Técnicas de diseño de algoritmos
Programación dinámica
Ejercicios (Bloque 2): con pautas de resolución

Luis Javier Rodríguez Fuentes
Amparo Varona Fernández

Departamento de Electricidad y Electrónica
Facultad de Ciencia y Tecnología, UPV/EHU
luisjavier.rodriguez@ehu.es
amparo.varona@ehu.es

OpenCourseWare 2015
Campus Virtual UPV/EHU

Programación dinámica – Ejercicios (Bloque 2)

(B2.1) Considerese el problema de calcular el coeficiente binomial, definido como sigue:

$$\binom{n}{k} = \begin{cases} 1 & k = 0 \vee k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 0 & \text{en caso contrario} \end{cases}$$

Escribir en lenguaje Python cuatro formas distintas de calcular el coeficiente binomial:

(a) Mediante un algoritmo recursivo según la definición.
(b) Mediante un algoritmo recursivo con memoria, que evite repetir cálculos.
(c) Mediante un algoritmo de programación dinámica que utilice una matriz de resultados intermedios (el conocido como *triángulo de Pascal*).
(d) Mediante un algoritmo de programación dinámica que utilice una única lista de resultados intermedios, que se va actualizando de izquierda a derecha (ya que los coeficientes de orden n sólo dependen de los de orden $n-1$, que se han calculado en la iteración anterior y están almacenados en la lista).

Aplicar los algoritmos desarrollados para calcular $\binom{10}{5}$ y $\binom{20}{10}$.

En Python, una matriz es una lista de listas, de manera que cada una de las listas (filas) que forman dicha matriz puede tener un número distinto de columnas. Esto permite almacenar de manera sencilla los resultados previos en los métodos (b) y (c). En lo que respecta al método (d), la definición del coeficiente binomial permite usar una única lista de longitud k que empieza teniendo los resultados de la iteración anterior ($n-1$) y se va actualizando de izquierda a derecha con los resultados de la iteración n.

Programación dinámica – Ejercicios (Bloque 2)

(B2.2) Sean u y v dos cadenas de caracteres. Se desea transformar u en u aplicando el menor número posible de operaciones de edición (borrados, inserciones y sustituciones). Escribir en lenguaje Python un algoritmo de programación dinámica que obtenga el número mínimo de operaciones de edición (y cuáles son esas operaciones) para transformar u en v. Calcular la complejidad temporal del algoritmo en función de las longitudes de u y v.

Si $u = abcac$ y $v = babba$, la conversión óptima implica una inserción (la primera b de v), una sustitución (la primera c de u por la tercera b de v) y un borrado (la última c de u). Para desarrollar el algoritmo deberá minimizarse la suma de operaciones de edición sobre una matriz E que guarda el número óptimo de operaciones de edición para convertir la subcadena de $u[1:i]$ en la subcadena de $v[1:j]$. En cada paso deberá elegirse la operación menos costosa:

$$E[i][j] = \text{mín}(E[i-1][j-1] + s(u[i], v[j]), E[i-1][j] + 1, E[i][j-1] + 1)$$

donde $s(x, y)$ es el coste de sustituir x por y (que será 1 si $x \neq y$ y 0 si $x = y$). Para recuperar el camino de edición (esto es, las operaciones realizadas), deberá definirse una matriz auxiliar C que guarde memoria del paso óptimo en cada casilla de E.