Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Johannes Krugel

Sommersemester 2009
Übungsblatt 3
15. Mai 2009

Algorithmische Bioinformatik I

Aufgabe 1

Im Folgenden soll die Anwendbarkeit der verschiedenen vorgestellten Algorithmen zur Suche eines Patterns in einem Text untersucht werden.

a) Warum ist der naive Algorithmus für ein sehr kurzes Pattern schneller als der KMP- und der BM-Algorithmus?

b) Die Worstcase-Laufzeit vom BM-Algorithmus mit Bad-Charakter-Regel ist $O(nm)$, die von KMP-Algorithmus ist $O(n+m)$. Warum ist dennoch der BM-Algorithmus in der Praxis deutlich schneller als der KMP-Algorithmus?

c) Warum ist der BM-Algorithmus bei großem Alphabet schneller als bei einem kleinen?

d) Der gesunde Menschenverstand und die Worst-Case-Laufzeit von BM mit Bad-Character-Regel legen nahe, dass der Algorithmus bei längerem Pattern auch mehr Zeit benötigt. Bei der Suche in einem englischen Text mit ungefähr 30.000 Zeichen stellt sich aber heraus, dass die Suche nach ”inter” fünf mal so lange braucht wie die Suche nach ”interactively”. Woran liegt das?

e) Die Suchlaufzeit des BM-Algorithmus wird also kleiner mit zunehmender Länge des Patterns. Wenn die Länge des Patterns gegen unendlich geht, müsste die Suchlaufzeit also immer kleiner werden. Oder?

Aufgabe 2

Ein zyklischer String S ist ein unendlicher String, für den gilt $\exists i : \forall j : s_j = s_{j+i}$. Das kleinste solche i ist die Länge des zyklischen Strings. Gegeben seien zwei zyklische Strings S_1 mit Länge l_1 und S_2 mit Länge l_2, $l_1 \leq l_2$. Geben Sie einen möglichst effizienten Algorithmus an, der herausfindet, ob ein Substring von S_1 der Länge l_1 in S_2 enthalten ist. Was ist die Laufzeit des Algorithmus?

Aufgabe 3

Sei Σ ein endliches Alphabet und sei $* \notin \Sigma$ ein so genanntes *don't-care*-Symbol. Ferner sei $s \in (\Sigma \cup \{*\})^+$ und $t \in \Sigma^+$ mit $|s| = m$ und $|t| = n$ sowie $k \in \mathbb{N}$ eine Konstante, die unabhängig von n und m ist.

Entwerfen Sie einen Algorithmus, der feststellt, ob s ein Teilwort von t matched, d.h., ob es ein $i \in [0 : n-m]$ gibt, so dass für alle $j \in [0 : m-1]$ entweder $s_j = *$ oder $s_j = t_{i+j}$ ist. Wenn das *don't-care*-Symbol dabei genau k-mal in s auftritt, soll der Algorithmus dabei maximal $O(k \cdot n + m)$ Vergleiche ausführen.

Hinweis: Versuchen Sie den Algorithmus von Knuth-Morris-Pratt geeignet zu modifizieren.

Aufgabe 4

Rechts ist der Suffix-Trie für das $t = ababba$ dargestellt. Konstruieren Sie mit Hilfe des in der Vorlesung vorgestellten Online-Algorithmus' schrittweise die Suffix-Tries für

- $t' := t \cdot b = ababbab$,
- $t'' := t' \cdot a = ababbaba$ und
- $t''' := t'' \cdot a = ababbabaa$.

Geben Sie für jedes Wort t', t'' und t''' eine eigene Zeichnung mitsamt der verwendeten und neu konstruierten Suffix-Links an.