# Longest Common Subsequence

*Wolfgang Mulzer*

Let $s = s_1 s_2 \dots s_k$ and $t = t_1 t_2 \dots t_\ell$ be two strings. First, we would like to determine the length of a longest common subsequence of $s$ and $t$. Let `LLCS` be a two-dimensional array of type `int` of size $(k+1) \times (\ell+1)$. We fill `LLCS` systematically according to the recursion from class.

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

The result can be found in $\texttt{LLCS}[k, \ell]$. The running time is $O(k\ell)$. After `LLCS` has been computed, we can find a longest common subsequence, by reconstructing the way through `LLCS` that leads to an optimal solution (i.e., we reconstruct the arrows, that we have drawn into the table in class). This works from bottom to top.

```
// 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 LLCS[i, j − 1] > LLCS[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()
```