Universidad de Valladolid

Programación Dinámica

Subsecuencia Común Más Larga

Sergio García Prado

Seguimiento del trabajo en:
https://github.com/garciparedes/Longest-Common-Subsequence

November 9, 2015

1 Introducción

1.1 Definición del problema

El problema analizado que se pretende resolver consiste en lo siguiente:

Primeramente describiremos lo que es una secuencia para despues explicar qué es una subsecuencia, ya que es una de las ideas fundamentales que habrá que tener claro para resolver el problema y que no debemos confundir con subcadena.

Secuencia: Es una colección ordenada de elementos en la cual la repetición está permitida.

Subsecuencia: Es una secuencia que se obtiene a partir de otra secuencia de igual o mayor tamaño mediante la supresión de algunos elementos sin cambiar el orden de los elementos restantes. Por ejemplo, la secuencia A, D, F es una subsecuencia de A, B, C, D, E, F.
No debe confundirse con el término subcadena, que además impone la restricción de que los elementos tienen que ser contiguos. Un ejemplo de subcadena es B, C, D.

Ahora que ya tenemos clara la definición de subsecuencia modelizaremos el problema a resolver: Dadas dos secuencias de carácteres, nuestro objetivo será encontrar la subsecuencia común de mayor longitud entre ambas. La longitud cada una de las cadenas puede ser distinta.

1.2 Aplicaciones

Este algoritmo tiene una gran cantidad de aplicaciones. Uno de los sectores donde su uso está más extendido es en el de la informática: se usa en software de control de versiones como **git** o el comando **diff** de linux, que muestra las diferencias entre ficheros. También se utiliza en el sector de la bioinformática (Aplicación de tecnología de computadores a la gestión y análisis de datos biológicos.) para **secuenciación de ADN**.

2 Programación Dinámica

2.1 Definición

La programación dinámica es un patrón de diseño de algoritmos basado en la división del problema base que se pretende resolver, en subproblemas de menor tamaño y complejidad que solo se resolverán una única vez, ya que se almacenará la solución de cada uno de ellos para luego reutilizarlo en el caso de que fuera necesario. Al proceso de almacenar las soluciones de los subproblemas se le conoce con el nombre de "memoization". Para que un problema pueda ser resuelto mediante programación dinámica este tiene que cumplir dos propiedades: Solapamiento de Problemas y Subestructura Óptima.

2.2 Solapamiento de Problemas

Se dice que un problema tiene esta propiedad si este se puede subdividir en problemas de menor tamaño cuyos resultados se pueden reutilizar para resolver sucesivos subproblemas. El ejemplo más claro de esta propiedad es la Sucesión de Fibonacci que se define como:

$$F(n) = \begin{cases} n, & \text{if } n \leq 1 \\ F(n-1) + F(n-2), & \text{otherwise} \end{cases}$$

Gráficamente esto se puede representar como un arbol binario, en el cual como vemos para calcular un nodo, tenemos que recurrir a los resultados de sus dos hijos, y así sucesivamente hasta llegar a F(0) o F(1):

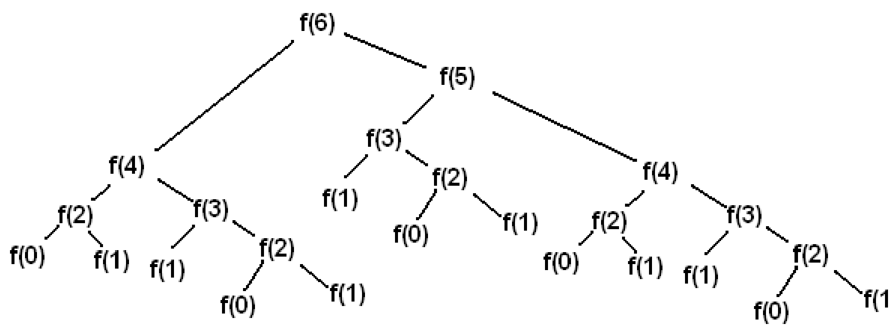

Figure 1: Sucesión de Fibonacci

2.3 Subestructura Optima

La segunda propiedad que debe cumplir el problema es la de subestructura óptima, que consiste en lo siguiente: La solución al problema contiene (o depende) de las soluciones optimas a sus subproblemas. Esta es una propiedad que la programación dinámica comparte con los algoritmos voraces.

3 Solución

3.1 Enfoques Disponibles

Para resolver el problema de encontrar la subsecuencia común más larga entre N secuencias existen distintos enfoques:

Primeramente podríamos pensar en un enfoque por fuerza bruta, es decir, comparando todas las subsecuencias posibles dadas dos listas. Esto tendría un crecimiento asintótico de $O(2^{n_1}\sum_{i=2}^{N} n_i)$ que corresponde al coste de obtener todas las subsecuencias de la primera secuencia ($O(2^{n_1})$) y compararlo con todas las demás (sumatorio de todas las restantes), es decir, de compararlas todas con todas.

Otra solución es utilizar un enfoque dinámico, ya que como demostraremos posteriormente, nuestro problema posee las características para ser resuelto de esta manera. Con este enfoque obtenemos un crecimiento asintótico de $O(N \prod_{i=1}^{N} n_i)$ que corresponde al coste de generar la matriz que almacena las longitudes de subsecuencias comunes.

Notese que para valores muy grandes de N el problema se transforma en uno de tipo NP-hard.

3.2 Subestructura Óptima

Veamos como encontrar la subsecuencia común más larga se puede resolver por programación dinámica. Abordaremos el problema referido a dos secuencias, pero este es extrapolable a cualquier número de secuencias fácilmente incrementando el tamaño de la entrada de nuestro algoritmo.
Supongamos dos secuencias X[0...m-1] e Y[0...n-1] de tamaños m y n respectivamente que a modo de ejemplo supondremos que son abcde y aert.
También que tenemos una matriz de L[0...m-1][0...n-1] de tamaño m x n en la cual iremos almacenando los resultados de cada subproblema

3.2.1 Programación Dinámica

En el caso de que X[m-1] sea igual a Y[n-1] (las últimas posiciones de cada secuencia), rellenaremos la matriz L con el valor de las secuencias sin esos últimos elementos + 1 (el elemento común encontrado):

$$L(X[0..m-1], Y[0..n-1]) = 1 + L(X[0..m-2], Y[0..n-2])$$

Ejemplo:

$$L(abcde, aert) = 1 + L(abcd, aer)$$

Por contra, si los dos últimos elementos no fueran iguales, es decir, si X[m-1] fuese distinto de Y[n-1] tendríamos que obtener el máximo de eliminar el último carácter de la primera secuencia y de la segunda, es decir:

$$L(X[0..m-1], Y[0..n-1]) = max(L(X[0..m-2], Y[0..n-1]), L(X[0..m-1], Y[0..n-2])$$

Ejemplo:

$$L(abcd, aer) = max(L(abc, aer), L(abcd, ae))$$

La formulación completa del problema es la siguiente:

$$LCS(X_i, Y_j) = \begin{cases} 0 & \text{if } i = 0 \vee j = 0 \\ LCS(X_{i-1}, Y_{j-1}) + 1 & \text{if } x_i = y_j \\ max(LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)) & \text{if } x_i \neq y_j \end{cases}$$

Para rellenar la matriz L aplicaremos la función $LCS(X_i, Y_j)$ a cada una de las entradas L_{ij}.

3.2.2 Solapamiento de problemas

Como se puede apreciar, cada subproblema se utiliza para obtener el resultado de los problemas de nivel superior. Por tanto podemos almacenar el resultado para solo calcularle una única vez al igual que en el caso de la Sucesión de Fibonacci..

3.2.3 Obtención de los resultados

Para obtener la longitud de la subsecuencia común más larga tan solo tendremos que obtener el valor de L_{mn}. Este valor se encuentra en esa posición dado que es la última entrada de la matriz que se rellena, es decir, el corresponde a las secuencias completas.

Para obtener la subsecuencia común más larga el proceso es algo más complicado. A grandes rasgos consiste en empezar en la entrada L_{mn} e ir decrementando los valores de i y j añadiendo a la secuencia el elemento correspondiente a dicha posición en el caso de que $X_i = Y_j$ y en caso contrario decrementar decrementar i o j según sea mayor X_i o Y_j hasta que $i = 0$ o $j = 0$. La secuencia resultante será la subsecuencia común más larga.

Este algoritmo se puede adaptar facilmentepara encontrar todas las subsecuencias comunes más largas en caso de que esta no fuera única simplemente recorriendo todos los caminos posibles (ya que este puede no ser único) que cumplan las condiciones anteriormente descritas.

3.2.4 Ejemplo de ejecución

Sean $X = abcde, Y = aert$ por lo que $m = 5, n = 4$.

Lo primero que haremos es aplicar el algoritmo para generar la matriz L.
Nota: Para mejorar la eficiencia del algoritmo aumenta en 1 el tamaño de la matriz para añadir 0 en las entradas L_{ij} tales que $i = 0$ o $j = 0$, lo que reduce el número de comprobaciones respecto del límite.

	∅	a	b	c	d	e
∅						
a						
e						
r						
t						

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0					
e	0					
r	0					
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1				
e	0					
r	0					
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1			
e	0					
r	0					
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1				
r	0					
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0					
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0					

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

Una vez rellenada la matriz ya conocemos la longitud de la subsecuencia común más larga, que, como dijimos antes se aloja en L_{mn}, es decir, L_{65} ya que en nuestro caso hemos hecho más grande la matriz (sino sería L_{54}).

Para obtener la subsecuencia común más larga utilizaremos el algoritmo expuesto anteriormente:

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

	∅	a	b	c	d	e
∅	0	0	0	0	0	0
a	0	1	1	1	1	1
e	0	1	1	1	1	2
r	0	1	1	1	1	2
t	0	1	1	1	1	2

Los resultados de la ejecución para las secuencias *abcde* y *aert* son los siguientes; $LCS = ae$ y su longitud 2

3.3 PseudoCódigo

Algorithm 1 lcs

```
function LCS(X, Y, m, n))
    X : secuencia de m elementos [0...m-1]
    Y : secuencia de n elementos [0...n-1]
    m : longitud de X
    n : longitud de Y

    L ← array[0...m][0...n]

    for i ← 0, i < m + 1, i + + do
        for j ← 0, i < n + 1, j + + do
            if i = 0 ∨ j = 0 then
                L[i][j] ← 0
            else if X[i − 1] = Y[j − 1] then
                L[i][j] ← L[i − 1][j − 1] + 1
            else
                L[i][j] ← max(L[i − 1][j], L[i][j − 1])
            end if
        end for
    end for

    index ← L[m][n]
    LCS ← array[0...index − 1]

    i ← m
    j ← n
    while i > 0 ∧ j > 0 do
        if X[i − 1] = Y[j − 1] then
            LCS[index − 1] ← X[i − 1]
            index ← index − 1
            j ← j − 1
            j ← j − 1
        else if L[i − 1][j] > L[i][j − 1] then
            i ← i − 1
        else
            j ← j − 1
        end if
    end while

    return LCS
end function
```

4 Referencia Bibliográfica

- https://en.wikipedia.org/wiki/Sequence
- https://en.wikipedia.org/wiki/Subsequence
- https://en.wikipedia.org/wiki/Bioinformatics
- https://en.wikipedia.org/wiki/Dynamic_programming
- https://www.cs.berkeley.edu/~vazirani/algorithms/chap6.pdf
- https://en.wikibooks.org/wiki/Algorithms/Dynamic_Programming
- http://faculty.ksu.edu.sa/Alsalih/CSC%20529/5.2_DynamicProgramming.pdf
- http://www.geeksforgeeks.org/dynamic-programming-set-4-longest-common-subsequence/