Prof. Dr. Norbert Blum
Maria Nikolaidou
Claus Rick

Institut für Informatik, Abt. IV

Vorlesung Informatik III
WS 1998/99
9. Übung

15. 12. 1998

Achtung! Wichtiger Hinweis:

Aus gegebenem Anlaß möchten wir daran erinnern, daß eine Voraussetzung für den Erwerb eines Übungsscheins die regelmäßige Mitarbeit – und daher die Anwesenheit – in den Übungsgruppen ist. Bei kontinuierlichem Fehlen behalten wir uns eine Anrechnung der Punkte aus den Übungszetteln vor.

Aufgabe 1:

Eine wichtige Anwendung von Tries ist das Stringmatching, d.h. die Suche nach einem oder mehreren Vorkommen eines Patternstring P in einem Text T. Ist der Text T statisch und werden viele Anfragen nach dem Vorkommen verschiedener Patternstrings gestellt, die möglichst schnell beantwortet werden sollen, so bietet sich an, für den Text eine Art Index zu erstellen. Dazu wird für den Text T ein sogenannter *Suffix Trie* erstellt. Dies ist ein Trie, in den jeder Suffix von T eingefügt wird, d.h. die Schlüsselmenge ist gerade die Menge der Suffixe von T (um die Präfixfreiheit zu garantieren, wird an T ein Sonderzeichen angefügt).

a) Konstruieren Sie einen Suffix-Trie für den Text *adcdacdad*. **(1P)**

b) Zeigen Sie, daß es eine Bijektion zwischen den Positionen des Textes und den Blättern des zugehörigen Tries gibt. **(1P)**

c) Welche worst-case Laufzeit hat ein Algorithmus, der einen Suffix-Trie so konstruiert, wie Sie dies in a) von Hand getan haben? **(1P)**

d) Angenommen der fertige Suffix-Trie für einen Text T steht zur Verfügung. Entwickeln Sie einen Algorithmus, der für einen Patternstring P feststellt, ob dieser in T vorkommt und falls ja, das erste bzw. alle Vorkommen (d.h. Startpositionen in T) ausgibt. Begründen Sie die Korrektheit Ihres Verfahrens. Welche Laufzeit hat Ihr Suchalgorithmus? **(4P)**

e) Eine weitere Problemstellung, bei der Tries zum Einsatz kommen, ist die folgende. Wie kann man für zwei Texte T_1 und T_2 einen gemeinsamen Suffix-Trie konstruieren, um mit seiner Hilfe den längsten gemeinsamen Teilstring von T_1 und T_2 zu berechnen (der längste gemeinsame Teilstring von $T_1 = superiorcalifornialives$ und $T_2 = sealiver$ wäre z.B. *alive*)? **(2P)**

Aufgabe 2:

Skizzieren Sie die Hashtafel bei Kollisionsbehandlung mittels verketteter Listen für $m = 9, S = \{5, 28, 19, 15, 20, 33, 12, 17, 10\}, h(x) = x \bmod 9$. **(1P)**

Aufgabe 3:

Wie könnte die Überlaufbehandlung von Hashtabellen mit n Schlüsseln modifiziert werden, um die worst-case Laufzeit zu verbessern (die Verwaltung der Überläufer soll weiterhin außerhalb der Hashtabelle erfolgen)? Wieviele Schritte werden im worst-case für die Operationen Zugriff, Einfügen und Löschen benötigt. **(2P)**

Aufgabe 4:

Gegeben sei eine anfangs leere Hashtabelle mit 7 Elementen, in die der Reihe nach die Schlüssel $12, 53, 5, 15, 2, 19$ mit sog. double Hashing (d.h. $h(k,i) = (h_1(k) - i \cdot h_2(k)) \bmod m)$) eingefügt werden sollen. Die zu verwendenden Hashfunktionen seien $h_1(k) = k \bmod 7$ und $h_2(k) = 1 + k \bmod 5$.

a) Geben Sie eine Belegung der Hashtabelle an. **(1P)**

b) Geben Sie eine Belegung der Hashtabelle an, wenn die Reihenfolge $53, 5, 15, 2, 19, 12$ ist. **(1P)**

c) Wie ist die durchschnittliche Suchzeit der beiden Belegungen? **(1P)**

d) Wie hoch ist die maximal denkbare durchschnittliche Suchzeit? Konstruieren Sie ein entsprechendes Beispiel. **(1P)**

e) Ersetzen Sie die Funktion $h_2 = 1+k \bmod 5$ durch $h_2 = k^2 \bmod 5$. Ist dies eine geeignete zweite Hashfunktion für h_1? Wie sollte die Funktion h_2 gewählt werden, damit möglichst wenig Kollisionen auftreten? **(2P)**

f) Eine Alternative eine Adresskollision zu lösen, wäre, anstatt den alten Schlüssel an Position $T[i]$ zu belassen, den neuen dort einzutragen und den alten weiterzuleiten. Führen Sie dies bei obigem Beispiel mit einem Schlüssel durch? Wie ändert sich ihre Suchzeit? **(1P)**

Aufgabe 5:

Die Anzahl der Elemente, die in einer Hashtabelle angeordnet werden soll, ist im voraus bekannt. Unter welchen Bedingungen ist Kollisionsbehandlung mittels verketteter Listen dem double Hashing vorzuziehen? **(1P)**