Längste gemeinsame Teilfolge

Wolfgang Mulzer

Seien $s = s_1 s_2 \dots s_k$ und $t = t_1 t_2 \dots t_\ell$ zwei Zeichenketten. Wir suchen zunächst die Länge der längsten gemeinsamen Teilfolge von s und t. Sei `LLGT` ein zweidimensionales Array vom Typ `int` mit Dimension $(k+1) \times (\ell+1)$. Wir füllen `LLGT` systematisch gemäß der Rekursion aus der Vorlesung auf.

```
// Initialization
for i := 0 to k do
  LLGT[i,0] <- 0
for j := 0 to l do
  LLGT[0,j] <- 0
// Implementing the recursion
for i := 1 to k do
  for j := 1 to l do
    if s[i] = t[j] then
      LLGT[i,j] <- 1 + LLGT[i-1, j-1]
    else
      LLGT[i,j] <- max {LLGT[i-1, j], LLGT[i, j-1]}
```

Das Ergebnis seht in $\texttt{LLGT}[k, \ell]$. Die Laufzeit ist $O(k\ell)$. Nachdem `LLGT` berechnet ist, kann man einen längste gemeinsame Teilfolge finden, indem man einen Weg durch `LLGT` rekonstruiert, der zu einer optimalen Lösung führt (d.h., wir rekonstruieren die Pfeile, die wir in der Vorlesung in der Tabelle vermerkt hatten). Dies geschieht von hinten.

```
  // a stack to store the result
  S <- new Stack
  // start at the end
  (i,j) <- (k,l)
  while i > 0 and  k > 0 do
    if s[i] = t[j] then
      // if the current symbols are equal, we can include them into the
      // sequence
      (i,j) <- (i-1, j-1)
      S.push(s[i])
    else
      // otherwise we need to determine which of the two choices led to the
      // maximum
      if LLGT[i, j-1] > LLGT[i-1, j] then
        (i, j) <- (i, j-1)
      else
        (i, j) <- (i-1, j)
  // now S contains an optimal subsequence
  while not S.isEmpty() do
    output S.pop()
```