Musterlösung
Test
Algorithmen & Datenstrukturen für Bioinformatiker

Adrian Haß

10. Dezember 2002

1. Aufgabe

(a)

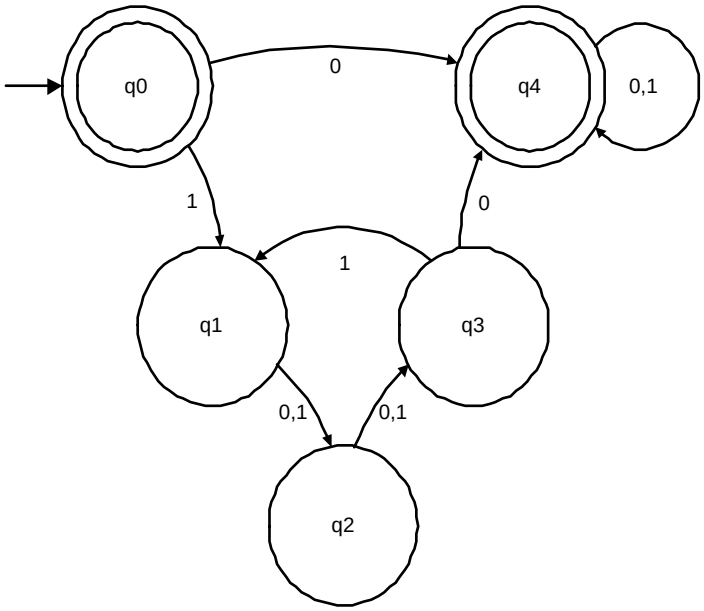

Abbildung 1: dfa für $L = \big((0|1)(0|1)(0|1)\big)^*0(0|1)^*|\varepsilon$

(b)

Generell ist es nicht richtig, dass der so konstruierte nfa die Komplementsprache L^c akzeptiert. Dazu ist ein Beispiel zu finden für:

- ein Wort w aus Σ^*, dass für einen A und dazu A' akzeptiert wird oder
- ein Wort w' aus Σ^*, dass von keinem der beiden akzeptiert wird.

Ein Gegenbeispiel konstruiert man leicht wie folgt:

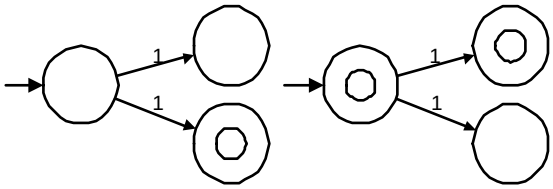

Abbildung 2: Gegenbeispiel mit $w \in A$ und $w \in A'$

Hier wir das Wort $w = 1$ von A sowie A' akzeptiert, somit akzeptiert A' nicht L^c.

Alternativ kann man die Behauptung auch so beweisen:

> Sei $w \in \Sigma^*$ ein Wort das von A akzeptiert wird und E der Zustandsraum, den A beim Lesen von w erreicht. Enthält jetzt E mindestens einen akzeptierenden und einen nicht akzeptierenden Zustand q, dann ist dieser Zustand nach Konstruktion in A' akzeptierend. Wird jetzt w von A' gelesen, dann wird genauso E erreicht und darin ist q akzeptierend, also akzeptiert auch A' das Wort w, womit die Behauptung gezeigt wäre.

Die Konstruktion ist jedoch richtig, wenn A ein dfa ist, denn dann besteht E nur aus einem Element und ist somit nur entweder akzeptierend oder nicht.

Bewertung

zu (a) dfa: 1P. für Sackgasse mit 0; 1P. Startzustand akz.; 1P. für richtigen 3-er Kreis
Regulärer Ausdruck: 2P. für korrekten Ausdruck; -1P. ε fehlt/kleinere Fehler; -0.5 kleine Notationsfehler

zu (b) 1P. für passenden A; 1P. für richtige Zustände in A'; 1P. für genanntes "Versager"-Wort.

2. Aufgabe

Siehe Gusfield 2.4.1 (Seite 26)...
Die richtigen Werte für das Beispiel lauten:

i:	1	2	3	4	5	6	7	8
T:	a	n	n	a	n	a	s	s
Z_i:	0	0	0	2	0	1	0	0
sp'_i:	0	0	0	0	2	1	0	0
sp_i:	0	0	0	1	2	1	0	0

Bewertung

shift-Regel: Vgl./Schiebe von L→R 1P.; Mismatch $\rightsquigarrow$ verschieben um $i - sp'_i$ (oder verbal) 1P.;
Vorverarbeitung Z-Algo: Erwartung Ausgabe Z-Algo. 1P.; Berechnen i aus j bzw. rechtes Ende von Z-Box an Stelle j; $sp'_i = Z_j$(!!) 1P.
Beispiel: sp'_i-Werte 1P. (-0.5 sp_i); Z_i-Werte 1P.

3. Aufgabe

(a)

Die einfachste und anschaulichste Methode für das Finden von Teilsequenzen von T (denn solche sind die Präfixe von den Suffices $T[i..m]$), die nur aus Zeichen in S bestehen, arbeitet wie folgt:

- Initialisiere Array A der Länge m mit Nullen;
- Durchlaufe den Text T ***von hinten*** und inkrementiere einen Zähler k bei Zeichen aus S oder setze diesen auf 0;
- Speichere Wert von k an der entsprechenden Stelle in A.
- Die Werte im Array A sind jetzt die gesuchten Matchlängen.

Folgendes Beispiel mag zur Illustration dienen:

> $\Sigma = \{0, 1\}$; $T = 010111011011110$ und $S = 1$, dann sieht die Ausgabe so aus:

T	0	1	0	1	1	1	0	1	1	0	1	1	1	1	0
A	0	1	0	3	2	1	0	2	1	0	4	3	2	1	0

Die Durchläufe durch den Text kosten trivialerweise nur $O(m)$ jedoch bereiten die Zugriffe auf S Schwierigkeiten, denn sie sind im allgemeinen nicht schneller als $O(\log |S|)$. Doch diese Kosten sind ja bei gegebenem Σ fix und kleiner als m, somit läuft alles in $O(m)$.

Man kann die Aufgabe auch als Suche nach allen Wörtern, die aus Buchstaben von S bestehen interpretieren. Da es davon $\sum_{i=1}^{m} i^{|S|}$ Stück gibt, wäre

man schlecht beraten diese explizit zu konstruieren und mit dem Aho-Corasick-Algorithmus zu suchen. Die einzige Möglichkeit eine derartige Menge von Pattern geschlossen darzustellen ist ein Regulärer Ausdruck.

Man erzeuge sich also einen Regulären Ausdruck für die Sprache S^+ und wende den bekannten Algorithmus für das Matchen von Regulären Ausdrücken mit $\Sigma^* S^+$ an. Die Laufzeit beträgt $O(|S|m)$ und da wieder $|S|$ konstant ist, ist das in $O(m)$.

(b)

Auch hier stehen wir vor dem gleichen Problem wie in (a), somit wenden wir wieder die gleiche Strategie an, nur konstruieren wir mit S^+ die Menge aller aus den Strings in S gebildeten Wörter. Diese können wir dann wiederum gegen jeden gegebenen Text matchen und die Laufzeit berechnet sich analog, nur dass jetzt für die Konstante gilt: $c = \max\{|w| : w \in S\}$, aber auch dies liegt in $O(m)$.

Bewertung

zu (a) Beispiel: T und S 1P.; Illustration 1P.; Laufzeit 1P.;
Algorithmus: in $O(m)$ (mögl./ist) 2P.; Vorgehen löst Aufgabe 1P.

zu (b) Name und es geht damit 1P.; Genaue Beschreibung 1P.