Grundzüge DS & Alg (WS14/15)

Lösungsvorschlag zu Aufgabenblatt 10

Aufgabe 1

Betrachtet man die rekursive Formel genauer, so fällt auf, dass man nur auf die vorherigen Indizes i und j zugreift. Die Idee ist nun, schrittweise den Index i zu erhöhen. Dann braucht man sich nur alle vorherigen Werte für j zu merken in zwei Arrays der grösse j.

```
function PLAYOFF(i, j, p)
    int[] prev = new int[j]
    initialisiere prev mit 1
    int[] curr
    for x = 1 to i do
        for y = 1 to j do
            curr[y] = p * prev[y] + (1-p) * curr[y-1]
        end for
        prev = curr
    end for
    return curr[j]
end function
```

Laufzeit: Die innere Schleife hat Laufzeit O(j) $\in$ O(n), da j $\leq$ n. Diese Schleife wird i mal ausgeführt, da i $\leq$ n kommt man auf eine Laufzeit von O(n^2). Zusätzlich muss man am Anfang noch das eine Array initialisieren, was in O(n) ist, daher ist die Laufzeit insgesamt in O(n^2).

Speicherbedarf: Man verwendet zwei Arrays der Grösse j, daher ist der Speicherbedarf insgesamt in O(n).

Aufgabe 2

Wir sagen, dass zwei Zahlen a_i und b_j zusammenpassen, falls $a_i \leq b_j$.

Sei $T(i, j)$ die Länge des längsten Dominanzteilfolgenpaars für die Teilfolgen $a_1, ..., a_i$ und $b_1, ..., b_j$. Sei außerdem $T^*(i, j)$ die Länge eines solchen Dominanzteilfolgenpaares, das a_i und b_j enthält, oder 0, falls a_i und b_j nicht zusammenpassen.

Für $T(i, j)$ gibt es drei mögliche Fälle:

1. a_i und b_j sind beide Teil des längsten Dominanzteilfolgenpaars, dann ist $T(i, j) = T^*(i, j)$
2. a_i kommt nicht vor, dann ist $T(i, j) = T(i - 1, j)$
3. b_j kommt nicht vor, dann ist $T(i, j) = T(i, j - 1)$

Rekursiv lassen sich die Funktionen $T(i, j)$ und $T^*(i, j)$ also wie folgt berechnen:

$$T(i, j) = max\{T^*(i, j), T(i - 1, j), T(i, j - 1)\}$$

$$T^*(i,j) = \begin{cases} T(i-1,j-1)+1, & \text{falls } a_i \text{ und } b_j \text{ zusammenpassen} \\ 0 & \text{sonst} \end{cases}$$

Das lässt sich mit dynamischer Programmierung implementieren:

```
DOMINANZTEILFOLGENPAAR(A, B, n, m)
  T = new array[0..n][0..m], initialisiert mit 0

  for i = 1 to n
      for j = 1 to m
          if A[i] <= B[j] then
              T_star = T[i - 1][j - 1] + 1
          else
              T_star = 0
          T[i][j] = max(T_star, T[i - 1][j], T[i, j - 1])

  k = T[n][m]
```

Gesucht ist aber nicht die maximale Länge k, sondern das Dominanzteilfolgenpaar. Es muss also aus den Werten von $T(i,j)$ rekonstruiert werden:

```
  A' = new array[1..k]
  B' = new array[1..k]
  i = n, j = m
  while i > 0 && j > 0 do
      if T[i][j] == T[i - 1][j - 1] + 1 then
          A'[k] = A[i]
          B'[k] = B[j]
          --k, --i, --j
      else if T[i][j] == T[i - 1][k] then
          --i
      else
          --j
  done

  return (A', B')
```

Für die Berechnung von T wird Laufzeit in $\Theta(nm)$ benötigt, da jeder der nm Wert in $\Theta(1)$ berechnet werden kann. Die Rekonstruktion braucht zusätzlich Laufzeit in $\Theta(n+m)$. Damit ist die Laufzeit insgesamt in $\Theta(nm)$.