Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Datenstrukturen und Algorithmen (SS 2013)

Übungsblatt 9

Abgabe: Montag, **01.07.2013**, 14:00 Uhr

- Die Übungen sollen in Gruppen von zwei bis drei Personen bearbeitet werden.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auf die abgegebenen Lösungen.
- Schreiben Sie die Namen jedes Gruppenmitglieds sowie alle Matrikelnummern auch in die Quellcode-Dateien.
- Geben Sie Ihre Lösungen am **Anfang** der Globalübung, montags, 14:00 Uhr, ab.
- Schicken Sie den jeweiligen Quellcode bitte per **E-Mail** direkt an Ihre/n Tutor/in.
- Geben Sie außerdem den ausgedruckten Quellcode zusammen mit den schriftlichen Lösungen ab.
- Zu spät abgegebene Lösungen werden nicht bewertet.
- Sofern nicht anders gefordert, müssen alle Lösungen und Zwischenschritte kommentiert werden.

Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

Informatik 8
RWTH Aachen

Aufgabe 1 (*B-Bäume* [**10 Punkte**])

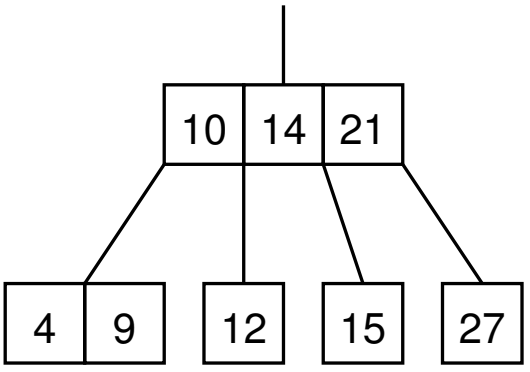
1. Fügen Sie folgende Schlüssel in dieser Reihenfolge in einen B-Baum mit minimalem Grad $t = 3$ ein.

   $$11, 19, 35, 41, 49, 25, 15, 17, 21, 7, 4, 3, 1$$

2. Löschen Sie in dieser Reihenfolge die Schlüssel

   $$14, 27, 10$$

   aus folgendem B-Baum mit minimalem Grad $t = 2$.



3. Zeigen oder widerlegen Sie: Bei gegebenen Schlüsselwerten ist die Zahl der Knoten in einem zugehörigen B-Baum eindeutig.

4. Gegeben sei eine aufsteigende, endliche Folge von ganzzahligen Schlüsselwerten mit n Elementen, also $[1, 2, \ldots, n]$. Diese Werte sollen nun in dieser Reihenfolge in einen B-Baum mit gegebenem minimalen Grad t eingefügt werden. Berechnen Sie möglichst genau, wie viele Split-Operationen beim Einfügen aller gegebenen Schlüsselwerte ausgeführt werden müssen. Für die Höhe des Baumes können Sie die Abschätzung $h \leq \log_t \frac{n+1}{2}$ benutzen.

Zeichnen Sie den Baum nach jedem Einfügen, Löschen und nach jeder Merge-, Split- oder Rotate-Operation.

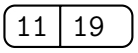
Prof. Dr. Leif Kobbelt
Thomas Ströder, Fabian Emmes
Sven Middelberg, Michael Kremer

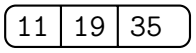
Informatik 8
RWTH Aachen

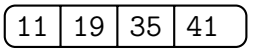
Lösungsvorschlag

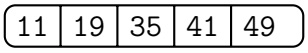
1. Der B-Baum nach jedem Einfügeschritt:

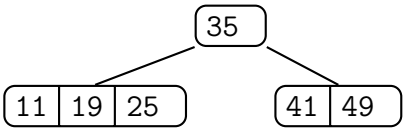
(1) 

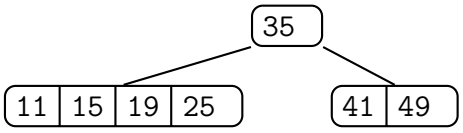
(2) 

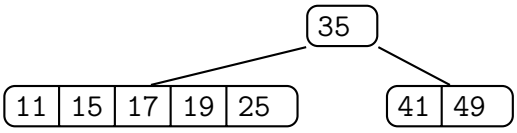
(3) 

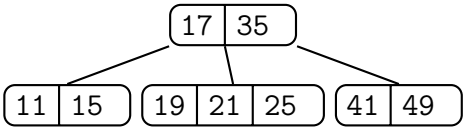
(4) 

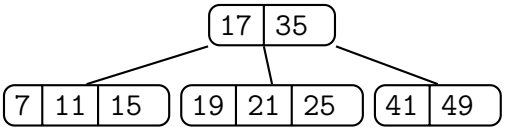
(5) 

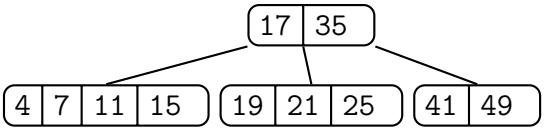
(6) 

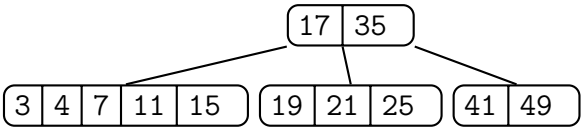
(7) 

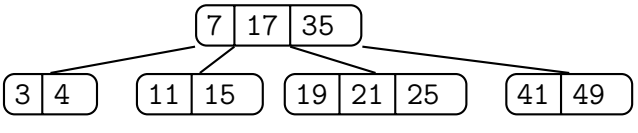
(8) 

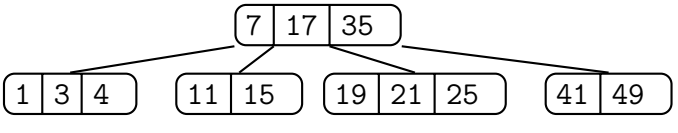
(9) 

(10) 

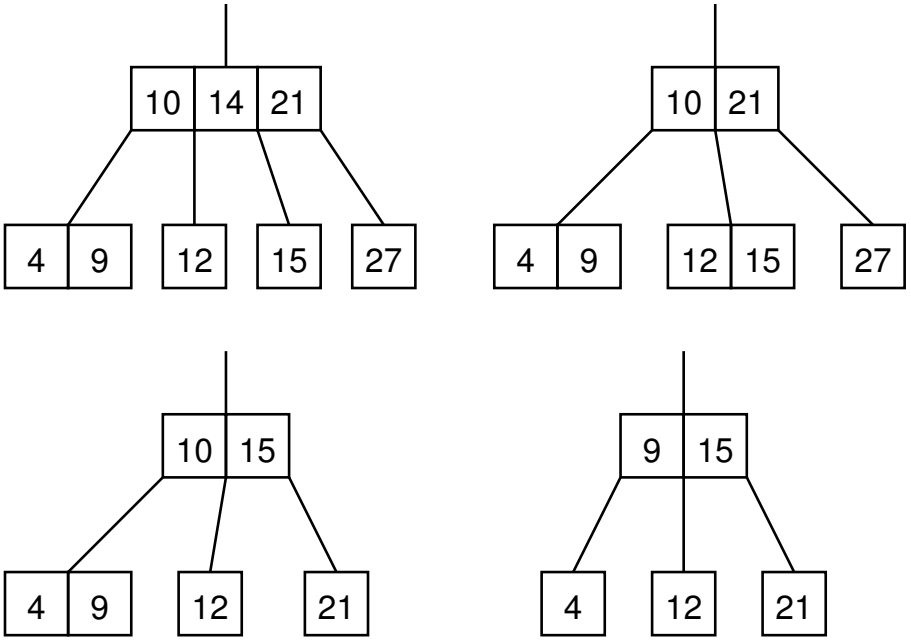
(11) 

(12) 

(13) 

(14) 

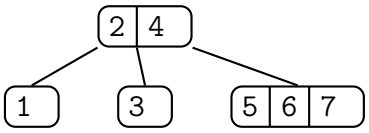
2. Der Baum jeweils nach den Löschoperationen:



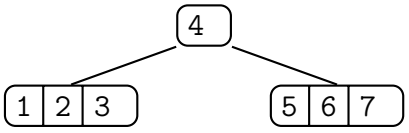
3. Die Zahl der Knoten ist bei gegebener Schlüsselmenge nicht eindeutig, da sie von der Reihenfolge, in der die Schlüssel in den B-Baum eingefügt werden, abhängig ist. Dies kann mit einem einfachen Gegenbeispiel gezeigt werden:

   Betrachte einen B-Baum mit $t = 2$ und der Schlüsselmenge $\{1, 2, 3, 4, 5, 6, 7\}$.

   - B-Baum nach Einfügen der Schlüssel in der Reihenfolge $[1, 2, 3, 4, 5, 6, 7]$.



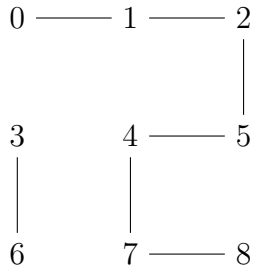
   - B-Baum nach Einfügen der Schlüssel in der Reihenfolge $[1, 4, 7, 2, 3, 5, 6]$.



Aufgabe 2 (*Union-Find* [**10 Punkte**])

In dieser Aufgabe wollen wir eine Erreichbarkeitsanalyse für Netzwerke implementieren. Das Netzwerk ist dabei gitterartig aufgebaut, d. h. jeder Knoten des Netzwerks hat höchstens vier direkte Verbindungen (nach links, unten, rechts und oben). Den Verbindungsstatus eines Knotens können wir daher als Bitfolge von vier Bits kodieren. Dabei entspricht das erste Bit dem Status der Verbindung nach links, das zweite der Verbindung nach unten, das dritte der Verbindung nach rechts und das vierte der Verbindung nach oben. Die jeweilige Verbindung ist vorhanden, wenn das

entsprechende Bit gesetzt (also gleich 1) ist. Interpretieren wir diese Bitfolgen als ganze Zahlen, können wir mit den Zahlen von 0 bis 15 alle Verbindungszustände eines Knotens beschreiben. Zum Beispiel kodiert die Zahl 7 (Bitfolge 0111) einen Knoten, der Verbindungen nach unten, rechts und oben hat, aber nicht nach links.
Wir betrachten in dieser Aufgabe nur quadratische Gitternetzwerke. Daher hat jede Zeile eines solchen Netzwerks gleich viele Knoten und die Anzahl der Knoten in einer Zeile entspricht auch der Anzahl der Zeilen im gesamten Netzwerk. Ein solches quadratisches Netzwerk können wir also nun als Array von ganzzahligen Werten aus dem Wertebereich $\{0, \ldots, 15\}$ kodieren. Dabei stellt jede Zahl den Verbindungsstatus eines Knotens dar. Die Reihenfolge der Zahlen im Array entspricht dabei der Reihenfolge der Knoten im Netzwerk, wenn man es zeilenweise durchläuft. Das Array muss also eine quadratische Länge haben und man erhält zu einer Position die rechts oder links gelegene, indem man 1 zum entsprechenden Index im Array addiert oder subtrahiert (außer an den Randpositionen). Die unten oder oben gelegene Position erhält man, indem man n addiert oder subtrahiert, wobei das Array die Länge n^2 hat (wiederum mit Ausnahme von Randpositionen). Das folgende Beispiel illustriert diese Kodierung für ein Netzwerk der Größe 3×3:



Array:
$[2, 10, 12, 4, 6, 9, 1, 3, 8]$

Die von uns bereitgestellte Datei `MainClass.java` enthält eine Klasse, deren Hauptmethode drei Argumente erwartet: die Zahl n und zwei Positionen x und y als Zahlen zwischen 0 und $n^2 - 1$. Sie generiert dann zufällig eine Netzwerk-Kodierung wie oben beschrieben (also ein Array der Länge n^2 mit Werten aus dem Wertebereich $\{0, \ldots, 15\}$) und soll anschließend `true` oder `false` ausgeben, je nach dem, ob x und y (möglicherweise über mehrere Positionen) miteinander verbunden sind. Im Beispiel oben sind die Werte 0 und 8 miteinander verbunden, während es die Werte 3 und 4 nicht sind. Die Methode `connected`, welche den zuletzt genannten Test implementieren soll, ist in der bereitgestellten Klasse allerdings nicht fertig implementiert. Stattdessen sind die Methoden `union` und `find` aus der Vorlesung bereits implementiert und sollen nun von Ihnen dazu verwendet werden, die Implementierung der `connected` Methode fertig zu stellen.
Um beispielsweise ein Netzwerk der Größe 10×10 generieren zu lassen und zu prüfen, ob darin die Positionen 0 und 99 miteinander verbunden sind, muss das Programm folgendermaßen aufgerufen werden:

```
java MainClass 10 0 99
```

Hinweis: Um zu prüfen, ob das kleinste Bit in einer Zahl `x` *gesetzt ist, kann man den Test* `(x & 1) == 1` *verwenden. Für die drei nächsten Bits funktioniert der Test analog mit den Werten 2, 4 und 8 anstelle von 1 (an beiden Positionen).*

Lösungsvorschlag

Siehe Quellcode.