Algorithmen zur Sequenzanalyse

Wintersemester 2018/2019 **Übungsblatt 4** Prof. Dr. E. Ohlebusch, Doktoranden

Besprechung am 11.12.2015 Institut für Theoretische Informatik

---

Aufgabe 4.1.

Wir betrachten noch einmal das SUS-Problem aus Aufgabe 3.5. Entwerfen Sie einen linearen Algorithmus, der das SUS-Problem auf folgende Weise löst:

- Durch eine bottom-up Traversierung des lcp-Intervallbaumes.
- Durch eine top-down Traversierung (Breitensuche) des lcp-Intervallbaumes.
- Die Länge des SUS, der an einer Position $\mathsf{SA}[i]$ im String S beginnt, kann durch die Werte $\mathsf{LCP}[i]$ und $\mathsf{LCP}[i+1]$ bestimmt werden. Mithilfe dieser Information kann das SUS-Problem gelöst werden, indem das LCP-Array einmal durchlaufen wird.

Aufgabe 4.2.

1. Entwerfen Sie einen linearen Algorithmus, der den *lcp*-Intervallbaum einmal bottom-up durchläuft und dabei alle lcp-Intervalle aufzählt. Ein lcp-Intervall bestand bislang aus den Komponenten $\langle lcp, lb, rb, childList\rangle$. Statt der Kindliste soll als vierte Komponente hier die Liste der lcp-Indizes (in aufsteigender Reihenfolge) des lcp-Intervalls berechnet werden, d.h. ein lcp-Intervall besteht nun aus den Komponenten $\langle lcp, lb, rb, lcpIndices\rangle$.
2. Erweitern Sie den Algorithmus um ein Array $lastOcc$ der Größe σ, sodass $lastOcc[c]$ den Index speichert, an dem der Buchstabe $c \in \Sigma$ zuletzt in der BWT vorkam. Genauer gesagt soll die for-Schleife der bottom-up Traversierung folgende Invariante haben:
   - Vor jeder Ausführung des Rumpfes der for-Schleife für einen Wert k $(2 \leq k \leq n+1)$ enthält $lastOcc[c]$ $(c \in \Sigma)$ den Index des letzten Vorkommens von c in $\mathsf{BWT}[1..k-1]$ (wobei $lastOcc[c] = 0$ gelten soll, falls c nicht in $\mathsf{BWT}[1..k-1]$ vorkommt).
3. Implementieren Sie nun die Prozedur $process(lastInterval, lastOcc)$, sodass der Gesamtalgorithmus alle supermaximalen Repeats ausgibt.
4. Weisen Sie die Korrektheit des Algorithmus nach.
5. Analysieren Sie die worst-case Laufzeit des Algorithmus.