Technische Universität München
Institut für Informatik
Dr. rer. nat. Tobias Weinzierl
Kaveh Rahnema

März 2013

Kompaktkurs Einführung in die Programmierung

Übungsblatt 10: Binärbäume

Lernziele

- Rekursive Datenstrukturen.
- Realisierung und Verwendung von Binärbäumen.

1) Einfügen in Suchbaum (P)

Schreiben Sie eine Funktion zum Einfügen in einen Suchbaum. Machen Sie sich hierzu mit der rekursiven Definition von binären Suchbäumen vertraut. Auf der WWW-Seite der Vorlesung finden Sie hierzu einen Foliensatz 'Baeume.pdf'. Nehmen sie folgende Funktion als Einstiegshilfe

```
void einfuegen_in_suchbaum(struct knoten **bp, int xneu) {
    struct knoten *b;
    b = *bp;
    if (b) {
        if (xneu < b->x)
            // in linken Teilbaum einfuegen
            einfuegen_in_suchbaum(&(b->l), xneu);
        else
            // in rechten Teilbaum einfuegen
            einfuegen_in_suchbaum(&(b->r), xneu);
    } else {
        // Neues Element in Suchbaum einfuegen
        *bp = neu(xneu, NULL, NULL);
    }
}
```

und ergänzen Sie sie um Anweisungen, die Meldungen ausdrucken, mit denen man den Ablauf des Einfügeprozesses nachvollziehen kann (z.B.: Beginn und Ende der Funktion, was für Vergleiche durchgeführt wurden und was für Konsequenzen daraus gezogen wurden).

2) Baum aufbauen (P)

Schreiben Sie nun eine Funktion `feld_einfuegen` mit einem Feld ganzer Zahlen a und einer ganzen Zahl n, die die Zahl der Komponenten von a angebe, und einem Binärbaum als Ergebnis. Die Komponenten des Feldes sollen der Reihe nach in einen anfänglich leeren Suchbaum eingefügt werden, das Ergebnis der Funktion soll der erzeugte Suchbaum sein.

3) Baum ausgeben (P)

Schreiben Sie eine Prozedur, die einen Suchbaum folgendermaßen ausdruckt: für jeden Knoten wird

- zunächst der linke Teilbaum (rekursiv) ausgedruckt,
- dann die Wurzelbeschriftung (nur die Zahl und ein Leerzeichen zum Trennen von der nächsten Ausgabe),
- und schließlich der rechte Teilbaum (rekursiv) ausgedruckt.

(Wen es interessiert: diese Art, einen Baum abzulaufen, heißt *Infix-Durchlauf*; auf den Folien hatten wir zum Drucken einen *Präfix-Durchlauf* – Wurzel vor den beiden Söhnen).

4) Beides zusammen (P)

i) Implementieren Sie nun noch eine Funktion mit Parametern wie bei „Baum aufbauen“, die erst mittels der dort geschriebenen Funktion den Suchbaum aufbaut und ihn dann mit der Funktion von „Baum ausgeben“ ausdruckt.

ii) Was macht die Funktion vom Standpunkt des Benutzers aus gesehen, der das Innenleben des Programms nicht kennt?

iii) ★ Achten Sie wiederum auf korrekte Freigabe aller Daten des Freispeichers.

5) Parallelisierung einer 3D MinMax-Suche (★)

Ziel dieser Aufgabe ist es das **minimale** und **maximale** Element eines 3-dimensionalen Gebiet

$$3 \times \text{arrsize} \times \text{arrsize} \qquad \text{mit} \qquad \text{arrsize} \gg 1$$

zu suchen und auszugeben. Das Gebiet wird durch ein Array `A[3][arrsize][arrsize]` modelliert. Folgender sequentieller Programmcode kann dazu verwendet werden:

```
// A entsprechend initialisiert
int min,max;
min = A[1][1][1];
max = A[1][1][1];
for ( int i = 0 ; i < 3 ; i++ )
    for ( int j = 0 ; j < arrsize ; j++ )
        for ( int k = 0 ; k < arrsize ; k++ )
        {
            if ( A[i][j][k] < min ) min = A[i][j][k];
            if ( A[i][j][k] > max ) max = A[i][j][k];
        }
```

i) Implementieren Sie zunächst eine sequentielle Variante der Suche ohne OpenMP.

ii) Parallelisieren Sie nun Ihre Suche mit Hilfe von OpenMP. Direktiven zur Parallelisierung **einer** Schleife reichen. Machen Sie sich – wenn nötig – Gedanken über ausreichende Synchronisation der Threads.

iii) Realisieren Sie eine Zeitmessung um die benötigte Zeit zur Suche zu messen. Vergleichen Sie die Berechnungszeiten für verschiedene Dimensionen `arrsize`, z.B. `arrsize` $= 10^5$.