DP (continuación): LCS y Edit Distance

Facundo Galán

LCS: Introducción

Se tienen dos strings S y T, y debemos determinar el longest common subsequence (subsecuencia común más larga).

Ejemplo:

S = CADSA

T = CBASAS

LCS = CASA

¿Se puede resolver con DP?

Tenemos que pensar si cumple con las propiedades necesarias:

- Optimal Substructure
- Overlapping Subproblems

Optimal substructure

Tenemos que encontrar la función recursiva que resuelva el problema

Podríamos pensarlo con una función LCS(n, m), a la cual llamamos con los tamaños de los string S y T respectivamente.

LCS(x, 0) = 0 para todo x

LCS(0, x) = 0 para todo x

Con esto tenemos suficiente para LCS(1, 1)

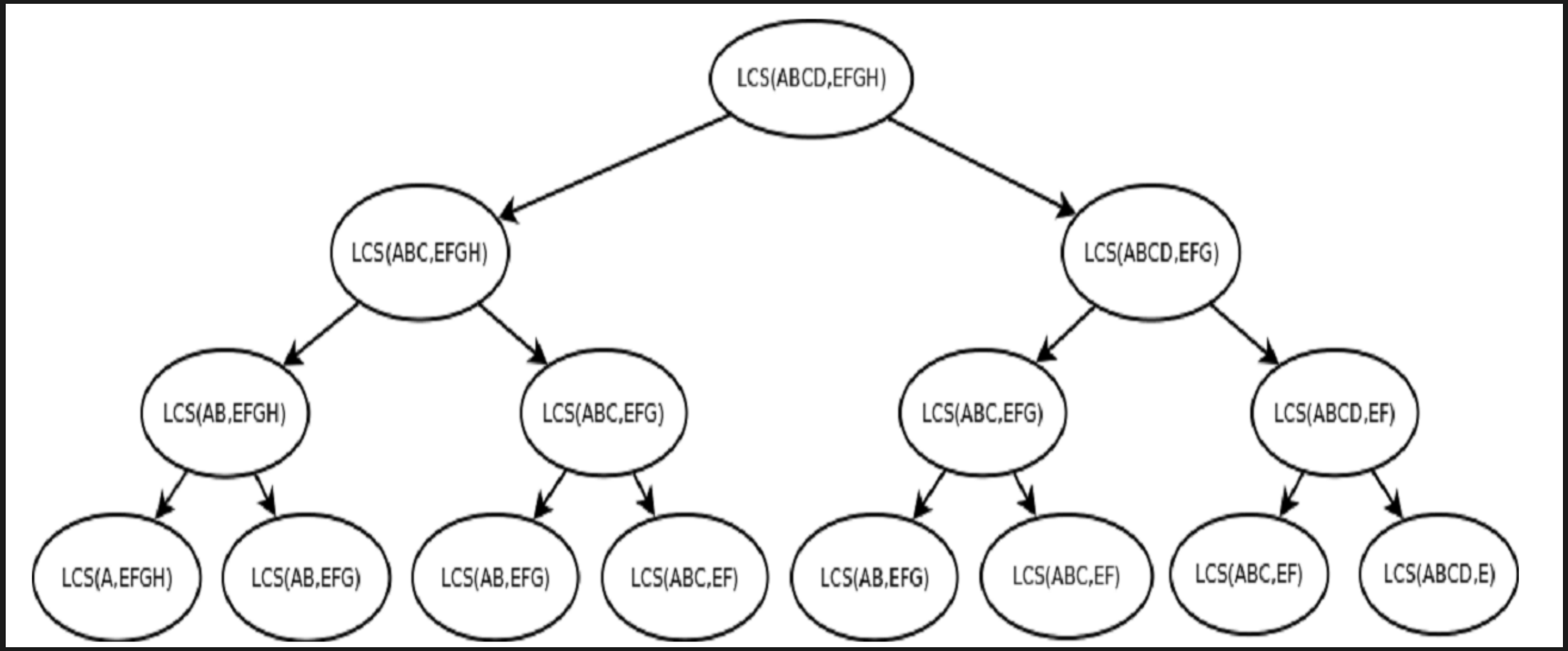
Optimal substructure

Tenemos que encontrar la función recursiva que resuelva el problema

Podríamos pensarlo con una función LCS(n, m), a la cual llamamos con los tamaños de los string S y T respectivamente.

$$
LCS(i, j) = \begin{cases} 0 & i == 0 \ \ v \ \ j == 0 \\ LCS(i-1, j-1) + 1 & S[i] == T[j] \\ \max(LCS(i-1, j), LCS(i, j-1)) & S[i] \ != T[j] \end{cases}
$$

Overlapping subproblems



Estructura

El valor de la función dependerá de dos parámetros númericos.

La estructura que podemos usar es una matriz de dos dimensiones:

	null	C	A	R
null				
R				
O				
C				
A				

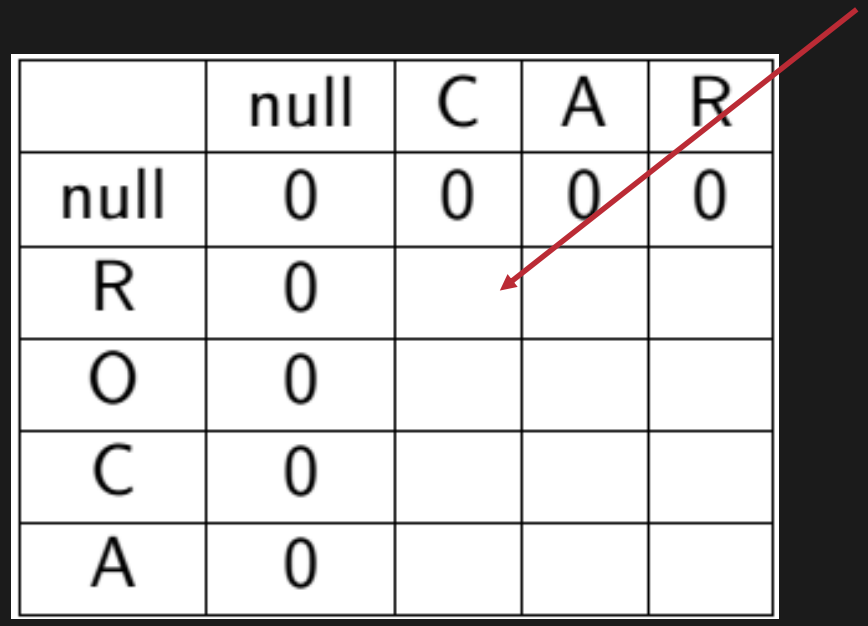
Procedimiento: inicialización

Completamos los valores de LCS que ya conocemos.

	null	C	A	R
null	0	0	0	0
R	0			
O	0			
C	0			
A	0			

Procedimiento: calculando los LCS

Con los valores actuales, cuál valor nuevo podemos calcular?



	null	C	A	R
null	0	0	0	0
R	0			
O	0			
C	0			
A	0			

Procedimiento: calculando los LCS (1)

Seguimos completando de la misma forma.

	null	C	A	R
null	0	0	0	0
R	0	0		
O	0			
C	0			
A	0			

Procedimiento: calculando los LCS (1)

	null	C	A	R
null	0	0	0	0
R	0	0	0	1
O	0	0	0	1
C	0	1	1	1
A	0	1	2	2

Solución Bottom-Up

```
int LCS (string s, string t) {
      int n = s.length() + 1, m = t.length() + 1, memo[n][m], i, j;
      for (i=0; i<n; i++) memo[i][0] = 0;
      for (j=0; j<m; j++) memo[0][j] = 0;
      for (i=1; i<n; i++) for (j=1; j<m; j++) {
            if (s[i-1] == t[j-1]) memo[i][j] = memo[i-1][j-1] + 1;
            else memo[i][j] = max(memo[i-1][j], memo[i][j-1]);
      }
      return memo[n-1][m-1];
}
```

Edit Distance: Introducción

Se tienen dos strings S y T, y debemos determinar la mínima cantidad de operaciones necesarias para transformar el primer string en el segundo. Las operaciones pueden tener costo, y son: reemplazo, inserción, y eliminación.

Ejemplo:

Insert: 3, Replace: 4, Delete: 2

S = CADSA

T = CBADSAS

ED = 6 (insertar B en la posición 1 y S en el final)

¿Se puede resolver con DP?

Tenemos que pensar si cumple con las propiedades necesarias:

- Optimal Substructure
- Overlapping Subproblems

Optimal substructure

Tenemos que encontrar la función recursiva que resuelva el problema. Podríamos pensarlo como una función ED(n, m), a la cual llamamos con los tamaños de los string S y T respectivamente.

Si no nos quedan más letras del primer string ED(0, x), sólo nos queda insertar las x faltantes del segundo.

Si no nos quedan más letras del segundo string ED(x, 0), sólo nos queda eliminar las x faltantes del primero.

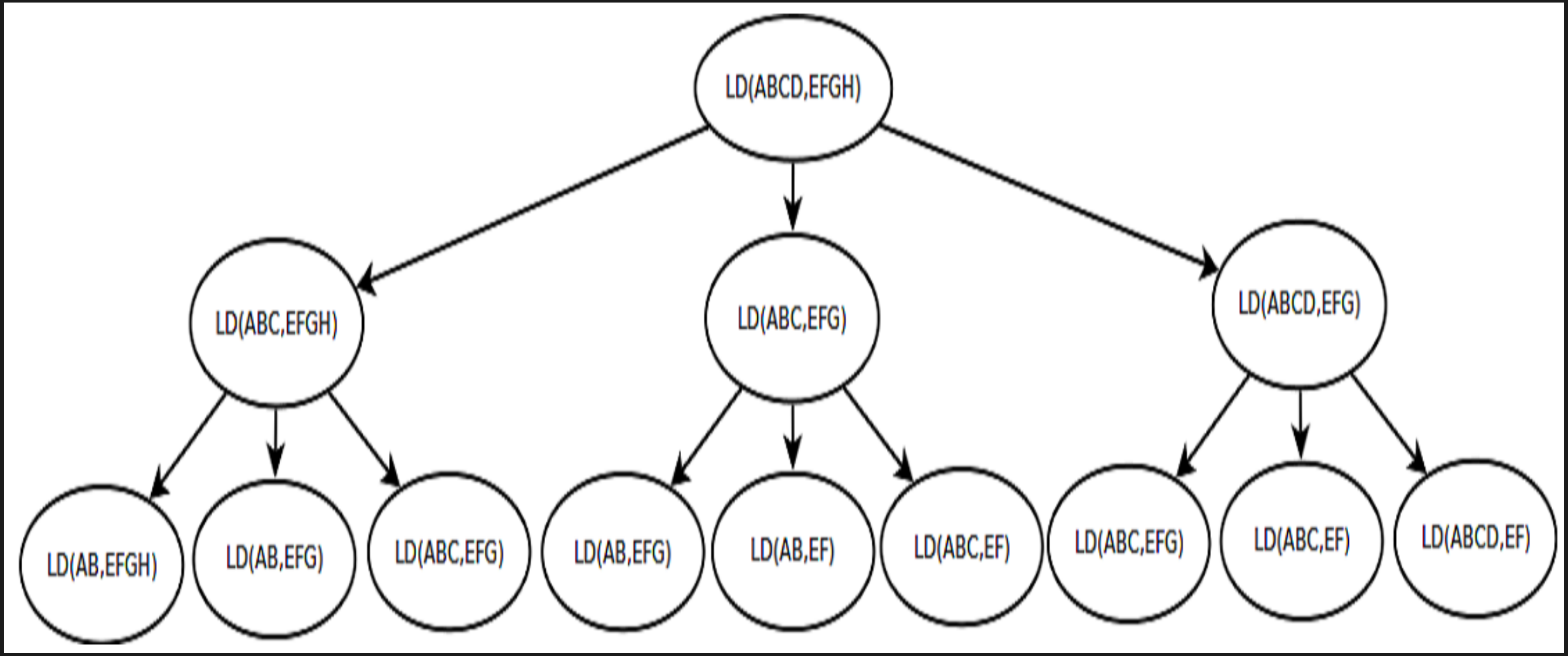
En caso de tener letras en ambos strings ED(x, y), tenemos que verificar cuál de las tres operaciones nos conviene realizar.

Optimal substructure

Tenemos que encontrar la función recursiva que resuelva el problema. Podríamos pensarlo como una función ED(n, m), a la cual llamamos con los tamaños de los string S y T respectivamente.

$$
lcs(i, j) = \begin{cases} j * cost\_del & i == 0 \\ i * cost\_ins & j == 0 \\ \min(LD(i-1, j) + cost\_del, & i > 0 \ \hat{} \ j > 0 \\ \quad LD(i, j-1) + cost\_ins, & \\ \quad LD(i-1, j-1) + ((S[i] == T[j]) \ ? \ 0 : cost\_rep)) & \end{cases}
$$

Overlapping subproblems



Estructura

El valor de la función dependerá de dos parámetros númericos.

La estructura que podemos usar es una matriz de dos dimensiones:

	0	C	A	R
0				
R				
O				
C				
A				

Procedimiento: inicialización

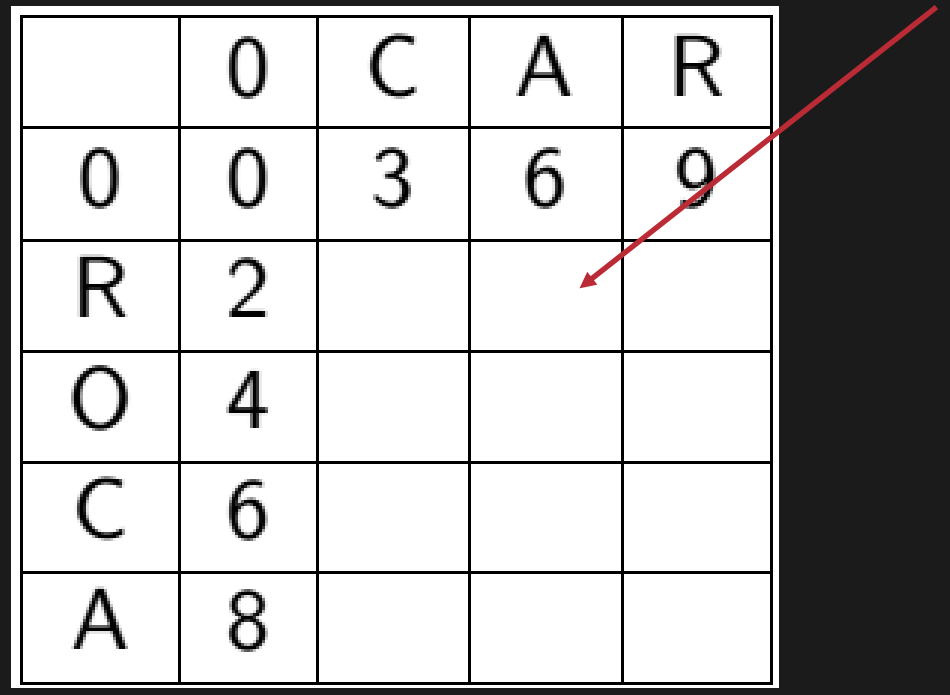
Completamos los valores de LCS que ya conocemos.

	0	C	A	R
0	0	3	6	9
R	2			
O	4			
C	6			
A	8			

Procedimiento: calculando los LCS

Con los valores actuales, cuál valor nuevo podemos calcular?

	0	C	A	R
0	0	3	6	9
R	2			
O	4			
C	6			
A	8			



Procedimiento: calculando los LCS (1)

Seguimos completando de la misma forma.

	0	C	A	R
0	0	3	6	9
R	2	4		
O	4			
C	6			
A	8			

Procedimiento: calculando los LCS (1)

	0	C	A	R
0	0	3	6	9
R	2	4	7	6
O	4	6	8	8
C	6	4	7	10
A	8	6	4	7

Solución Bottom-Up

```
int ED (string s, string t, int cost_del, int cost_ins, int cost_rep) {
        int n = s.length() + 1, m = t.length() + 1, memo[n][m], i, j;
        for (i=0; i<n; i++) memo[i][0] = i * cost_del;
        for (j=0; j<m; j++) memo[0][j] = j * cost_ins;
        for (i=1; i<n; i++) for (j=1; j<m; j++) {
                memo[i][j] = matrix[i-1][j-1] + (S[i-1] != T[j-1])?cost_rep:0;
                memo[i][j] = min(memo[i][j], memo[i-1][j] + cost_del);
                memo[i][j] = min(memo[i][j], memo[i][j-1] + cost_ins);
        }
        return memo[n-1][m-1];
}
```