Klausur Informatik B

Teil 1: Informatik 3
Juli 2005

ACHTUNG
Die vorliegende Beispielklausur wurde im Juli 2005 geschrieben. Sie umfaßt lediglich den Informatik-3-Teil der Informatik-B-Klausur und ist für eine Bearbeitungszeit von 120 Minuten gedacht. In der realen Klausur wurde dieser Teil auch getrennt vom Informatik-4-Teil geschrieben.

Dieses Muster soll den möglichen Aufbau und die Art der Fragestellung des Informatik-3-Teils der Klausur beispielhaft verdeutlichen. Wir erheben deshalb hier keinen Anspruch auf vollständige Stoffüberdeckung. In der realen Klausur können und werden also auch andere Stoffschwerpunkte geprüft werden. Weiterhin kann die Zusammensetzung ('Quickies', Textaufgaben, Hand-simulationen, Programmieraufgaben) variieren.

Diese Beispielklausur soll Ihnen helfen, sich mit der Herangehensweise zum Lösen der Aufgaben vertraut zu machen. Es nutzt Ihnen nichts, wenn Sie die Aufgaben oder Lösungen dieser Beispielklausur auswendig lernen. Prüfungsrelevant ist der in der Veranstaltung (Vorlesung und Übungen) gelehrte Stoff!

Name: *Matrikelnummer:*

Aufgabe 1: Quickies **(4 Punkte)**

Kreuzen Sie für die einzelnen Aussagen an, ob diese wahr oder falsch sind.
Hinweis: Falsche Kreuze führen zu Abzug! Insgesamt können natürlich bei dieser Aufgabe nicht weniger als 0 Punkte erreicht werden!

(a) (1 Punkt)

Jeder AVL-Baum, in dem $2^n - 1$ Knoten eingefügt wurden, ist vollständig.

wahr ◯ falsch ◯

(b) (1 Punkt)

Ein Rot/Schwarz-Baum ist immer auch ein binärer Suchbaum.

wahr ◯ falsch ◯

(c) (1 Punkt)

Die Höhe jedes binären Suchbaums mit n Knoten ist $\mathcal{O}(\log n)$.

wahr ◯ falsch ◯

(d) (1 Punkt)

Eine perfekte Skip-Liste mit n-Elementen hat Höhe $\mathcal{O}(\log n)$.

wahr ◯ falsch ◯

Punkte	

Name: *Matrikelnummer:*

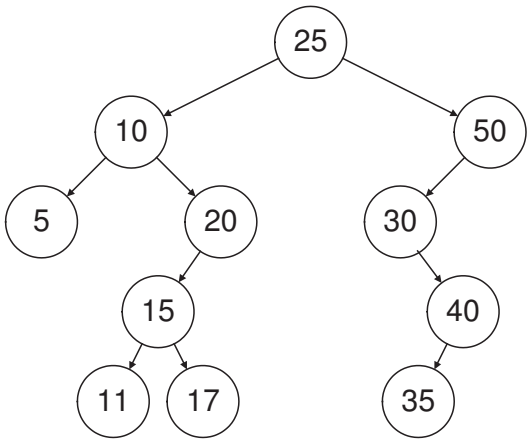
Aufgabe 2: Bäume **(9 Punkte)**

(a) (1 Punkt) Welchen Vorteil bietet die Fädelung bei der Inorder-Traversierung?

(b) (1 Punkt)

Löschen Sie die 25 aus folgendem **binären Suchbaum**.

Hinweis: Ihr Lösungsweg muss nachvollziehbar sein. Geben Sie zusätzlich zum Endergebnis auch alle Zwischenschritte an. Es reicht aus, jeweils den veränderten Teilbaum anzugeben.

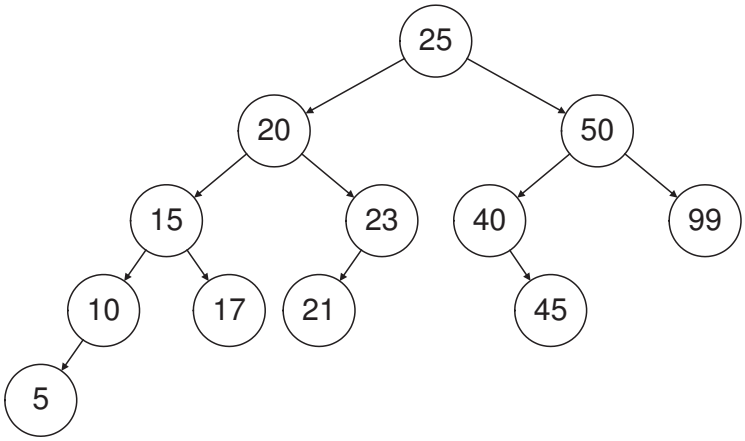


Punkte	

Name: *Matrikelnummer:*

(c) (2 Punkte) Löschen Sie die 99 aus folgendem **AVL-Baum**.

Hinweis: Ihr Lösungsweg muss nachvollziehbar sein. Geben Sie zusätzlich zum Endergebnis auch alle Zwischenschritte an, dabei reicht es aus, nur jeweils den veränderten Teilbaum anzugeben. Geben Sie außerdem die durchgeführten Rotationen an.



Punkte	

Name: *Matrikelnummer:*

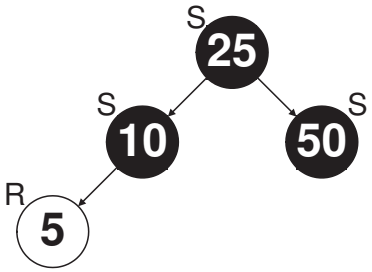
(d) (2 Punkte)

Wieviele Werte müssen in einen AVL-Baum mindestens eingefügt werden, damit Schicht 2 vollständig **sein kann**?

Wieviele Werte müssen in einen AVL-Baum mindestens eingefügt werden, damit sichergestellt ist, dass Schicht 2 vollständig **ist**?

(e) (2 Punkte) Fügen Sie 7 in den folgenden **Rot/Schwarz-Baum** ein. Rote Knoten sind mit einem R und schwarze Knoten mit einem S gekennzeichnet. Kennzeichnen Sie Ihre Knoten ebenso!

Hinweis: Ihr Lösungsweg muss nachvollziehbar sein. Geben Sie zusätzlich zum Endergebnis auch alle Zwischenschritte und die durchgeführten Rotationen und Umfärbungen an. Es reicht aus, jeweils den veränderten Teilbaum anzugeben.



Punkte	

Name: *Matrikelnummer:*

(f) (1 Punkt) Wieviele Kanten hat der kürzeste Pfad von der Wurzel zu einem Blatt in einem Rot/Schwarz-Baum der Höhe 7 mindestens?

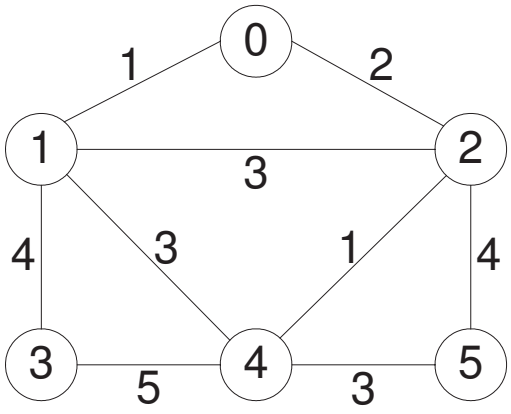
mindestens: [ ]

Punkte	

Name: *Matrikelnummer:*

Aufgabe 3: Graphen **(9 Punkte)**

(a) (2 Punkte) Gegeben sei der folgende ungerichtete, gewichtete Graph $G = (V, E)$. Benutzen Sie den Algorithmus von Prim, um einen minimalen Spannbaum für G zu konstruieren. Zeichnen Sie dazu den entstehenden Spannbaum in den gegebenen Graphen ein. Geben Sie außerdem die Reihenfolge der vom Algorithmus in den Spannbaum eingefügten Kanten an, wenn Sie mit dem Knoten 0 beginnen.



(b) (2 Punkte) Erklären Sie, warum Prims Algorithmus (zur Bestimmung des minimalen Spannbaums) korrekt ist!

Punkte	

Name: *Matrikelnummer:*

(c) (1 Punkt) Welche Datenstruktur sollte benutzt werden, um kürzeste Wege beim Dijkstra-Algorithmus zu speichern, und wie werden die kürzesten Wege darin abgelegt?

(d) (4 Punkte) Der ungerichtete und zusammenhängende Graph $G = (V, E)$ sei als Adjazenzliste gegeben. Beschreiben Sie einen **möglichst effizienten** Algorithmus, der (irgend)einen Knoten $v \in V$ bestimmt, so dass der Graph G nach Entfernen von v (und aller inzidenten Kanten) zusammenhängend bleibt. Bestimmen Sie die Laufzeit Ihres Algorithmus.

Laufzeit: $\mathcal{O}($ $)$

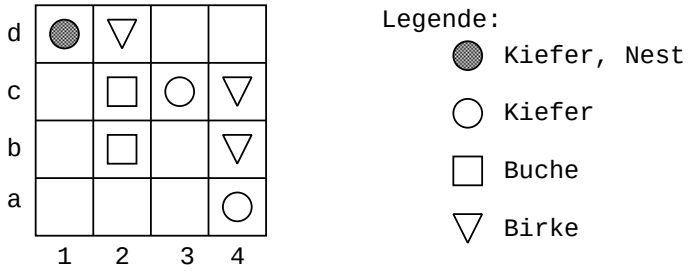
Punkte	

Name: *Matrikelnummer:*

Aufgabe 4: Eichhörnchen und Nüsse

In einem Wald lebt ein Eichhörnchen. Dieser Wald (siehe Bild) besteht aus Bäumen verschiedener Sorten: Kiefern (○), Buchen (□) und Birken (▽). Das Eichhörnchen hat auf einer Kiefer sein Nest (◍). Es kann von einem Baum zu einem anderen benachbarten Baum springen (also zu einem Baum in einem angrenzenden Feld). Die Kosten für einen Sprung hängen vom Risiko ab, dass das Eichhörnchen beim Sprung herunter fällt oder von einem Raubvogel gefangen wird. Diese Kosten sind in der folgenden Tabelle angegeben.

Wald:



Kosten für Sprünge:

Start \ Ziel →	○	□	▽
○	5	2	3
□	6	2	2
▽	6	4	3

(a) (2 Punkte) Stellen Sie das Problem (den Wald und die möglichen Sprünge) geeignet als Graph dar.

Punkte	

Name: *Matrikelnummer:*

(b) (10 Punkte) Das Eichhörnchen legt im Sommer einen Vorrat verschiedener Nüsse für den Winter an. Die Nüsse werden nach Sorten getrennt in verschiedenen Verstecken auf verschiedenen Bäumen im Wald versteckt (pro Sorte genau ein Versteck). Damit das Eichhörnchen die Nüsse im Winter wiederfindet, muss es sich die Verstecke und die jeweilige Nussart dort merken. Dazu benutzt es eine Hashtabelle und arbeitet mit Hashing mit berechneter Kollisionsfolge (*Open-Adressing*) und benutzt zur Kollisionsauflösung schlüsselabhängiges Sondieren (*double hashing*). Helfen Sie dem Eichhörnchen, und implementieren Sie Methoden zum Einfügen, Suchen und Löschen beim doppelten Hashing. Vervollständigen Sie dazu die vorgegebenen Klassenrümpfe um alle notwendigen Einträge (Variablen, Datenstrukturen und Methoden).
Hinweise:

- Als Schlüssel wird ein String mit der Nussart verwendet.
- Als Daten sollen die Nussart (String), der Ort des Verstecks und die Menge der dort versteckten Nüsse verwendet werden. Vervollständigen Sie dazu die Klasse `NussData`.
- Mit Hilfe der vorgegebenen Methode `NussData.calcKey()` können Sie aus dem String-Schlüssel einen Integer-Schlüssel berechnen, den Sie zum Hashing verwenden.
- Sie müssen auch geeignete Konstruktoren schreiben!
- Als Hashfunktionen werden $h(x) = x \bmod N$ und $a(x) = (x \bmod (N-3)) + 1$ verwendet; N ist die Größe der Hashtabelle und es gilt: $N > 3$.

Vervollständigen Sie zunächst die Klasse `NussData.java`:

```
public class NussData {
    public int calcKey() {
        // Vorgabe: ist als implementiert vorauszusetzen!!!
        // liefert zum vorliegenden Datensatz einen eindeutigen Integer
        return nussArt.hashCode();
    }

    // hier kommen alle noetigen Klassenvariablen
    public String nussArt;









    // hier kommt der Konstruktor









}
```

Punkte	

Name: *Matrikelnummer:*

Vervollständigen Sie nun die Klasse `NussHash.java` um alle benötigten Datenstrukturen/Variablen, implementieren Sie einen Konstruktor und die beiden Hashfunktionen:

```
public class NussHash {
    // hier kommen alle noetigen Klassenvariablen




    // hier kommt der Konstruktor




    // hier die beiden Hashfunktionen
```

Punkte	

Name: *Matrikelnummer:*

Implementieren Sie nun die Methode `NussHash.searchElement`:

```
// hier eine Methode zum Suchen
// Gesucht wird nach String nuss (Nuss-Sorte)
// Zurueckgeliefert wird gefundener Eintrag oder null
public NussData searchElement(String nuss) {




}
```

Punkte	

Name: *Matrikelnummer:*

Implementieren Sie nun die Methode `NussHash.insertElement`:

```
    // hier eine Methode zum Einfuegen
    // es soll ein Element vom Typ NussData eingefuegt werden
    // Returnwerte: true, falls erfolgreich, sonst false
    public boolean insertElement(NussData data) {

    }
}
```

Punkte	

Name: *Matrikelnummer:*

(c) (2 Punkte) Das Eichhörnchen hat sich leider nur die Lage der Verstecke gemerkt, aber nicht den Weg dorthin. Nehmen Sie an, dass es an jedem Baum im Wald Nüsse abgelegt hat. Nun möchte es das Gesamtrisiko, beim Sprung vom Baum herabzufallen, minimieren. D.h. es möchte jeden Baum von seinem Nest aus auf dem jeweils risikoärmsten Weg erreichen.

Welchem Ihnen aus der Vorlesung bekannten Prinzip entspricht dieses Problem? Mit welchen aus der Vorlesung bekannten Algorithmen kann es gelöst werden? Begründen Sie Ihre Antwort kurz, aber stichhaltig!

Punkte	

Name: *Matrikelnummer:*

Aufgabe 5: Restrukturierung **(10 Punkte)**

Als Mitarbeiter einer Beratungsfirma sollen Sie die internen Arbeitsabläufe einer Abteilung optimieren. Die Abteilung besteht aus n Sachbearbeitern $s \in S = \{0, 1, \ldots, n-1\}$, die auf genau n Büros $b \in B = \{0, 1, \ldots, n-1\}$ verteilt sind (jedem Sachbearbeiter ist also genau ein Büro zugeteilt). Im Zuge ihrer Arbeit müssen sich die Sachbearbeiter gegenseitig aufsuchen und dabei Akten austauschen. Diese notwendigen Besuche verursachen Kosten, die von Ihnen minimiert werden sollen. Für jedes Paar von Sachbearbeitern (i, j) sei die Zahl der Besuche bekannt als $\mathrm{f}(i, j)$. Weiterhin ist der Abstand der Büros bekannt als Betrag der Differenz der Büronummern. Die Kosten für ein Paar von Sachbearbeitern ist nun definiert als Produkt der Besuchshäufigkeit und der Entfernung ihrer Büros. Ihre Aufgabe als Berater ist es nun, die Sachbearbeiter so auf die Büros zu verteilen, dass die Gesamtkosten, also die Summe über die Kosten aller Paare von Sachbearbeitern, minimiert werden.

Beispiel für 5 Sachbearbeiter:

Sachbearbeiter	4	1	0	2	3
Büro-Nummer	0	1	2	3	4

(a) (2 Punkte) Geben Sie eine geeignete (formale) Kodierung einer Lösung für das obige Problem an, so dass es mit den aus der Vorlesung bekannten Optimierungs-Algorithmen bearbeitet werden kann.

Punkte	

Name: *Matrikelnummer:*

(b) (2 Punkte) Geben Sie auf Basis Ihrer Lösungscodierung die Kosten für einen Besuch eines Sachbearbeiters i bei einem anderen Sachbearbeiter j als Funktion $\mathrm{k}(i, j)$ an (formal).

(c) (1 Punkt) Geben Sie nun die Gesamtkosten für einen Sachbearbeiter i (also für alle Besuche, die er machen muss) als Funktion $\mathrm{k}_{\mathrm{sum}}(i)$ in Abhängigkeit von $\mathrm{k}(i, j)$ an (formal).

(d) (1 Punkt) Geben Sie die zu optimierende Zielfunktion an (formal).

(e) (1 Punkt) Geben Sie einen Elementarschritt für das Optimierungsproblem an.

Punkte	

(f) **(3 Punkte)** Positionieren Sie den ersten Sachbearbeiter ($s = 0$) möglichst günstig. Belegen Sie die restlichen Büros mit Hilfe eines Greedy-Algorithmus, so dass die Gesamtkosten in Bezug auf den Sachbearbeiter Nummer 0 optimal (so gering wie möglich) sind. Geben Sie den Algorithmus in (Java-ähnlichem) Pseudo-Code an.

Punkte	

Name: *Matrikelnummer:*

Aufgabe 6: Datenstrukturen **(4 Punkte)**

(a) (4 Punkte)

Beschreiben Sie verbal eine Datenstruktur, die die Operationen `remove(x)`, `insert(x)`, `lookup(x)` und `remove_oldest()` für ganzzahlige Schlüssel `x` unterstützt.

Die erwartete Laufzeit jeder Operation sollte so gering wie möglich sein.

Die Operation `remove_oldest()` entfernt den Schlüssel, der unter den gegenwärtig gespeicherten Schlüsseln am frühesten eingefügt wurde.

Geben Sie auch die erwartete Laufzeit im $\mathcal{O}$-Kalkül für die vier Operationen an!

Hinweis: Wählen Sie eine geeignete, in der Vorlesung behandelte Datenstruktur aus, die die Standardoperationen `remove(x)`, `insert(x)` und `lookup(x)` unterstützt. Erweitern Sie dann diese Datenstruktur so, dass auch die Operation `remove_oldest()` unterstützt wird. **Es ist keine Implementierung gefordert!** Sie sollen nur beschreiben (verbal, Skizze), welche Erweiterung Sie vornehmen und die entstehenden Laufzeiten ($\mathcal{O}$-Kalkül) begründen.

Je besser die Laufzeit der Operationen ist, desto mehr Punkte gibt es bei dieser Aufgabe!

Punkte	