

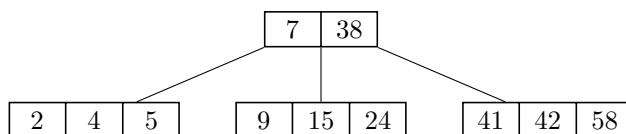
P1 (Gruppendiskussion)

Nehmen Sie sich etwas Zeit, um die folgenden Fachbegriffe in einer Kleingruppe zu besprechen, sodass sie anschließend in der Lage sind, die Begriffe dem Rest der Übungsgruppe zu erklären:

- (a) B-Bäume
- (b) Skip-Listen
- (c) Hash Tables

P2 (B-Bäume – Einfügen)

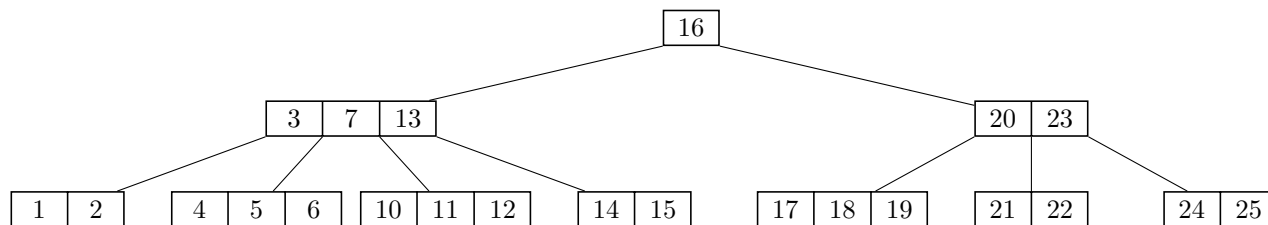
Betrachten Sie folgenden B-Baum mit Grad $t = 2$:



Fügen Sie der Reihe nach Knoten mit den Schlüsseln 66, 45, 53, 37 in diesen Baum ein. Verwenden Sie dabei die Algorithmen aus der Vorlesung. Skizzieren Sie Ihr Zwischenergebnis nach jeder Einfügeoperation.

P3 (B-Bäume – Löschen)

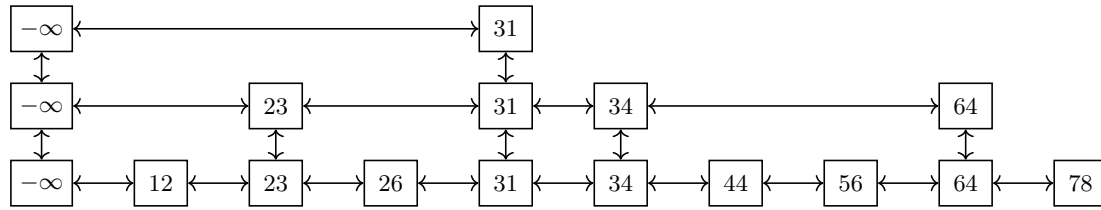
Betrachten Sie folgenden B-Baum mit Grad $t = 3$:



Löschen Sie der Reihe nach die Knoten mit den Schlüsseln 6, 13, 7, 4, 2. Verwenden Sie dabei die Algorithmen aus der Vorlesung. Skizzieren Sie Ihr Zwischenergebnis nach jeder Löschoption.

P4 (Skip-Listen)

Gegeben sei folgende Skip-Liste mit Wahrscheinlichkeit $p = 1/2$:



- (a) Wie viele Suchschritte sind notwendig, um die Elemente 78, 12, 63 in der Skip-Liste zu finden?
 (b) Fügen Sie der Reihe nach die Werte 15, 27 und 33 in die Skip-Liste ein, wenn der Zufallsgenerator fortlaufend folgende Zahlenfolge liefert:

0,34	0,58	0,87	0,49	0,12	0,26	0,69
------	------	------	------	------	------	------

- Betrachten Sie nun eine allgemeine Skip-Liste, mit n Elementen in der untersten Liste und mit Wahrscheinlichkeit p .
- (c*) Geben Sie eine asymptotische Abschätzung für die erwartete Gesamtlaufzeit von m aufeinanderfolgenden Suchvorgängen an.
- (d*) Wie viele Elemente enthält die Skipliste im Erwartungswert insgesamt? Ignorieren Sie bei Ihrer Rechnung die Sentinel-Werte.
- (e*) Wir definieren die *Höhe* einer Skip-Liste als die Anzahl der vorhandenen Expresslisten. Zeigen Sie, dass die Skip-Liste im Erwartungswert logarithmische Höhe hat.

P5 (Hash Tables)

- (a) Fügen Sie Elemente mit Schlüsseln 16, 34, 50, 7, 22, 14, 49, 33, 40, 11, 3, 2 in der angegebenen Reihenfolge in eine Hash-tabelle T der Länge 8 ein. Verwenden Sie dabei die Hashfunktion

$$h: \mathbb{N} \rightarrow \{0, \dots, 7\}, \quad k \mapsto \lfloor 8 \cdot (0,37 \cdot k \pmod{1}) \rfloor.$$

Sollte es zu Kollisionen kommen, benutzen Sie doppelt verkettete Listen, um diese aufzulösen.

- (b) Löschen Sie anschließend die Schlüssel 50 und 55 aus der Hashtabelle T . Geben Sie die resultierende Hashtabelle nach jeder Löschoperation an.

H1 (Hash-Tabellen)

(5+2+8 Punkte)

In dieser Übung sollen Sie lernen, Hash-Tabellen zu implementieren. **Bitte lesen Sie sich zunächst die allgemeinen Hinweise für die praktischen Übungen auf Moodle durch!** Laden Sie sich anschließend das vorgegebene Framework herunter, um die Aufgabe implementieren zu können. Bitte entfernen Sie vor der Abgabe alle eigenen `System.out.println`, damit Ihr Tutor die Ausgabe der Tests nicht suchen muss!

- (a) Wir verwenden Linked Lists, falls bei Hash Tables Kollisionen auftreten. Implementieren Sie die Funktionen `length`, `insertBefore`, `append` und `delete` in `LinkedList.java`.
- (b) Für die Hash-Tabelle brauchen wir natürlich auch eine Hash-Funktion. Diese rechnet zunächst die Strings in Zahlen um, indem die Summe über die Produkte der einzelnen Zeichen mit ihrer Position im String gebildet wird:

$$\sum_{i=0}^{\text{key.len}-1} (i+1) \cdot \text{key}[i].$$

Dabei soll für das Umwandeln eines Zeichens in eine Zahl der jeweilige ASCII-Wert des Zeichens genommen werden. Anschließend soll die universelle Hashfunktion aus der Vorlesung auf die Summe angewendet werden, wobei wir abweichend von der Vorlesung $p = T.\text{length}$ annehmen. Implementieren Sie die Funktion `hash` in `HashTable.java`. Generieren Sie dabei a und b zufällig im Konstruktor und bei jeder Änderung der Kapazität und speichern Sie diese in den Variablen `hash_a` und `hash_b`.

- (c) Implementieren Sie den Konstruktor von `HashTable.java` sowie die Funktionen `find`, `insert`, `delete`, `size`, `rehash` und `quality`. Beachten Sie:
- Die Kapazität sollte immer eine Primzahl sein. Wird der Konstruktor nicht mit einer Primzahl aufgerufen, soll die nächstgrößere Primzahl gewählt werden.
 - Falls `insert` mit einem Key aufgerufen wird, der in der Hashtabelle schon existiert, soll dieser existierende Eintrag durch den neuen Eintrag ersetzt werden.
 - Die Anzahl der Elemente in der Hashtabelle sollte immer kleiner sein als 75% der Kapazität. Wird dieser Wert überschritten, soll `rehash` aufgerufen werden. Diese Funktion multipliziert die aktuelle Kapazität mit 10 und rundet zur nächsten Primzahl auf. Vergessen Sie nicht, auch Ihre Parameter für die Hashfunktion anzupassen! Anschließend müssen alle Elemente neu einsortiert werden.
 - Die Methode `quality` berechnet die maximale Anzahl an Kollisionen für einen Zielwert in der Tabelle, also die Länge der längsten Linked List in der Tabelle.