Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
E. Angriman, A. Ermshaus, F. Nelles, M. Sänger

Berlin, den 28.06.2021

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 6

Abgabe: Montag, den 12.07.2021, bis 11:10 Uhr über Moodle. Die Übungsblätter sind in Gruppen von zwei (in Ausnahmefällen drei) Personen zu bearbeiten. Die Lösungen sind auf nach Aufgaben getrennten **PDFs** über Moodle abzugeben. Alle Teilaufgaben einer Aufgabe sind in einem PDF hochzuladen. Pro Gruppe genügt es, wenn eine Person die Lösung der Gruppe abgibt. Zur Bewertung wird der zuletzt hochgeladene Stand herangezogen. Vermerken Sie auf allen Abgaben Ihre **Namen**, Ihre **CMS-Benutzernamen** und Ihre **Abgabegruppe (z.B. AG123)** aus Moodle. Benennen Sie die hochgeladenen PDF-Dateien nach dem Schema *A<Aufgabe>-<Person1>-<Person2>.pdf*, bspw. A03-Musterfrau-Beispiel.pdf für Aufgabe 3 von Lisa Musterfrau und Mark Beispiel. Die Auflistung der Namen kann in beliebiger Reihenfolge erfolgen.
Beachten Sie die Informationen im Moodle-Kurs (https://hu.berlin/algodat21).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays beginnt bei 1 (und endet also bei $|A|$). Bitte beginnen Sie die Indizierung der Arrays in Ihren Lösungen auch bei 1.
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist gemeint, dass Sie eine möglichst gute obere Schranke der (Worst Case) Zeitkomplexität angeben und diese begründen sollen.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.

Aufgabe 1 (Binäre Suchbäume) — 4 + 4 + 3 + 3 = 14 Punkte

In der Vorlesung haben Sie binäre Suchbäume kennengelernt, die ganze Zahlen als Schlüssel in den Konten speichern und die Operationen Einfügen, Suchen und Löschen von Schlüsseln unterstützen. Wir betrachten hier nur Suchbäume, die keine Duplikate enthalten, d.h., alle Schlüssel in den Knoten eines Baumes sind paarweise verschieden.

1a) Zeigen Sie: Zu jedem binären Suchbaum T gibt es eine Reihenfolge seiner Schlüssel, so dass man durch Einfügen der Schlüssel in dieser Reihenfolge in einen anfangs leeren Baum den Baum T erhält. *Hinweis*: Eine geeignete Beweismethode ist die vollständige Induktion.

1b) Beweisen oder widerlegen Sie: Angenommen ein Knoten k eines binären Suchbaums hat einen symmetrischen Nachfolger, dann befindet sich der symmetrische Nachfolger immer im rechten Teilbaum von k. *Hinweis*: Im Allgemeinen ist der symmetrische Nachfolger von k der kleinste Knoten größer als k im gegebenen binären Suchbaum.

1c) Beweisen oder widerlegen Sie: Wenn man aus einem binären Suchbaum zwei beliebige Schlüssel x und y löscht, so erhält man unabhängig von der Löschreihenfolge stets den gleichen Suchbaum. Hierbei wird beim Löschen eines Knotens v mit zwei Kindern der Knoten v durch den symmetrischen Vorgänger ersetzt (vgl. Vorlesung).

1d) Angenommen Sie suchen einen Schlüssel k in einem binären Suchbaum und finden ihn schließlich. Seien P die Schlüssel auf dem zugehörigen Suchpfad ab der Wurzel und L bzw. R die Schlüssel in allen Teilbäumen, die links bzw. rechts von diesem Suchpfad liegen. Jeder Knoten in L befindet sich demnach in einem Teilbaum dessen Wurzel ein linkes Kind von einem Knoten des Suchpfades P ist. Analog ist jeder Knoten in R in einem Teilbaum dessen Wurzel ein rechtes Kind von einem Knoten des Suchpfades P ist. D. h. L, R und P sind paarweise disjunkt. Zeigen Sie, dass dann für $x \in L$, $y \in P$ und $z \in R$ nicht immer $x < y < z$ gelten muss, indem Sie ein Gegenbeispiel angeben.

Aufgabe 2 (AVL-Bäume) **7 + 7 = 14 Punkte**

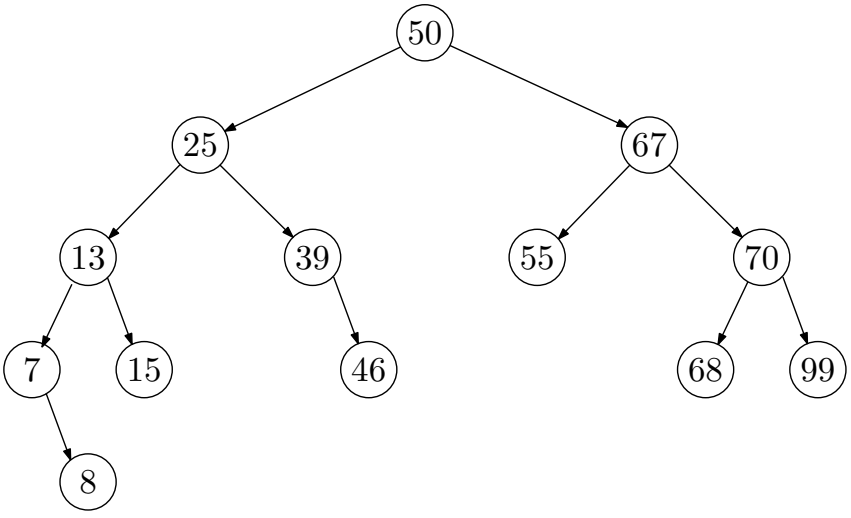
Führen Sie einen Schreibtischtest für die folgenden zwei Aufgaben zu AVL-Bäumen aus.

2a) Sei T ein leerer AVL-Baum. Fügen Sie nacheinander die Schlüssel

$$4, 21, 42, 5, 3, 64, 50, 78, 35, 47, 39, 1, 2.$$

in T ein und zeichnen Sie den jeweiligen AVL-Baum nach jeder `insert`-Operation. Geben Sie zusätzlich alle Zwischenschritte in Form von Rotationen an. Annotieren Sie in jedem Schritt jeden Knoten mit seiner Balance (in der Vorlesung definiert als bal(p)).

2b) Entfernen Sie nacheinander die Schlüssel $70, 67, 25$ aus dem unten gegebenen AVL-Baum. Zeichnen Sie den jeweiligen AVL-Baum nach jeder `delete`-Operation. Geben Sie zusätzlich alle Zwischenschritte in Form von Rotationen an. Annotieren Sie in jedem Schritt jeden Knoten mit seiner Balance. Falls ein Knoten mit zwei Kindern gelöscht wird, dann soll mit dem symmetrischen Vorgänger getauscht werden.



Aufgabe 3 (Implementierung Huffman-Kodierung) **4 + 4 + 4 = 12 Punkte**

In dieser Aufgabe sollen Sie mit Hilfe der Huffman-Kodierung einen Text kodieren und dekodieren. Die Huffmann-Kodierung ordnet einer festen Menge von Quellsymbolen jeweils Codewörter variabler Länge zu. Dazu werden zunächst die Häufigkeiten der einzelnen Symbole bestimmt und in einer Priority Queue verwaltet. Danach wird, wie in der Vorlesung kennengelernt (Folie 10ff, Priority Queues), in einem Bottom-Up Verfahren der binäre Huffman-Baum erstellt, mit dem man dann den Huffman Code für alle Symbole bestimmen kann. Betrachten Sie für die Implementierung die in Moodle bereitgestellte Vorlage `Huffman.java`. Die Klasse enthält die folgenden zwei privaten Variablen:

- Node *root*: Speichert die Wurzel des binären Huffman-Baums für die Kodierung.
- Map<Character, String> *code*: Speichert die Zuordnung eines Buchstaben zu seiner Kodierung.

Zusätzlich ist die Methode *prepare* schon implementiert, in welcher die Häufigkeiten aller Buchstaben bestimmt werden und für jeden Buchstaben ein Knoten erstellt wird, welche nach Häufigkeit geordnet in einer Priority Queue verwaltet werden. Anschließend wird mit Hilfe der Methode *createTree* der binäre Huffman-Baum erstellt und danach mittels der Methode *createCode* für jeden Buchstaben die dazugehörige Kodierung bestimmt. Die so entstandene Kodierung wird beispielhaft von der (vorgegebenen) Methode *encode* verwendet, um Strings zu kodieren.

Ergänzen Sie die Vorlage `Huffman.java` indem Sie folgende Funktionen implementieren:

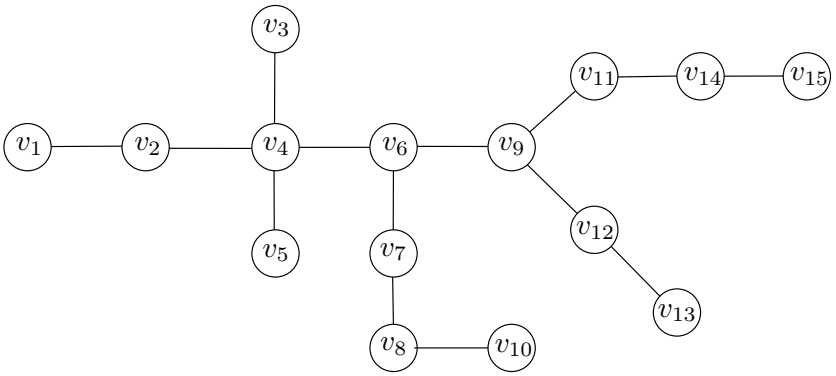
- void *createTree(*PriorityQueue<Entry> *pq)*: Erstellt den binären Baum für die Kodierung anhand der übergebenen Priority Queue und speichert die Wurzel in der privaten Variable *root*.
- void *createCode(*Node *node*, String *prefix)*: Speichert die Zuordung eines jeden Buchstaben zu seiner Kodierung in der privaten Variable *code*. Dazu wird der binäre Baum mit der aktuellen (Teil-)Kodierung traversiert. Der erstmalige Aufruf erfolgt mit der Wurzel des binären Huffman-Baums und einem leeren String.
- String *decode(*String *input)*: Dekodiert den String *input* (bestehend aus 0 und 1) anhand des binären Huffman-Baums, welcher in der privaten Variable *root* gegeben ist.

Zum Testen können Sie die main-Methode in der Vorlage `Huffman.java` verwenden. Benutzen Sie dafür die Testdateien `lorem1.txt`, `lorem2.txt` und `lorem3.txt`. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen, dürfen jedoch keine außer den von Java bereitgestellten Standard-Bibliotheken verwenden.

Hinweis: Ihr Java-Programm muss auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, sodass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Quellcodes `Huffman.java` erfolgt über Moodle.

Aufgabe 4 (Längster Pfad eines Baums) **7 + 3 = 10 Punkte**

Entwerfen Sie einen Algorithmus, der für einen beliebigen Baum mit n Knoten die Endpunkte eines längsten Pfades in diesem Baum berechnet. Für diese Aufgabe sei ein Baum ein ungerichteter, kreisfreier und zusammenhängender Graph. Beachten Sie, dass es in so einem ungerichteten Baum im Allgemeinen kein ausgezeichnetes Wurzelelement gibt. Die Länge eines Pfades ist die Anzahl der Kanten im Pfad. Betrachten Sie das dazu nachfolgende Beispiel:



In diesem Baum gibt es genau vier längste Pfade der Länge 7:

- $P_1 = v_1, v_2, v_4, v_6, v_9, v_{11}, v_{14}, v_{15}$,
- $P_2 = v_{15}, v_{14}, v_{11}, v_9, v_6, v_4, v_2, v_1$,
- $P_3 = v_{10}, v_8, v_7, v_6, v_9, v_{11}, v_{14}, v_{15}$ und
- $P_4 = v_{15}, v_{14}, v_{11}, v_9, v_6, v_7, v_8, v_{10}$.

Ihr Algorithmus sollte in diesem Beispiel also eine beliebige der folgenden Ausgaben liefern: „v_1, v_{15}“, „v_{15}, v_1“, „v_{10}, v_{15}“ oder „v_{15}, v_{10}“.

4a) Beschreiben Sie einen Algorithmus (inklusive geeigneter Datenstrukturen der Eingabedaten), der das beschriebene Problem für n Knoten in Laufzeit $\mathcal{O}(n)$ löst. Begründen Sie, warum Ihr Algorithmus diese obere Schranke für die Laufzeit einhält.

4b) Begründen oder widerlegen Sie, dass Ihr Algorithmus auch für beliebige ungerichtete, zusammenhängende (aber nicht notwendigerweise kreisfreie) Graphen stets die Endpunkte eines längsten Pfades ausgibt.