Musterlösung
Übungszettel 4
Algorithmen & Datenstrukturen für Bioinformatiker

Adrian Haß

17. Februar 2003

1 k-Cover

Wir können im voraus schon einmal festellen, dass es reicht, zu entscheiden ob sich S_2 aus Teilen von S_1 der Länge l, $k \leq l < 2k$ zusammensetzen lässt, denn größere Stücke lassen sich dann ja wieder in kleinere unterteilen.

Dazu konstruiere man einen Suffixbaum für S_1 in Linearzeit. Dann beginne man S_2 von der Wurzel an zu matchen, solange bis ein Mismatch auftritt oder wir bis zur Tiefe $2k-1$ vorgedrungen sind. Befinden wir uns nun noch oberhalb der k-$2k$-Bande, dann gibt es kein k-Cover. Wenn nicht, dann springen wir (implizit!) k-mal über Suffixlinks und fahren dort mit matching fort. In jedem Schritt behalten wir jedoch die bisher erreichte Tiefe in einer Variable gespeichert und ziehen beim springen k ab. Dies wird wiederholt.

Wir sind nach dem Sprung immer unterhalb der k-Marke. Ist jetzt unser erster Mismatch zu tief, dann springen wir nur zurück zur Wurzel und versuchen dann die k weiter aufzufüllen. Gelingt dies nicht (weil S_2 zuende ist oder wieder zu früh ein Mismatch auftritt), dann gibt es kein k-Cover.

Die Sprünge im Suffixbaum kann man sich am besten anhand der Werte von der Zählvariable vorstellen. Diese erhöht sich sukzessive und springt dann wieder um k zurück.

Ein k-Cover existiert nun dann, wenn sich alle "Spitzen" (d.h. die Werte bevor wir zurückspringen) innerhalb der k-$2k$-Bande befinden.

Wenn wir einem Suffixlink gefolgt sind, dann lassen wir die Grenze zwischen den Teilwörtern sozusagen varibel und geben keinen festen Trennpunkt vor. Wir wissen nur, dass wir für den bisherigen Teil von S_2 ein k-Cover finden könnten.

War jetzt ein Teilwort zu kurz, dann wissen wir, dass wir den Bereich den wir bisher durch ein k-Cover abdecken konnten, maximal um den Rest der Zählvariable ausdehen können um die aktuelle "Spitze" noch ein wenig zu erhöhen. Aber eben nur um den Rest und deshalb muss das letzte Stück bis zur k-Linie jetzt wirklich gefunden werden.

Die Konstruktion des Baumes ging in $O(|S_1|)$ vonstatten und es wurden auch nur $|S_2|$ Vergleiche ausgeführt. Einzig die k-Sprünge bereiten zunächst Kopfschmerzen. Doch davon kann es ja maximal $|S_2|/k$ Stück geben und jeder Sprung ist in $O(k)$, also ergibt sich hierfür auch $O(|S_2|)$. Demnach hat der angegebene Algorithmus lineare Laufzeit.

2 Teilstringfamilie

Die Minimalität ist hier so zu verstehen: S' ist dann minimal, wenn S nicht mehr teilstringfrei ist, sobald man einen beliebigen String aus S' wieder zu S hinzufügt.

Man konstruiert einen gemeinsamen Suffixbaum in Linearzeit indem, beginnend mit dem größten String, immer die nächstkleineren eingefügt werden. Dabei kann folgendes auftreten:

- **neuer String ist in einem Teilstring enthalten,** dann ist dieser String in die Menge S' einzufügen, denn er ist in dem jeweiligen Suffix enthalten.
- **neuer String matcht keinen der bisherigen Suffices,** dann kann er gefahrlos in S verbleiben und wird nach Ukkonen in den bisherigen Baum eingefügt.

Die Menge aller nun im Suffixbaum nicht berücksichtigten String ist nun nach obiger Definition minimal und sie wurde in linearer Zeit ermittelt.