ugr | Universidad de Granada

ALGORÍTMICA

Curso Académico 2021/2022

Práctica 4

PRÁCTICA 4: PROGRAMACIÓN DINÁMICA

Grupo: Buganvilla

Autores: Manuel Vicente Bolaños Quesada, Pablo Gálvez Ortigosa, Carlos García Jiménez

Índice

1. Introducción

Una forma razonable y comúnmente empleada de resolver un problema es definir o caracterizar su solución en términos de las soluciones de subproblemas del mismo.

En muchos casos, dado un **problema de tamaño** n solo puede obtenerse una caracterización efectiva de su solución en términos de la solución de los **subproblemas de tamaño** $n - 1$. En estos casos, la técnica conocida como **Programación Dinámica (PD)** proporciona algoritmos bastante eficientes.

Es una técnica que se ha utilizado poco a lo largo de los años, debido a sus **enormes requerimientos de memoria**. Sin embargo, hoy día, con el auge de la secuenciación de ADN (Proyecto Genoma Humano), la Programación Dinámica ha vuelto a cobrar una gran importancia.

Aplicaremos esta técnica al problema de encontrar la **mayor subsecuencia común a dos secuencias**, y mostraremos su funcionamiento con dos ejemplos sencillos.

2. Subsecuencia común más larga: Algoritmo

Sabemos de las clases de teoría que este problema verifica las cuatro propiedades que debe tener un problema para ser resuelto utilizando programación dinámica:

- **Naturaleza n-etápica**: La subsecuencia se va construyendo paso a paso, añadiendo en cada etapa una letra más.
- **Verifica el Principio de Optimalidad de Bellman:** pues en cada etapa se calcula el máximo de las etapas anteriores.
- **Plantea una recurrencia** para formar la matriz que permitirá obtener el resultado.
- **Calcula la solución** a partir de la matriz que se ha construido.

En concreto, este algoritmo va construyendo una matriz de tamaño $(n+1) \times (m+1)$, donde n y m son las longitudes de las cadenas. Para ello, en primer lugar, la primera fila y la primera columna se rellenan con ceros. A partir de ahí, cada celda estará compuesta por un número y una flecha o puntero. Para rellenar las demás casillas, si las letras de su fila y su columna coinciden se escogen (el número de la diagonal izquierda)$+1$, y una flecha que apunte hacia la diagonal izquierda.

En caso de que no coincidan, se escoge el máximo entre el elemento de la izquierda y el de encima y se pone la flecha apuntando a ese elemento, en caso de que sean iguales, da igual a cuál de ellos apunte.

Una vez construida la matriz, nos situamos en la celda inferior derecha y siguiendo las flechas se construye la subsecuencia: se van añadiendo a la subsecuencia las celdas con flecha diagonal hasta llegar al comienzo de la matriz.

Esta es la implementación de la parte del algoritmo que calcula la matriz:

```
for (int i=1 ; i<matrix.size(); i++)
    for (int j=1; j<matrix[i].size(); j++) {

        if (cadena1[i] == cadena2[j]) {
            matrix[i][j].value = 1 + matrix[i-1][j-1].value ;
            matrix[i][j].arrow = Arrow::LeftUp ;
        }

        else {

            if (matrix[i][j-1].value >= matrix[i-1][j].value) {
                matrix[i][j].value = matrix[i][j-1].value ;
                matrix[i][j].arrow = Arrow::Left ;
            }
            else {
                matrix[i][j].value = matrix[i-1][j].value ;
                matrix[i][j].arrow = Arrow::Up ;
            }
        }
    }
```

y esta, la de la parte del algoritmo que, teniendo la matriz, calcula la subsecuencia deseada:

```
        while (!stop) {

            switch (element_to_check.arrow) {

                case (Arrow::LeftUp):
                    result += cadena1[row_index] ;
                    row_index -- ;
                    col_index -- ;
                    stop = element_to_check.value == 1 ;
                    break;

                case (Arrow::Left):
                    col_index -- ;
                    break;

                case (Arrow::Up):
                    row_index -- ;
                    break ;
            }
        }
```

A continuación, estudiemos la **eficiencia teórica**:

En primer lugar, el algoritmo de fuerza bruta, que calcula todas las subsecuencias de una secuencia y comprueba si dichas subsecuencias son subsecuencias de la otra secuencia, al haber 2^n y 2^m subsecuencias totales, la eficiencia teórica de este algoritmo es de $\mathbf{O(m \cdot 2^n)}$.

Veamos por otro lado la eficiencia del algoritmo explicado anteriormente: el algoritmo consiste en recorrer una matriz de orden $(n+1) \times (m+1)$, por tanto, la eficiencia teórica será $\mathbf{O(n \cdot m)}$.

3. Subsecuencia común más larga: Dos ejemplos

Estudiemos el siguiente problema: dos hermanos fueron separados al nacer y mediante un programa de televisión se han enterado que podrían ser hermanos. Ante esto, los dos están de acuerdo en hacerse un test de ADN para verificar si realmente son hermanos.

Debemos encontrar el % de similitud que existe entre estos posibles hermanos.

Lo haremos para dos parejas de cadenas diferentes:

1. Primera pareja:

   a) "abbcdefabcdxzyccd"

   b) "abbcdeafbcdzxyccd"

   Hemos ejecutado nuestro algoritmo con esta entrada, y hemos obtenido la siguiente matriz:

		a	b	b	c	d	e	a	f	b	c	d	z	x	y	c	c	d
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1
b	0	$\uparrow$ 1	$\nwarrow$ 2	$\nwarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\nwarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2	$\leftarrow$ 2
b	0	$\uparrow$ 1	$\nwarrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\nwarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3	$\leftarrow$ 3
c	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\nwarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 4	$\nwarrow$ 4	$\nwarrow$ 4	$\leftarrow$ 4
d	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\nwarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\nwarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\leftarrow$ 5	$\nwarrow$ 5
e	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\nwarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6	$\leftarrow$ 6
f	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\leftarrow$ 6	$\nwarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7
a	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\nwarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7	$\leftarrow$ 7
b	0	$\uparrow$ 1	$\nwarrow$ 2	$\nwarrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\nwarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8	$\leftarrow$ 8
c	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\nwarrow$ 9	$\leftarrow$ 9	$\leftarrow$ 9	$\leftarrow$ 9	$\leftarrow$ 9	$\nwarrow$ 9	$\nwarrow$ 9	$\leftarrow$ 9
d	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\nwarrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\uparrow$ 9	$\nwarrow$ 10	$\leftarrow$ 10	$\leftarrow$ 10	$\leftarrow$ 10	$\leftarrow$ 10	$\leftarrow$ 10	$\nwarrow$ 10
x	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\uparrow$ 9	$\uparrow$ 10	$\leftarrow$ 10	$\nwarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11
z	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\uparrow$ 9	$\uparrow$ 10	$\nwarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11	$\leftarrow$ 11
y	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\uparrow$ 9	$\uparrow$ 10	$\uparrow$ 11	$\leftarrow$ 11	$\nwarrow$ 12	$\leftarrow$ 12	$\leftarrow$ 12	$\leftarrow$ 12
c	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\nwarrow$ 9	$\uparrow$ 10	$\uparrow$ 11	$\leftarrow$ 11	$\uparrow$ 12	$\nwarrow$ 13	$\nwarrow$ 13	$\leftarrow$ 13
c	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\nwarrow$ 9	$\uparrow$ 10	$\uparrow$ 11	$\leftarrow$ 11	$\uparrow$ 12	$\nwarrow$ 13	$\nwarrow$ 14	$\leftarrow$ 14
d	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 4	$\nwarrow$ 5	$\uparrow$ 6	$\uparrow$ 7	$\leftarrow$ 7	$\uparrow$ 8	$\uparrow$ 9	$\nwarrow$ 10	$\uparrow$ 11	$\leftarrow$ 11	$\uparrow$ 12	$\uparrow$ 13	$\uparrow$ 14	$\nwarrow$ 15

   Y hemos obtenido la **subsecuencia más larga:** "abbcdeabcdzyccd".

   Podemos ahora calcular el % de similitud de ambas cadena con un cálculo sencillo:

$$\frac{15}{17} \cdot 100\,\% = 88{,}2352941\,\%$$

2. Segunda pareja:

   a) “01011100010001010101001000 1001001001”

   b) “11000010010010101000101001001 1010100”

   Para esta pareja de datos, hemos obtenido una matriz parecida a la anterior, y hemos obtenido que la **mayor subsecuencia más larga** es: “11000000101010010001001001 00”.

   Similarmente, calculamos el porcentaje de similitud entre ambas cadenas:

$$\frac{30}{36} \cdot 100\,\% = 83{,}3333333\,\%$$

En ambos casos, **las secuencias obtenidas no son únicas**. En los casos en los que al calcular el máximo entre el número de la izquierda y el número de arriba, los números eran iguales, nuestro algoritmo ha escrito una flecha a la izquierda, pero también podría escribirse una flecha hacia arriba, obteniendo así distintas subsecuencias (aunque la longitud será la misma).

Por ejemplo, en el caso 1, una subsecuencia que también sería válida sería “abbcdefbcdzyccd”.

4. Conclusión

La programación dinámica puede ser muy útil en determinados problemas, en los que tenga sentido **construir la solución a partir de las soluciones de etapas anteriores**.

En el problema que hemos tratado, hemos aprendido a comprobar que un problema verifica las condiciones que debe cumplir un problema de PD y hemos conseguido mejorar la eficiencia desde una **exponencial hasta una eficiencia cuadrática**, demostrando de esta forma la gran importancia de la Programación Dinámica.

Sin embargo, también tenemos que tener en cuenta los **requerimientos de memoria** que exige esta técnica. En este caso, por ejemplo, por cada letra que añadamos a las secuencias, tendremos que añadir una fila y/o columna a la matriz.

A día de hoy, el auge de la genética y los aumentos en la memoria de los dispositivos siguen otorgando cada vez más y más interés a esta técnica.

5. Materiales utilizados

- Google Spreadsheets para organizar información
- Visual Studio Code para programar el algoritmo
- Overleaf (LaTeX)