Práctica programación dinámica: subsecuencia común más larga

Maitena Tellaetxe Abete

2 de noviembre de 2015

Dadas dos secuencias $V = v_1 v_2 \dots v_n$ y $W = w_1 w_2 \dots w_m$, calcular la subsecuencia común más larga (LCS).

(a) **Caracterizar la estructura de una solución óptima.**

El objetivo es hallar la LCS entre W y V.
Para cada par de subsecuencias SW_i y SV_j de W y V respectivamente tal que

$$SW_i = W_1 \dots W_i$$

$$SV_j = V_1 \dots V_j$$

$$i = 1 \dots m, \quad j = 1 \dots n$$

Si $W_i = V_j$, se añade dicho elemento a la LCS calculada hasta el momento:

$$LCS(SW_i, SV_j) = LCS(SW_{i-1}, SV_{j-1}) \cup W_i$$

y por tanto la longitud del LCS aumenta en un elemento.
Si dichos elementos no son iguales, la subsecuencia no se extiende y seguirá siendo la óptima o más larga calculada hasta el momento:

$$LCS(SW_i, SV_j) = \text{ópt}\{LCS(SW_{i-1}, SV_j), LCS(SW_i, SV_{j-1})\}$$

(b) **Definir recursivamente el valor de una solución óptima.**

Sea LLCS la longitud del LCS óptimo:

Si $V = \emptyset$ o $W = \emptyset$:

$$i = 1 \dots m, \quad j = 1 \dots n$$

$$LLCS(W_i, 1) = 0$$

$$LLCS(1, V_j) = 0$$

Recursivamente:

$$i = 2 \dots m, \quad j = 2 \dots n$$

$$LLCS(W_i, V_j) = \begin{cases} LLCS(W_{i-1}, V_{j-1}) + 1 & \text{si } W_i = V_j \\ \max\{LLCS(W_{i-1}, V_j), LLCS(W_i, V_{j-1})\} & \text{si } W_i \neq V_j \end{cases}$$

$LLCS(m, n)$ será la longitud de uno de los LCS óptimos si hay más de uno, y del óptimo si es único.

(c) **Cálculo del valor de una solución óptima.**

El código para resolver este problema desarrollado en R se muestra a continuación:

```
LCS <- function(seq1, seq2) {
  m <- nchar(seq1)
  n <- nchar(seq2)
  cost_mat <- matrix(0, nrow = m+1, ncol = n+1)
  cost_mat[1,1] <- 0
  # Llenado de la matriz dinamica y calculo de la longitud
  # del LCS optimo
  # Llenado de la primera fila y columna
  # Fila
  for (i in 2:(m+1)) {
    cost_mat[i,1] <- 0
  }
  # Columna
  for (j in 2:(n+1)) {
    cost_mat[1,j] <- 0
  }
  # Llenado del resto de la matriz
  for (i in 2:(m+1)) {
    for(j in 2:(n+1)) {
      if (areEqual(seq1, seq2, i-1, j-1)) {
        cost_mat[i,j] <- cost_mat[i-1, j-1] + 1
      } else {
        values <- c(cost_mat[i, j-1] ,
                    cost_mat[i-1, j] )
        cost_mat[i,j] <- max(values)
      }
    }
  }
  print (paste("Length of LCS is: ", cost_mat[m+1, n+1],
               sep = ""))
```

```
  # Reconstruccion del LCS
  subsequence <- backtrack(cost_mat, strsplit(seq1, "")[[1]],
                    strsplit(seq2, "")[[1]], i = m, j = n)
  print (paste("The LCS is: ",
      paste(subsequence[2:length(subsequence)],
                                  collapse = ""), sep = ""))
}

# Funciones helper
areEqual <- function(seq1, seq2, i,j) {
  seq1 <- strsplit(seq1, "")[[1]]
  seq2 <- strsplit(seq2, "")[[1]]
  return(seq2[j] == seq1[i])
}
backtrack <- function(cost_mat, seq1, seq2, i, j) {
  if (i==0 || j==0) { #Una de las dos secuencias está vacía
    return ("")
  } else if (seq1[i] == seq2[j]) {
    return (c(backtrack(cost_mat, seq1, seq2, i-1, j-1),
              seq1[i])) #Si tenemos  un match, retrocedemos
    # por la diagonal y guardamos esa letra
  } else {
    if (cost_mat[i+1, j] > cost_mat[i,j+1]) { #En este caso
      # venimos por la izquierda y no guardamos nada
      return (backtrack(cost_mat, seq1, seq2, i, j-1))
    } else { #En este caso venimos por arriba y no guardamos
      return (backtrack(cost_mat, seq1, seq2, i-1, j))
    }
  }
}
```

(d) **Calcular la subsecuencia común más larga para los siguientes pares de secuencias:**
ATCTGAT y TGCATA.

```
seq1 <- "ATCTGAT"
seq2 <- "TGCATA"
LCS(seq1,seq2)

## [1] "Length of LCS is: 4"
## [1] "The LCS is: TCTA"
```

BDCABA y ABCBDAB.

```r
seq1 <- "BDCABA"
seq2 <- "ABCBDAB"
LCS(seq1, seq2)

## [1] "Length of LCS is: 4"
## [1] "The LCS is: BDAB"
```