Prof. G. Zachmann
David Mainzer

TU Clausthal
Institut für Informatik
8. Juli 2010

Sommersemester 2010

Übungen zu Informatik II - Blatt 13

Abgabe in der Übung am 13. 07 / 14. 07. 2010

Bitte beachten Sie, dass die Programmieraufgaben von Ihnen in der Übung vorgeführt und erklärt werden müssen. Zusätzlich senden Sie die Lösung, unter Angabe ihres Namens, an **dm@tu-clausthal.de**.

Aufgabe 1 (Suchbäume, *2 Punkte*)

Welchen Vorteil haben B-Bäume mit Parameter k gegenüber allgemeinen k-Wege-Bäumen bzgl. der Höhe? Begründen Sie ihre Antwort.

Aufgabe 2 (Patricia-Tries, *1.5+3 Punkte*)

Die Abbildung 1 zeigt ein Beispiel für einen Patricia-Trie (eine Erläuterung was ein Patricia-Trie ist, befindet sich am Ende des Übungsblattes). Gehen Sie davon aus, dass wir im Folgenden einen solchen Patricia-Trie verwenden. Die Knoten enthalten die Position im Bit-String, beginnend bei 1, in welcher sich der rechte und linke Zweig unterscheiden.

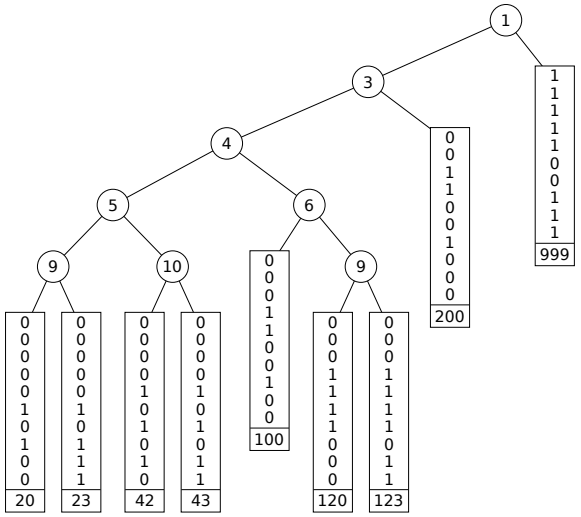

Abbildung 1: vereinfachter Patricia-Trie (ohne Aufwärts-Pointer)

a) Beschreiben Sie einen Algorithmus in Pseudo-Code, welcher nach einem beliebigen Element im Patricia-Trie sucht.

b) Beschreiben Sie einen Algorithmus in Pseudo-Code, welcher das Löschen eines Elementes in den Patricia-Trie durchführt. Beachten Sie das eventuelle Aktualiesieren/Anpassen der Knotenwert.

Patricia-Tries

Bei Patricia-Tries handelt es sich um eine Variante der binären Tries. Sie vermeiden Pfade im Baum ohne Gabelung (sog. "Einweg-Pfade"). Wir betrachten hier in dieser eine stark vereinfachte Variante der Patricia-Tries, die wir im folgenden trotzdem genau so nennen wollen. Patricia Tries wurden 1968 von Morrison vorgestellt.

Die **Ideen**:

Speichere an den inneren Knoten die Anzahl Bits, die übersprungen werden können und nicht getestet werden brauchen (weil es sowieso keine Gabelung auf dem Pfad darunter gibt, d.h., alle Keys in diesem Teilbaum bis dahin den gleichen Präfix haben).

Die Abbildung 2 zeigt auf der linken Seite einen Baum, welcher entlang seiner Kante die Bit-Werte trägt. Um einen Bit-String, bestehend aus 4-Bit zu vergleichen, sind 4 Kanten durchläufe nötig. Ein entsprechender Patricia-Trie, siehe rechter Teil der Abbildung 2, benötigt nur eine Kante, da im Knoten die "critical bit number" gespeichert ist. Es ist also nur der Vergleich dieses Bit's nötig, um die beiden Bit-Strings zu vergleichen. Hieran kann man deutlich erkennen, wie die sogenannten "Einweg-Pfade" durch einen entsprechenden Knoten ersetzt wurden.

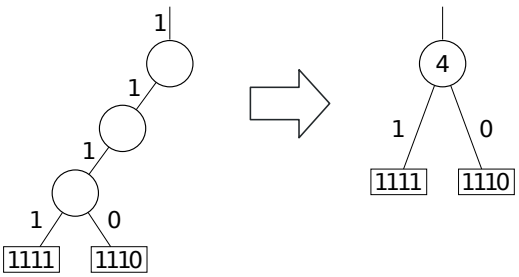

Abbildung 2: Beispiel für das Umwandeln eines binär Trie in einen Patricia-Trie

Aufgabe 3 (Floating Point, *2 Punkte*)

Sei $x \in \mathbb{R}$ gegeben und $\hat{x} \in \mathbb{F}$ die Floating-Point-Zahl, die aus x unter dem Rundungsmodus "round to nearest" hervorgeht. Begründen Sie, warum $|\hat{x} - x| \leq \varepsilon_m |x|$.

a) Sei $x \in \mathbb{R}$ gegeben und $\hat{x} \in \mathbb{F}$ die Floating-Point-Zahl, die aus x unter dem Rundungsmodus "round to nearest" hervorgeht. Begründen Sie, warum $|\hat{x} - x| \leq \varepsilon_m |x|$ gilt.
   Tip: Überlegen Sie sich die worst-cases beim Runden auf der Bit-Ebene. Zur Vereinfachung können Sie einfach den Rundungsmodus wählen, der Ihnen am einfachsten zu sein scheint.

b) Ausgehend von $|\hat{x} - x| \leq \varepsilon_m |x|$, zeigen Sie, dass

$$|x|(1 - \varepsilon) \leq |\hat{x}| \leq |x|(1 + \varepsilon)$$

   Tip: Erinnern Sie sich an die Rechenregeln bzgl. des Betrages.

Aufgabe 4 (Floating Point, *3 Punkte*)

Gegeben sei das nachstehende Beispiel-Programm 1 in Python. Zu Beginn wird die Variable `sum` auf den Wert `0.0` gesetzt (Zeile 4). Anschließend wird diese Variable in der **`for`**-Schleife 10 Mal um den Wert `0.1` erhöht (Zeile 9).

```
1  #/bin/python
2  # -*- coding: utf-8 -*-
3
4  sum = 0.0
5  n = 10
6  d = 1.0 / n
7  print "d = %.60f \n" % ( d )
8  print "%2i: sum = %.60f" % (0, sum )
9  for i in range(n):
10     sum += d
11     print "%2i: sum = %.60f" % (i, sum )
12
13 print "\nErgebnis:"
14 print "%f -- %.60f" % (sum, sum )
```

Quellcode 1: Beispiel-Programm zu "Numerische Robustheit" in Python

a) Geben wir nun die Variable `sum` aus, erhalten wir den Wert `0.999999999999999888977697537484345957636833190917968750000000` anstelle des erwarteten Wertes `1.0`. Erklären Sie was passiert ist und wie es zu diesem Fehler kommt. Führen Sie das Programm einmal selber aus um zu sehen was genau passiert.

b) In Zeile 7 wird das Delta `d` ausgegeben und beträgt $0.1(1+\delta)$ (`d = 0.1000000000000000055511151231257827021181583404541015625000000`), wobei $\delta < \varepsilon_m$ ist. Erklären Sie, warum am Ende `sum` < 1.0 gilt