Entwurf und Analyse von Algorithmen

WS 1999/2000
Prof. Dr. K. Madlener
Übungsblatt 7

Aufgabe 7.1. Wir betrachten das Problem der Äquivalenz vollständiger deterministischer endlicher Automaten (DEA).

Gegeben seien zwei DEA $M_1 = (Q_1, \Sigma, \Pi_1, s_1, F_1)$ und $M_2 = (Q_2, \Sigma, \Pi_2, s_2, F_2)$ (gleiches Alphabet!) mit $Q_1 \cap Q_2 = \emptyset$. Wir nennen M_1 und M_2 äquivalent genau dann, wenn $L(M_1) = L(M_2)$. Weiter nennen wir $q_1 \in Q_1$ und $q_2 \in Q_2$ äquivalent genau dann, wenn für jedes Wort $x \in \Sigma^*$, $q_1 x \vdash_{M_1} f_1$ und $q_2 x \vdash_{M_2} f_2$ genau dann $f_1 \in F_1$ gilt, wenn $f_2 \in F_2$. Machen Sie sich klar, dass, wenn zwei Zustände $q_1 \in Q_1$ und $q_2 \in Q_2$ äquivalent sind, auch die Zustände $q_{1,a} \in Q_1$ und $q_{2,a} \in Q_2$ mit $q_1 a ::= q_{1,a} \in \Pi_1$ und $q_2 a ::= q_{2,a} \in \Pi_2$ für alle $a \in \Sigma$ jeweils äquivalent sind. Kann ein akzeptierender Zustand aus Q_1 äquivalent zu einem nicht akzeptierenden Zustand aus Q_2 sein (oder umgekehrt)?

Konstruieren Sie einen Algorithmus, der entscheidet, ob zwei DEA M_1 und M_2 äquivalent sind, indem Sie anfänglich annehmen, dass die Zustände s_1 und s_2 äquivalent seien und dann wiederholt die obige Eigenschaft (die Äquivalenz von Zuständen betreffend) ausnutzen, um $Q_1 \cup Q_2$ geeignet zu partitionieren. Ihr Algorithmus sollte dann durch Inspektion der entstandenen Partition die Äquivalenz von M_1 und M_2 entscheiden können und idealerweise im schlechtesten Fall eine Laufzeit in $O(n \log^* n)$ besitzen, wobei $n = |Q_1| + |Q_2|$ sei und $|\Sigma|$ als konstant angenommen werde.

Aufgabe 7.2. Der Lowest Common Ancestor (LCA; zu deutsch etwa: tiefster gemeinsamer Vorfahr) zweier Knoten u und v eines (Wurzel-)Baumes ist der Knoten größter Tiefe im Baum, der zugleich Vorfahr von u und v ist. Beim LCA-Problem ist ein (Wurzel-)Baum T und eine Menge $P = \{\{u, v\}, \ldots\}$ ungeordneter Paare von Knoten aus T gegeben; man will den LCA eines jeden Paares in P bestimmen.

Wir verwenden dazu folgenden Algorithmus, der mit LCA($wurzel[T]$) aufgerufen werde; wir gehen dabei davon aus, dass zu Beginn jeder Knoten in T weiß gefärbt ist:

```
procedure LCA(u)
begin
    MakeSet(u)
    vorfahr[FindSet(u)] := u
    for jedes Kind v von u in T
        LCA(v)
        Union(u, v)
        vorfahr[FindSet(u)] := u
    farbe[u] := Schwarz
    for jeden Knoten v mit {u, v} ∈ P
        if farbe[v] = Schwarz
            print "LCA von " u " und " v " ist " vorfahr[FindSet(v)]
```

Dabei soll in naheliegender Weise die Verwaltung disjunkter Mengen zum Einsatz kommen: MAKESET(u) erzeugt eine Menge mit einem einzelnen Element u (das bislang kein Element einer anderen Menge sei), FINDSET(u) gibt einen Repräsentanten derjenigen Menge zurück, zu der das Element u gehört, und UNION(u, v) erzeugt eine neue Menge, die die Vereinigung der disjunkten Mengen ist, die von den Elementen u bzw. v repräsentiert werden; diese beiden Mengen werden dabei „zerstört".

Zeigen Sie, dass die letzte Zeile für jedes Paar $\{u, v\} \in P$ genau einmal ausgeführt wird.

Zeigen Sie weiter, dass zum Zeitpunkt des Aufrufes LCA(u) die Anzahl der disjunkten Mengen in der Mengen-Datenstruktur gleich der Tiefe von u in T ist.

Begründen Sie, dass obiger Algorithmus tatsächlich für jedes Paar $\{u, v\} \in P$ den jeweiligen LCA ausgibt.

Analysieren Sie abschließend die Laufzeit des Algorithmus im schlechtesten Fall bei Verwendung der effizienten Verwaltung disjunkter Mengen, die aus der Vorlesung bekannt ist.

Aufgabe 7.3. a) Gegeben Sei das Alphabet $\{0, \dots, 9\}$, der Text 3141592653589793 sowie das Muster 26. Wenn man modulo 11 arbeitet, wieviele (und welche) falsche Treffer treten bei der Rabin-Karp-Methode auf?

b) Gegeben Sei das Problem, in einem $n \times n$-Raster von Buchstaben ein $m \times m$-Muster zu finden. Zeigen Sie, wie man die Methode von Rabin und Karp für die Lösung dieses Problems anpassen kann.

Aufgabe 7.4. In dieser Übung werden wir eine String-Matching-Methode von Ricardo Baeza-Yates und Gaston Gonnet kennenlernen, die auf Bitmanipulationen basiert.

Dazu betrachten wir ein (Such-)Muster P der Länge n und einen Text T der Länge m. Sei M ein $n \times m$-Array mit binären Einträgen (also 0 oder 1). Index i laufe von 1 bis n und Index j laufe von 1 bis m. Der Eintrag $M(i, j)$ ist 1 genau dann, wenn die ersten i Zeichen von P exakt mit den i Zeichen, die in T bei Position j enden, übereinstimmen (d. h. $M(i, j) = 1$ gdw. $P[1..i] = T[j - i + 1..j]$); sonst ist $M(i, j) = 0$. Bestimmen Sie zur Probe M für $T = \textit{californian}$ und $P = \textit{for}$.

Ist nun M vollständig gegeben, wie ermittelt man alle Vorkommen von P in T?

Um M zu berechnen, konstruieren wir zunächst einen binären n-Vektor $U(x)$ für jeden Buchstaben x des Alphabets, wobei $U(x)$ an genau den Positionen auf 1 gesetzt wird, wo x in P vorkommt. Berechnen Sie als Beispiel $U(a)$ für $P = \textit{abacdeab}$.

Die Idee ist, M Spalte für Spalte zu konstruieren. Wie muss die erste Spalte gesetzt werden? Erklären Sie, wie man die anderen Spalten nun mit Hilfe der jeweils vorhergehenden Spalte und den Vektoren U durch einfache Operationen auf Binärvektoren bestimmen kann. Begründen Sie, warum Ihr Verfahren korrekt ist. Führen Sie das Verfahren für $P = \textit{abaac}$ und $T = \textit{xabxabaaxa}$ durch (Alphabet $\{a, b, x\}$).

Bestimmen Sie die Anzahl der Bitoperationen im schlechtesten Fall. Unter welchen Umständen ist diese Methode des String-Pattern-Matching effizient? Wie sieht es mit dem benötigten Platz des Verfahrens aus?

Informationen zur Vorlesung:
http://www-madlener.informatik.uni-kl.de/ag-madlener/lectures/eaa/ws1999-2000/eaa.html