Technische Universität Braunschweig
Institut für
Betriebssysteme und Rechnerverbund
Prof. Dr. H. Langendörfer · Prof. Dr. M. Zitterbart

Übungen zu Algorithmen und Datenstrukturen II

Blatt 5

11. Aufgabe: Suchbäume

(a) Geben Sie den binären Suchbaum an, der entsteht, wenn sie die Schlüssel 10, 5, 15, 9, 12, 14, 16, 8 der Reihe nach in einen anfangs leeren Baum einfügen.

(b) Welcher Baum entsteht, wenn der Schlüssel 10 entfernt wird?

(c) Angenommen, Sie hätten eine Menge von Zahlen, die in aufsteigender Reihenfolge vorsortiert sind. Wie wählen Sie die Zahlen aus, damit beim Einfügen in einen leeren Baum ein ausgeglichener Suchbaum entsteht? Verdeutlichen Sie die Auswahl der Zahlen am Beispiel der Zahlen von $1 \ldots 7$.

12. Aufgabe: AVL-Bäume

(a) Geben Sie die AVL-Bäume an, die nach dem Einfügen der Schlüssel 10, 15, 11, 4, 8, 7, 3, 2, 13 in einen anfangs leeren Baum entstehen. Geben Sie auch an, nach welchen Einfügeoperationen Baumrotationen notwendig sind.

(b) Formulieren Sie einen Algorithmus, der die Höhe eines AVL-Baumes berechnet (die Höhe eines Baumes ist der maximale Abstand eines Blattes von der Wurzel).

Ergänzung zu Axiomen der Mengen und Mehrfachmengen

Die in der Vorlesung angegebenen Axiome für die Opertaionen `union`, `intersect` und `diff` gelten nur für Mengen, für Mehrfachmengen unterscheiden sich die Axiome.

Die Operation `add` darf nicht auf die Operation `union` zurückgeführt werden. Die Operation `add` ist ein Konstruktor einer Menge und besitzt keine axiomatische Definition.

Die Axiome für Mengen sind wie folgt definiert:

`init` und `add` sind Konstruktoren.

```
M, N Menge ; e, f Element

create ( e ) = add ( e, init );
insert ( e, M ) = union ( create(e), M );
delete ( e, M ) = diff ( create(e), M );
member ( e, M ) = ( is_n_elem(e, M) != 0 );
is_n_elem ( e, init ) = 0;
is_n_elem ( e, add(f,M) ) = if ( f == e ) 1 + is_n_elem ( e, M )
                            else is_n_elem ( e, M );
union ( init, M ) = M;
union ( add(f,N), M ) = if ( member(f,M) ) union ( N, M );
else add ( f, union(N, M) );
intersect ( init, M ) = init;
intersect ( add(f,N), M ) = if ( member(f,M) ) add ( f, intersect(N, M) )
                            else intersect ( N, M );
diff ( init, M ) = init;
diff ( add(f,N), M ) = if ( member(f,M) ) diff ( N, M );
                       else add ( f, diff ( N, M ) );
```