Technische Universität München
Institut für Informatik
Lehrstuhl für Effiziente Algorithmen
Volker Heun

SS 2000
Lösungsblatt 2
25. Mai 2000

Algorithmen der Bioinformatik

Abgabetermin: 23. Mai

Aufgabe 1

Geben Sie den Suffixbaum für das Wort $T = acctagctagct$ an. Ist dieser eindeutig?

Lösungsvorschlag: Siehe Figur 1;

Aufgabe 2

a) Geben Sie einen Algorithmus für das Problem *exact string matching* auf der Basis von Suffixbäumen an, der zu einem Text T und einem Suchwort P die Position *jedes* Vorkommens von P in T ausgibt. Für k Vorkommen von P in T soll die Komplexität des Algorithmus $O(|T| + |P| + k)$ betragen.

Lösungsvorschlag:

Ein geeigneter Algorithmus könnte etwa folgendermaßen aussehen:

(1) Baue den Suffixbaum zu T;
(2) Suche im Baum den (eindeutigen) Pfad von der Wurzel zu einem internen Knoten bzw. zu einem Blatt, dessen konkatenierte Labels das Wort P ergeben;
(3) Falls P nach einem echten Präfix eines Kantenlabels endet, so führe dort einen neuen Knoten vom Grad 1 ein;
(4) Falls P nicht vollständig in einem Pfad enthalten ist: `return false`;
(5) v:=Endknoten des Pfades;
(6) Führe BFS oder DFS im Teilbaum des Knotens v aus und gib die die Indizes aller Blätter dieses Teilbaums aus;

Die Ausgabe ist dann entweder die Menge aller Startindizes des Teilstrings P in T oder die Information, daß das Suchmuster in T nicht enthalten ist. Die Korrektheit folgt sofort aus der Definition von Suffixbäumen. Auch die Laufzeitabschätzung ist trivial: Schritt (1) kostet Zeit $O(|T|)$, Schritt (2) kostet $O(|P|)$, und für Schritt (6) sind die Kosten der Traversierung gegeben durch die Größe des Teilbaums unter v. Diese ist aber proportional zur Anzahl k der Blätter des Teilbaums.

b) Wie kann man die Komplexität der Suche nach P auf $\mathrm{O}(|T| + |P|)$ senken, wenn nur höchstens ein Vorkommen von P in T auszugeben ist?

Lösungsvorschlag:
Wollen wir nur höchstens ein Vorkommen des Suchmusters P in T, so kann die Zeit für Schritt (6) eingespart werden. Dazu genügt es aber nicht, die Graphentraversierung nach dem Auffinden des ersten Blattes abzubrechen und dieses auszugeben. Denn die Höhe des durchlaufenen Teilbaums kann u.U. zu groß sein. Es kann aber das Preprocessing, d.h. der Aufbau des Suffixbaums, leicht erweitert werden: Bei jedem Knoten w wird der Index eines Blattes im Teilbaum von w gespeichert. (Dies kann etwa in $O(|T|)$ Zeit durch einmalige Traversierung des Suffixbaums geschehen) Dieser Index wird in Schritt (6) für $v = w$ ausgegeben.

c) Welchen Platz belegt eine optimal implementierte Datenstruktur für einen Suffixbaum eines Textes der Länge n?

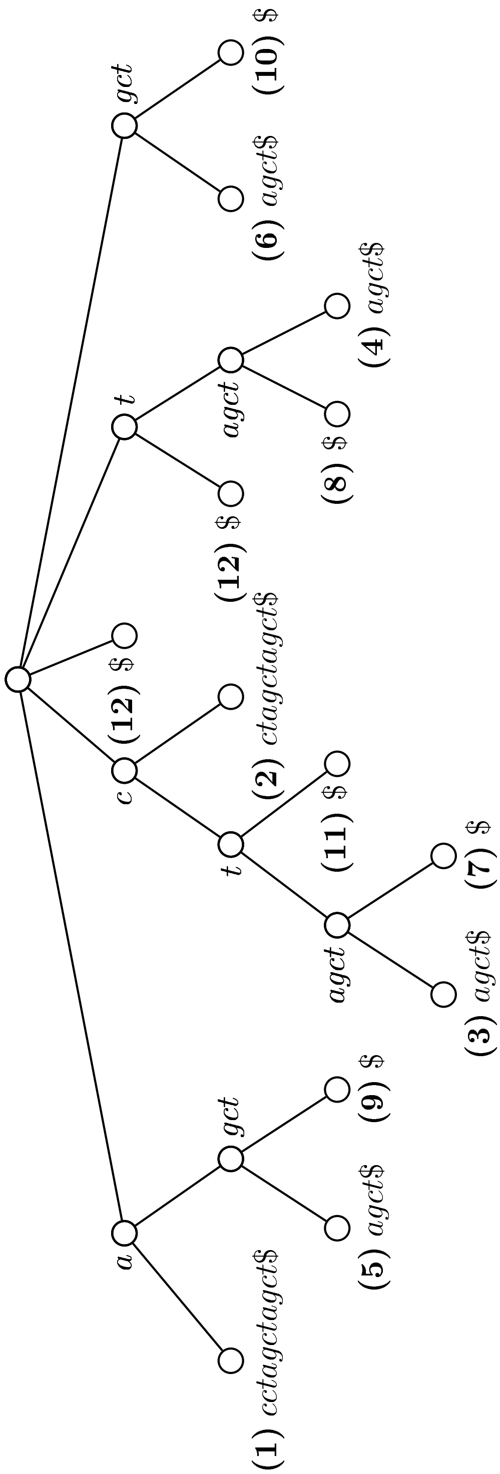
Lösungsvorschlag:
Eine naive Datenstruktur würde zu jeder Kante des Suffixbaums den entsprechenden Teilstring von T speichern. Da aber jedes Suffix der Länge $0, 1, \ldots, n$ genau einmal vorkommt, könnte der Platzaufwand allein hierfür bereits $\Omega(n^2)$ betragen. (Ein Beispiel mit hinreichend großem Alphabet ist leicht zu konstruieren.) Wird stattdessen pro Kante nur ein Paar von Pointern gespeichert, die innerhalb von P den gewünschten Teilstring markieren, so ist der Platzaufwand für die Labels beschränkt durch die Kantenzahl im Baum. Diese hängt aber, genau wie die Knotenzahl, linear von der Anzahl der Blätter des Baums ab, und diese ist gleich $n+1$. Auch die Größe der Datenstruktur für den Baum selbst (d.h. die Anzahl der Knoten und Kanten) ist linear in der Anzahl der Blätter. Also benötigt die gesamte Struktur Platz $\Theta(n)$.

Aufgabe 3

Gegeben sei eine Menge $M = \{s_1, \ldots, s_m\}$ von Strings mit $n = \sum_{i=1}^{m} |s_i|$. Geben Sie einen Algorithmus an, mit dem man in Zeit $\mathrm{O}(n)$ alle Strings $s_i \in M$ finden kann, die Teilstrings eines anderen Strings $s_j \in M$ sind.

Lösungsvorschlag:
Konstruiere den kanonischen *gemeinsamen Suffixbaum* für die Strings s_i, $1 \leq i \leq m$, wie folgt: Sei T_i der Suffixbaum für die Strings $s_1, \ldots, s_i$, dann ist T_1 der gewöhnliche Suffixbaum für das Wort s_1, und T_i entsteht aus T_{i-1} durch Weiterverwenden der Wurzel von T_{i-1} und aller in T_{i-1} bereits enthaltenen Suffixe für das Wort s_i. Für die Labels der Blätter gilt folgendes: Ein Blatt erhält Label (i, p) genau dann, wenn die Konkatenation der Kantenlabels auf seinem zugehörigen Pfad das Suffix ab Position i innerhalb von s_p ist. Nun wird der gemeinsame Suffixbaum durch Elimination des Terminationszeichens so modifiziert, daß auch interne Knoten labels haben können. (Mit "\$" markierte Kanten zu Blättern fallen fort). Dann gilt für jedes p folgende Beobachtung: Es taucht das Label $(1, p)$ genau dann an einem internen Knoten auf,

wenn String s_p Teilstring eines anderen Strings der Menge ist. Die Erfülltheit unserer Zeitschranke ist offensichtlich.



Figur 1: Suffixbaum für T = *acctagctagct* (Labels sind hier an Knoten statt an Kanten gezeichnet). Dieser Baum ist nach der Definition von Suffixbäumen eindeutig.