Musterlösung
Übungszettel 4
Algorithmen & Datenstrukturen für Bioinformatiker

Adrian Haß

17. Februar 2003

1 Vom dfa zum Regulären Ausdruck

Wir konstruieren den dfa mittels der DP-Methode wie in der Vorlesung vorgestellt. Dazu werden in jedem Iterationsschritt aus den alten Minisprachen neue erzeugt nach folgender Vorschrift:

> Sei D ein dfa, dessen Sprache in einen Regulären Ausdruck zu überführen ist. Dieser habe n Zustände (entsprechend nummeriert). Dann benutzen wir folgende Rekursionsgleichung:
>
> $$R_{i,j}^k = R_{i,j}^{k-1} \cup R_{i,k}^{k-1} \cdot (R_{k,k}^{k-1})^* \cdot R_{k,j}^{k-1}, \text{ mit } i,j \in \{1,\ldots,n\}$$
>
> Dabei ist $R_{i,j}^{k-1}$ die Reguläre Sprache um von Zustand i zu Zustand j zu kommen, mit Umwegen über die Zustände aus $\{1,\ldots,k\}$. Das Prizip ist hier so ähnlich wie beim Dijkstra. Die Initialisierung ist demnach das Kantenlabel zwischen i und j. Respeichert wird das natürlich in einer Matrix.

Der dfa sieht vereinfacht so aus:

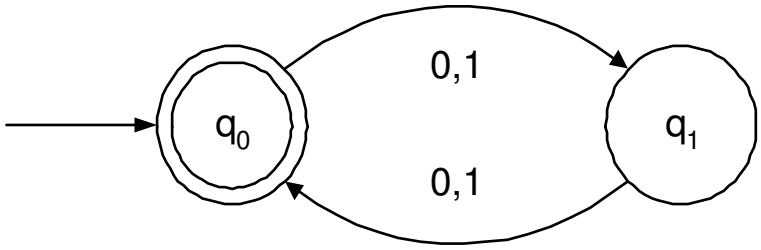

Abbildung 1: dfa für Wörter grader Länge

Demnach ergeben sich für die Minisprachen nacheinander:

$$R^0 = \begin{pmatrix} \varepsilon & (0|1) \\ (0|1) & \varepsilon \end{pmatrix} \tag{1}$$

$$R^1 = \begin{pmatrix} \varepsilon & (0|1) \\ (0|1) & (0|1)\varepsilon^*(0|1)|\varepsilon \end{pmatrix} \tag{2}$$

$$R^2 = \begin{pmatrix} (0|1)\big((0|1)(0|1)\big)^*(0|1)|\varepsilon & (0|1)\big((0|1)(0|1)\big)^* \\ (0|1)\big((0|1)(0|1)\big)^* & (0|1)\big((0|1)(0|1)\big)^*(0|1)|\varepsilon \end{pmatrix} \tag{3}$$

und die gesuchte Sprache ist nun die Vereinigung aller Sprachen, die vom Startzustand zu einem Akzeptierenden Zustand führen. Hier ist dass leicht:

$$L = R^2_{1,1} = (0|1)\big((0|1)(0|1)\big)^*(0|1)|\varepsilon = \big((0|1)(0|1)\big)^*.$$

2 ε-Übergänge

Was wir in der Vorlesung gezeigt haben ist folgendes:

$$\text{nfa} \Rightarrow \text{dfa} \Rightarrow \text{Reg. Sprache} \Rightarrow \text{nfa}.$$

Da jeder nfa (insbesondere nfa ohne ε-Übergänge) in einen dfa zu verwandeln ist und daraus wiederum eine reguläre Sprache konstruierbar ist, können wir nach dem besprochenen Prizip (Baukastensystem für die erlaubten Operationen) diese in einen nfa mit ε-Übergängen überführen, der die gleiche Sprache akzeptiert.

Anders herum ist ein dfa leicht in einen nfa ohne ε-Übergänge zu verwandeln: Man ändert die Überführungsfunktion so ab, dass statt $(z, s) \mapsto \delta(z, s) = n$ steht $\delta(z, s) = \{n\}$, also eine einelementige Menge. Dies ist dann ein nfa ohne ε-Übergänge, weil in der Überführungsfunktion des dfa keine solchen Übergänge enthalten sind.

3 Aho-Corasick/Reguläre Ausdrücke

Feature	Aho-Corasick	Reguläre Ausdrücke
Pattern-Menge	$\surd$	$\surd$
-Anzahl	endlich	abzählbar (unendlich)
Datenstruktur für Match	$\surd$	$\surd$
Laufzeit	$O(n)$	$O(k \cdot n)$

uvm. . .

4 Rechteckige Muster

Da uns mit dem Aho-Corasick nur ein effektiver Algorithmus für lineare Patternmengen zur Hand ist, überlegen wir uns eine sinnvolle Vorverarbeitung der rechteckigen Muster $\{p_1, \ldots, p_q\}$ bzw. des Texts T. Der Text wird einfach linearisiert indem er zeilenweise (durch Sonderzeichen \$ getrennt) konkateniert wird ($\rightsquigarrow \overline{T}$). Aus den Pattern wird ein Aho-Corasick-Baum $\mathcal{K}$ erzeugt und zwar wie folgt:

Die Zeilen p_i^j der Muster werden sukzessive in den Baum eingefügt ohne doppelte Zeilen zuzulassen und von $1 \dots n$ durchnummeriert. Währenddessen wird für jedes Muster ein Array/String z_i erzeugt, der codiert, aus welchen Zeilen (-nummern) es sich in welcher Reihenfolge zusammensetzt.

Nun wird mit diesem Baum der Aho-Corasick auf den linearisierten Text $\overline{T}$ angewandt, dabei werden alle Matches auf die entsprechenden 2D-Koordinaten umgerechnet und in einer Matrix der Größe von T über die Patternnummer gespeichert. Sei dabei u Postition des Match in $\overline{T}$, (u, v) die Koordinaten in T und l die Breite von T, dann gilt

$$(v, w) = (\lfloor u/(l-1) \rfloor, u \mod (l-1)).$$

Wir wissen jetzt, an welchen Stellen in T die einzelnen Zeilen vorkommen, doch es gilt noch einerseits die richtigen Musterzeilen zu finden, die andererseits auch in derselben Spalte untereinander beginnen. Dazu wenden wir wieder Aho-Corasick an, jedoch mit den z_i als Pattern über dem nun spaltenweise linearisierten Text an. Diese geben ja genau an, welche Patternnummern aus $\mathcal{K}$ untereinandergeschrieben wieder das Muster p_i ergeben.

Nun ist jeder Beginn eines z_i (analog zu oben umgerechnet) genau die Koordinate eine Linken, oberen Ecke eines Matches des rechteckigen Musters p_i.

5 Wild Cards

Wir unterscheiden die drei Fälle:

(1) *** nur in T:** Dann wissen wir, dass wir an jedem * ein Match auftritt. Es gilt aber noch richtige Matches zu finden und solche, die in die * hineinragen bzw. Präfix rechts und Suffix links davon beginnen. Ersteres ist leicht, also mit Linearzeitmatching und * mit $ ersetzt zu erledigen. Letzteres erscheint zunächst schwer, aber es geht mit einer Modifikation des Z-Algorithmus auch in linearer Zeit:

> Wir ersetzen wieder * durch $ und wenden Z-Algo mit $P\$T$ an. Doch jetzt suchen wir nicht nur Z-Boxen der Länge $|P|$, sondern in der Nähe eines $ im Text auch solche, die genau so lang sind wie die Entfernung zum $. Dies entspricht einem in * von rechts hereinhängenden Pattern. Analog für die andere Seite nur mit $P^r\$T^r$ und danach suchen wir noch beidseitige Matches, die die Bedingung erfüllen müssen, dass |rechtes Ende| + |linkes Ende| $\leq |P|$.

(2) *** nur in P:** Dann reicht es nach Vorkommen der verschiedenen Teile von P, die durch * getrennt werden in der richtigen Reihenfolge zu suchen ($\rightsquigarrow$ Aho-Corasick, linear). Das Auffinden der richtigen Reihenfolge lässt sich (obwohl erst nicht so offensichtlich) auch in linear Zeit bewerkstelligen, denn ich muss die angeordneten Teile als Subsequenz in einer Liste der Matches finden. (DP!)

(3) *** in beiden:** Nun können wir nicht anders als zu jedem Stück von P auch alle Präfixe und Suffixe in den Aho-Baum einzufügen (denn Z-Algo

geht nicht für mehrere Pattern), dann zu suchen und nicht an die * grenzende Präfixe und Suffixe herauszufiltern. Die ganzen Präfixe und Suffixe (explizit oder implizit) zu erzeugen verschlechtert aber die Laufzeit zum quadratischen.

Bemerkung: Es war hier bei weitem nicht so viel für die vollen Punkte gefordert.