Práctica 5 “Solución al problema de encontrar la subsecuencia común mas larga mediante el uso de programación dinámica”

Angel Zait Hernández López
No. Boleta: 2014080682
Análisis de Algoritmos, Grupo: 3CM4

31 de mayo de 2020

Resumen

La programación dinámica es un método para reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas [3]. El problema de subsecuencia común más larga (en inglés, longest common subsequence problem, abreviado LCS problem), se trata de encontrar una subsecuencia más larga que es común en un conjunto de secuencias (Aunque en la mayor parte solamente se toman dos secuencias). [4]

Palabras clave: Programación, dinámica, algoritmo, LCS.

1. Introducción

1.1. LCS

Cómo se mencionó anteriormente, el problema de LCS se trata de encontrar una subsecuencia más larga que es común en un conjunto de secuencias. En casos generales de un número arbitrario de secuencias de entrada, el problema llega a ser NP-hard. Cuando el número de secuencias es constante, el problema se puede resolver en un tiempo polinomial a través de programación dinámica.
Formalmente, dadas dos secuencias $X = \{x_1, ..., x_n\}$ y $Y = \{y_1, ..., y_m\}$, una subsecuencia $Z = \{z_1, ..., z_q\}$ se dice común si es subsecuencia de X y Y simultaneamente. El objetivo de este problema es encontrar una subsecuencia común a dos secuencias y que sea de longitud máxima. Este es un problema con diversas aplicaciones reales, por ejemplo:

- Biología molecular. Las cadenas de ADN se pueden representar como secuencias del alfabeto $\{A, C, G, T\}$. cuando se encuentra una nueva secuencia, interesa encontrar a qué otras secuencias se parece más.
- Comparación de modificaciones realizadas en archivos. Por ejemplo, el comando *diff* de Unix se usa para comparar dos versiones distintas de un mismo archivo. Funciona buscando una subsecuencia de líneas común de longitud máxima.

1.2. Planteamiento del problema

Dadas dos secuencias dadas de las siguientes formas:

$$X = \{x_1, x_2, ..., x_m\} \tag{1}$$

$$Y = \{y_1, y_2, ..., y_n\} \tag{2}$$

Decimos que Z es una subsecuencia común de X e Y sí es subsecuencia de X y subsecuencia de Y. el objetivo es encontrar la subsecuencia de longitud máxima que sea común a las secuencias X e Y.

2. Desarrollo de la práctica

El lenguaje que se utilizó para el desarrollo de esta práctica fue C++, ya que su velocidad es mayor a comparación de otros lenguajes, como lo es Python ó Java. Esto se tomó a consideración por el tipo de problema que se nos presenta, porque puede que exista un problema muy grande y se quiera aproximar a la mejor solución de una manera rápida. Además se optó por realizar programación orientado a objetos para hacer más fácil el proceso del problema, ya que los mismos datos se utilizan varias veces.
Al momento de correr el programa, hay que ingresarle, desde la consola, el nombre del archivo a analizar y el archivo donde se va a guardar el resultado. Este resultado se recomienda que se guarde en un archivo CSV, para poder visualizar los resultados de mejor forma. Para más información, se puede leer el README.txt.

2.1. Algoritmo para la solución del problema LCS

El primer paso para diseñar un algoritmo basado en 'Programación dinámica' consiste en encontrar una formulación del problema que cumpla el principio de optimalidad. Resulta sencillo probar lo siguiente:
Sean $X = \{x_1, ..., x_n\}$ y $Y = \{y_1, ..., y_m\}$ dos cadenas. Si la cadena $Z = \{z_1, ..., z_q\}$ es su subsecuencia común de longitud máxima entonces se cumple:

1. Si $x_n = y_m$ entonces $z_q = x_n = y_m$ y $\{z_1, ..., z_q\}$ es una subsecuencia común de longitud máxima para X y Y ambas sin el último símbolo, es decir, para $\{x_1, ..., x_{n-1}\}$ y $\{y_1, ..., y_{m-1}\}$.

2. Si $x_n \neq y_m$ entonces si se verifica $z_q \neq x_n$ se cumple que Z es la subsecuencia común de longitud máxima para X sin el último símbolo $X = \{x_1, ..., x_{n-1}\}$ y Y.

3. Si $x_n \neq y_m$ entonces, si se verifica $z_1 \neq y_m$ se cumple que Z es la subsecuencia común de longitud máxima para Y sin el último símbolo $Y = \{y_1, ..., y_{m-1}\}$ y X.

A partir de los supuestos anteriores es como se diseña un algoritmo basado en programación dinámica que encuentra una subsecuencia común de longitud máxima en dos cadenas (X y Y). Usando la siguiente ecuación de optimalidad:

$$L(x_i, y_j) = \begin{cases} 0 & si \quad i = 0 \ \ o \ \ j = 0 \\ L(x_{i-1}, y_{j-1}) + 1 & si \quad\quad x_i = y_j \\ max\{L(x_i, y_{j-1}), L(x_{i-1}, y_j)\} & si \quad\quad x_i \neq x_j \end{cases} \tag{3}$$

Viéndolo un poco diferente, esto se puede dividir en dos partes, como son el algoritmo 1 y algoritmo 2. donde se hacen dos matrices, que podríamos ver como tablas, los cuales los llamaremos c y d, donde guardaremos los valores del tamaño de la cadena resultante (Z) y los movimientos para encontrar los caracteres de la subsecuencia final, respectivamente. Estos algoritmos están basados en la teoría dada.

3. Lecturas

3.1. Of mice and men

En esta pequeña lectura, nos hablan sobre el ADN, que la podemos ver como una cadena de caracteres, en los cuales están 4 letras principales:

$$\{A, C, G, T\} \tag{4}$$

Que ayudan a identificar varias cosas dentro del ADN.
En los últimos tiempos, la computación ha ayudado a cumplir objetivos específicos usando la programación dinámica. Estas tareas casi siempre son el encontrar cierta secuencia o patrón de una cadena determinada dentro de los genes.
Esto ayuda demasiado a científicos para poder entender un poco más el cuerpo humano. Pero también menciona que no es del todo fácil, ya que los datos son

exageradamente grandes y que tal vez no cualquier computadora pueda realizar dicha tarea por los recursos que se necesitan. [2]

3.2. Longest common subsequence

En esta parte entra un poco más a fondo sobre la aplicación que tiene dicho algoritmo, al igual que en la lectura anterior, usas las mismas letras, que describen la adenina, guanina, citosina y timina.
Uno de los objetivos de esta aplicación es saber en qué tanto se parecen dos organismos. Una de esas similitudes puede ser que una subcadena esté presente en ambas cadenas a analizar (S_1 y S_2). O bien, al problema que estamos estudiando, existe una tercer cadena (subsecuencia) que es similar en ambos casos y que en ellas es la subsecuencia más larga que existe en ambas cadenas.
Además de mencionar la aplicación de este algoritmo, nos dicen de una forma más formal, la definición de este problema, que ya se habló en la secciones anteriores. [1]

4. Pruebas

Para las pruebas se usaron cadenas especiales relacionadas con contenido ADN como en los ejemplos que se mencionan en las lecturas de la sección 3. Estas cadenas están cargadas en archivos de texto plano (extención TXT), los cuales son: *Ejercicio1.txt*, *Ejercicio2.txt* y *Ejercicio3.txt*. Una con cadenas más grandes que la otra, respectivamente, por lo que mientras más larga es la cadena a comparar, más tardado será el proceso de encontrar el resultado.
Para el primer archivo, se encontró una cadena con una coincidencia de 11.38 %; la segunda cadena es de 57.80 % y el último tiene una coincidencia de 15.55 %. En las imágenes 1 a la 7, se muestra como es que quedan los archivos para cada uno de los archivos. Además, se agrega la tabla 1 dónde se muestran algunos datos de la ejecución para cada uno de los archivos.

Archivo	Tiempo de ejecución	Tamaño de Z	Coincidencia
Ejercicio1.txt	0.029 seg.	46	11.3821 %
Ejercicio2.txt	0.146 seg.	426	57.8019 %
Ejercicio3.txt	0.076 seg.	172	15.5515 %

Tabla 1: Datos de ejecución final para LCS

Como último punto, el programa se ejecutó en dos sistemas operativos distintos

(Windows y Linux), por lo que no se tuvo ningún inconveniente.

5. Conclusiones

Como podemos ver, existen muchas aplicaciones para la programación dinámica, y que hay aplicaciones reales al utilizar este tipo de algoritmos. Notamos nuevamente que este tipo de algoritmos son utilizados para problemas un tanto complejas de resolver para una sola persona. Además, de que puede encontrar rápidamente una solución. Hay que recordar, que encuentra o trata de encontrar la solución más óptima, a comparación de otros algoritmos, como lo son los algoritmos voraces, que se encargan de encontrar la primera solución sin importar que sea la mejor de todas.

Referencias

[1] CORMEN, T. H. *Introduction to Algorithms.* Cambridge, Massachusetts London, England, 2009.

[2] DASGUPTA, S., PAPADIMITRIOU, C., AND VAZIRANI, U. *Algorithms*. 2006.

[3] ECURED. Programación dinámica, https://www.ecured.cu/programación_dinámica.

[4] WIKIPEDIA, L. E. L. Problema de subsecuencia común más larga, https://es.wikipedia.org/wiki/problema_de_subsecuencia_común_más_larga.

```
zait@zait-Lenovo-C260:~/Documentos/AnalisisDeAlgoritmos/AnalisisDeAlgoritmos-ESCOM/Practica05$ time ./main Ejercicio1.txt res01.csv

Programa terminado

real    0m0.029s
user    0m0.012s
sys     0m0.006s
zait@zait-Lenovo-C260:~/Documentos/AnalisisDeAlgoritmos/AnalisisDeAlgoritmos-ESCOM/Practica05$ time ./main Ejercicio2.txt res02.csv

Programa terminado

real    0m0.146s
user    0m0.105s
sys     0m0.013s
zait@zait-Lenovo-C260:~/Documentos/AnalisisDeAlgoritmos/AnalisisDeAlgoritmos-ESCOM/Practica05$ time ./main Ejercicio3.txt res03.csv

Programa terminado

real    0m0.076s
user    0m0.065s
sys     0m0.009s
```

Figura 1: Ejecución del programa

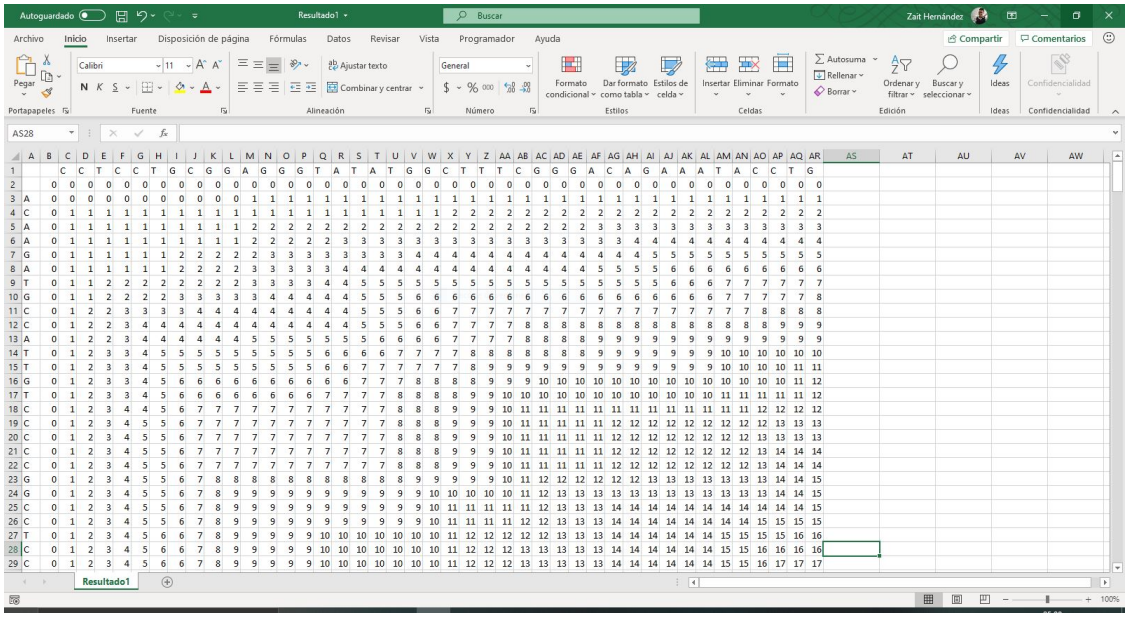

Figura 2: Tabla del ejercicio 1

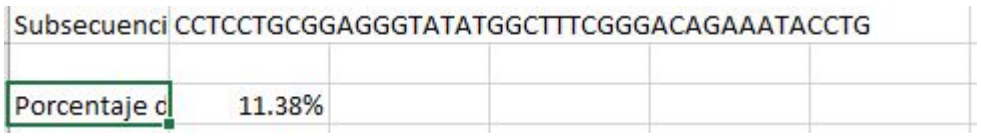

Figura 3: Cadena y coincidencia del ejercicio 1

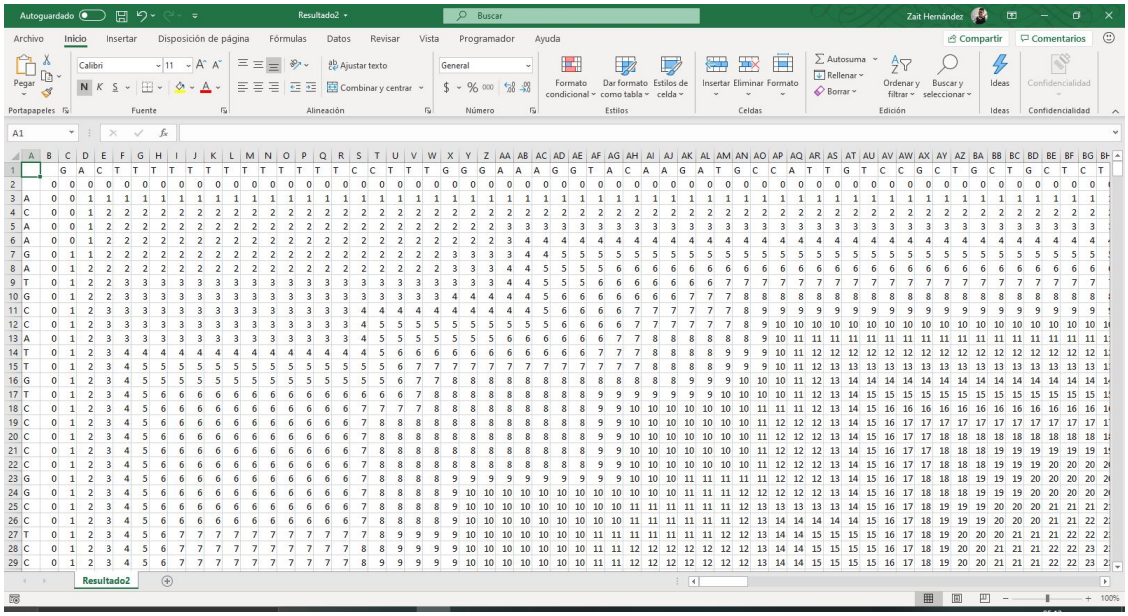

Figura 4: Tabla del ejercicio 2

Figura 5: Cadena y coincidencia del ejercicio 2

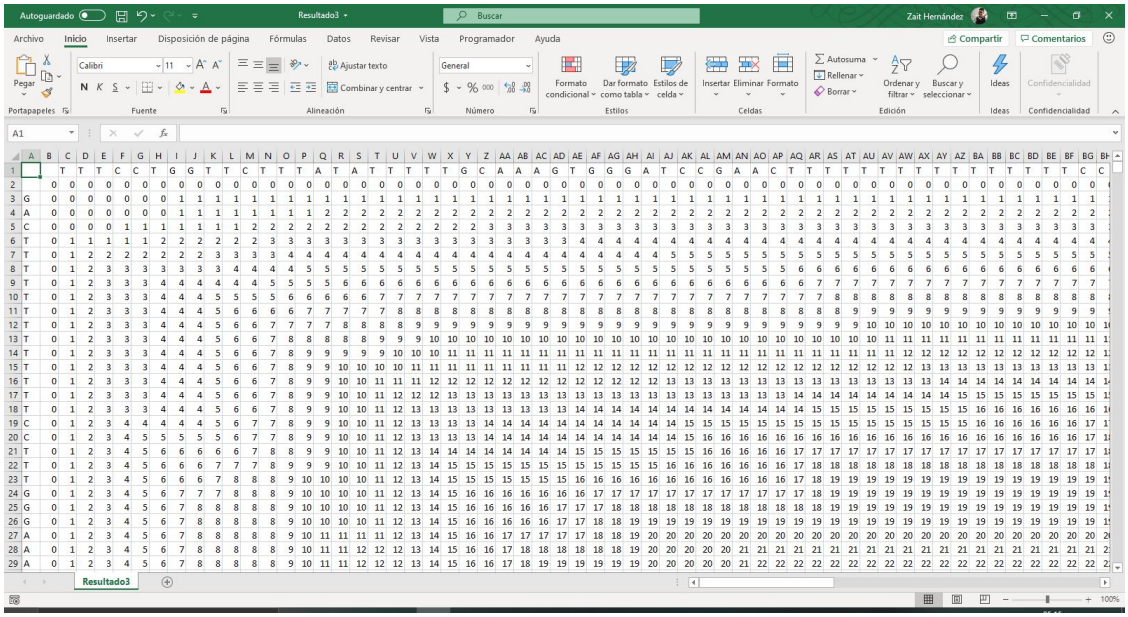

Figura 6: Tabla del ejercicio 3

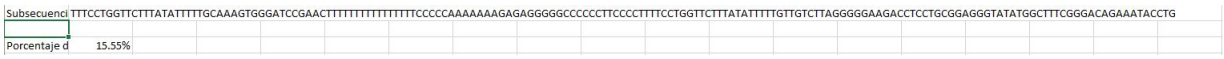

Figura 7: Cadena y coincidencia del ejercicio 3

Algorithm 1: Programación dinámica. LCS-LENGTH

Data: X: Cadena a analizar
Data: Y: Cadena a comparar
Result: c: Matriz donde se guardaran las decisiones en cada iteración.
Result: b: Matriz donde se almacenan los movimientos

```
1  begin
2      let m ← length(X)
3      let n ← length(Y)
4      for i ← 0 to m do
5          c[i, 0] ← 0
6      for j ← 0 to n do
7          c[0, j] ← 0
8      for i ← 1 to m do
9          for j ← 1 to n do
10             if x_i = y_j then
11                 c[i, j] ← c[i − 1, j − 1] + 1
12                 b[i, j] ← " ↖ "
13             else
14                 if c[i − 1, j] ≥ c[i, j − 1] then
15                     c[i, j] ← c[i − 1, j]
16                     b[i, j] ← " ↑ "
17                 else
18                     c[i, j] ← c[i, j − 1]
19                     b[i, j] ← " ← "
20     return c y b
21 end
```

Algorithm 2: Programación dinámica. PRINT-LCS

```
   Data: b: Matriz donde se almacenan los movimientos
   Data: X: Cadena comparada
   Data: i: Posición de fila
   Data: j: Posición de columna
   Result: Z: Cadena/subsecuencia final
 1 PRINT-LCS(b, X, i, j)
 2 begin
 3     if i = 0 or j = 0 then
 4         return
 5     if b[i, j] = " ↖ " then
 6         PRINT-LCS(b, X, i − 1, j − 1)
 7         Z.add(X_i) // Agrega el caracter hasta el final de la cadena
 8     else
 9         if b[i, j] = " ↑ " then
10             PRINT-LCS(b, X, i − 1, j)
11         else
12             PRINT-LCS(b, X, i, j − 1)
13 end
```