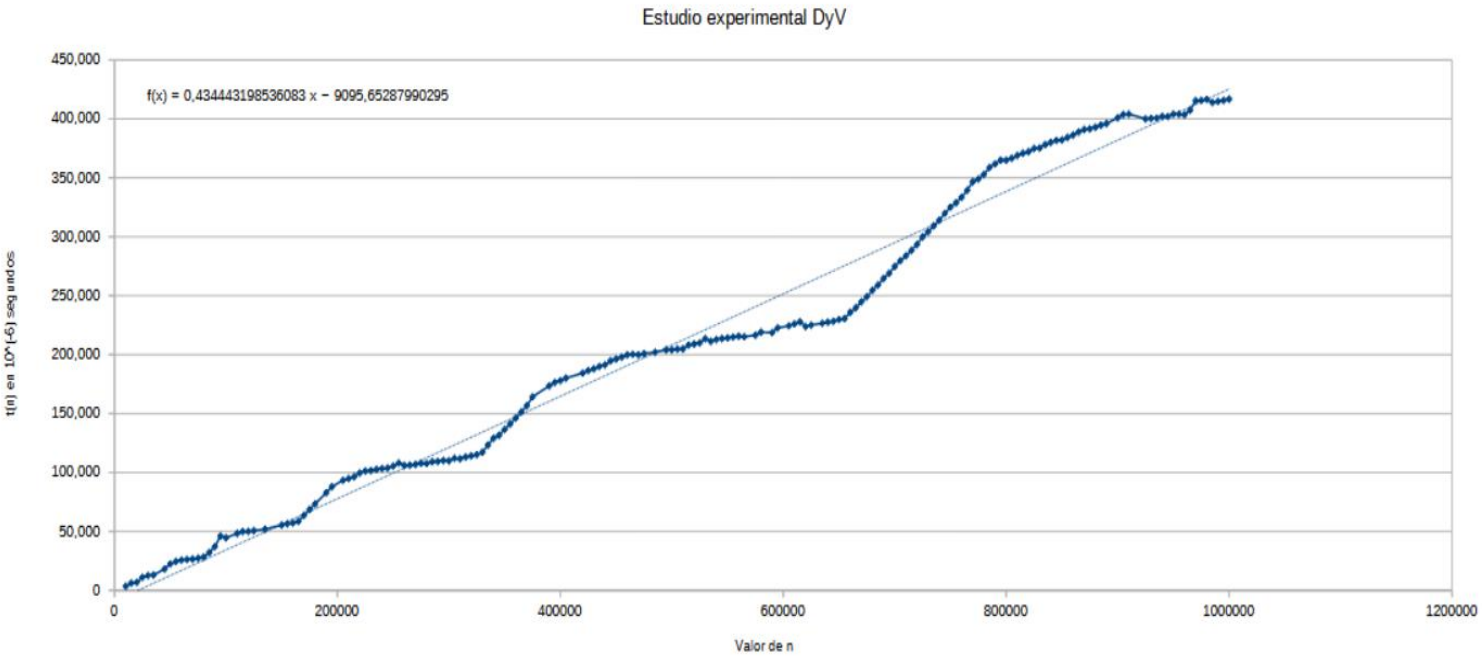
Las funciones del pseudocódigo están traducidas a C++ y son equivalentes. En el código está comentado la labor de cada método y al ejecutar el método “main”, mostrará por pantalla el resultado de un caso de prueba con DyV. Además, he incluido dos DyV más, uno con el que verifico que todas las soluciones encontradas son correctas (muestro por pantalla todas las cadenas que ha detectado como solución y efectivamente se puede comprobar que funciona correctamente. El otro es con el que he hecho el estudio experimental y al ejecutarlo muestra por pantalla el valor de n y t(n) separados por “;”.

4. Validación del algoritmo

Como he comentado anteriormente para la validación del algoritmo he incluido el fichero “DyV-Verificación.cpp”, este incluye una función auxiliar que comprueba que no haya soluciones repetidas (no puede ser posible ya que utilizo sets y estos no pueden contener elementos repetidos, además en el main recorro el set que contiene todas las soluciones con un iterador e imprimo todas las cadenas que ha detectado como resultado para comprobar que todas encajan. El tamaño por defecto está dentro de la función generarCadenaTamañoAleatorio , ahí cambiando la variable entera “tamaño” se puede cambiar el tamaño de la cadena que se va a analizar.

5. Estudio experimental del tiempo de ejecución

Para estudiar el tiempo de ejecución del algoritmo he incluido el fichero "DyV-Experimental.cpp", el cual imprime por pantalla el valor de n y t(n) para 200 valores tales que $10^4 \leq n \leq 10^6$, además también he incluido "estudio.csv" y "estudio.ods" donde se puede ver la salida de forma más clara, además de una gráfica.



Esta gráfica se encuentra en la hoja de cálculo "estudio.ods" y muestra la relación entre el valor de n y el tiempo de ejecución para 200 casos de prueba.

6. Contraste del estudio teórico y el experimental

En el estudio teórico he llegado a la conclusión de que $t(n) \in O(n + log_2 n)$ y por los valores del estudio experimental puedo afirmar que t(n) crece de esta forma. Además si estudio la relación entre el mejor caso y el peor caso del estudio teórico y del estudio experimental, obtengo los siguientes datos:

- t(n) para el peor caso / t(n) para el mejor caso del estudio teórico:
  1000019.93 / 10013.29 = 99.8

- t(n) para el peor caso / t(n) para el mejor caso del estudio experimental:
  416.812 / 3.459 = 120.5

Por lo tanto, como el tiempo de ejecución en el estudio experimental crece de la misma forma que el del estudio teórico y la relación entre el tiempo de ejecución para el mejor y peor caso es similar en ambos estudios, podemos concluir que el análisis teórico no estaba equivocado al predecir el tiempo de ejecución del algoritmo una vez implementado.

7. Conclusiones y valoración

En conclusión, pienso que el algoritmo DyV es una solución adecuada para la búsqueda de subcadenas comunes en cadenas grandes y complejas porque funciona correctamente para diferentes tamaños y combinaciones de cadenas.

Esta práctica es buena para entender mejor cómo se trabaja con algoritmos, y en concreto la técnica de divide y vencerás, una herramienta muy útil, ya que nos permite trabajar con problemas complejos dividiéndolos en subproblemas más pequeños y manejables, y luego combinar las soluciones de estos subproblemas para obtener la solución final.

Por cierto, el curso pasado hice este trabajo con un compañero, pero este curso lo estoy haciendo solo.