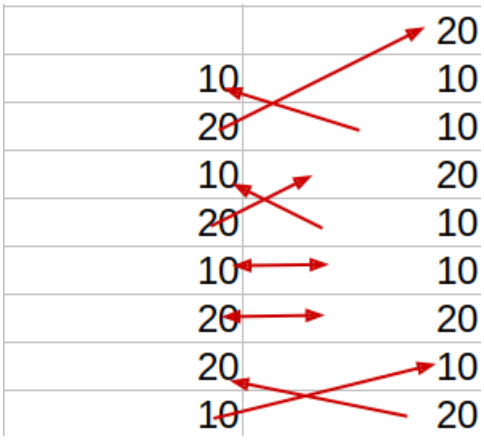
Álvaro de la Rosa Zarzuelo

Francisco Javier Dujo Villacorta

Reparto del trabajo:
El trabajo lo hemos realizado de forma conjunta en las horas de laboratorio y repartiendo de forma equitativa por cuenta propia, por lo que la estimación del trabajo desarrollado por cada uno es de un 50%.

Explicación realización practica:
Al comienzo de la realización de la practica planteamos el siguiente algoritmo:

```
funcion torres(torre1, torre2)
        while torre1 no vacia & torre2 no vacia
                if torre1[0] = torre2[0] then
                        añadir numero a la solucion
                        eliminar numero de ambas torres
                else
                        buscar torre1[0] en torre2
                        buscar torre2[0] en torre1
                        if encuentra then
                                eliminar menor cantidad de numeros hasta que torre1[0] = torre2[0]
                                añadir numero a la solucion
                                eliminar numero de ambas torres
        end
devolver solucion
```



En cuanto probamos con muestras más grandes nos dimos cuenta de que esta idea tiene un problema: no siempre encuentra la operación más optima si no la primera posible.

Nos pusimos a pensar e investigar y al final llegamos a la conclusión de que la mejor solución posible sería programación dinámica. Entonces primero pensamos en como sería la matriz de resultados intermedios para este problema. Nuestra primera idea fue una matriz de 0`s que marcase con un 1 al encontrar dos números iguales.

	3	5	7	4	8	6	7	8	2
1	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0
5	0	1	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0
7	0	0	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1	0	0
8	0	0	0	0	1	0	0	1	0

Pero esta idea era un problema para posteriormente extraer un resultado concreto. Investigando un poco más por internet encontramos una solución a un problema parecido al nuestro, solo que en este caso comparaba cadenas de caracteres en vez de torres de números [1], y lo intentamos comprender y adaptar a nuestras necesidades personales.
Este programa al encontrar dos números iguales incrementa en 1 tanto esa posición en la matriz como las siguientes a la derecha y hacia abajo. De esta forma cuando tenemos que seleccionar una solución, comenzamos en la esquina inferior derecha y si el número es mayor que los números a su izquierda y encima será una pieza de la torre solución. Si no se da este caso se avanzará en la búsqueda por el camino con el numero máximo entre la posición izquierda y superior. En caso de ser iguales se ha tomado la decisión de seguir hacia arriba (cambiar esto solo nos daría una solución diferente pero igual de válida). El algoritmo termina al llegar a los limites de la matriz.
Ejemplo:

	3	5	7	4	8	6	7	8	2
1	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1
4	1	1	1	2	2	2	2	2	2
5	1	2	2	2	2	2	2	2	2
6	1	2	2	2	2	3	3	3	3
7	1	2	3	3	3	3	4	4	4
7	1	2	3	3	3	3	4	4	4
8	1	2	3	3	4	4	4	5	5

Análisis de la eficiencia del programa desarrollado:

n = n.º de piezas en la torre 1 // m = n.º de piezas en la torre 2
El tiempo que tarda en crear la tabla es: O(n) + O(m) + O((n-1)*(m-1)) = O(n*m)
El tiempo que tarda en extraer una solución es: O(max(n,m))

Por lo tanto la eficiencia total del programa es: O(n*m) + O(max(n,m)) = O(n*m)

Referencias:

[1] La subsecuencia común más larga de LCS con DP. [Online]
Available: https://programmerclick.com/article/20711646348/