Informatik II SS 2006

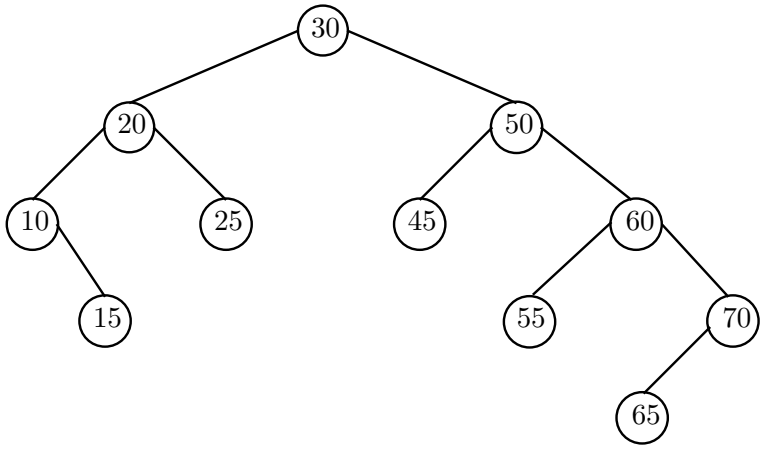
Übungsblatt 3

Abgabe: 16.5.06 - 19.5.06

Aufgabe 1 (Bäume und Traversierung) Punkte: 1+2+2

a) Was ist die minimale und die maximale Anzahl an Knoten, die ein Baum der Ordnung k und der Höhe h haben kann?
b) Sei D-B-G-E-F-C-A die Postorder- und D-B-A-E-G-C-F die Inorder-Traversierung eines Binärbaumes. Geben Sie den zugehörigen Binärbaum an.
c) Zeigen Sie anhand eines Beispiels, dass die Angabe der Preorder- und Postorder-Traversierung nicht ausreicht um einen Binärbaum eindeutig zu charakterisieren. Verwenden Sie dazu einen Baum mit möglichst wenig Knoten.

Aufgabe 2 (Suchbäume) Punkte: 5



Geben Sie eine mögliche Reihenfolge an, in der die Elemente in den zu Beginn leeren Suchbaum eingefügt worden sein könnten. In welcher Reihenfolge hätten die Elemente eingefügt werden müssen, damit ein vollständiger Baum entsteht (die Reihenfolge ist nicht eindeutig, geben Sie eine der möglichen Reihenfolgen an). Geben Sie für den rechten Teilbaum alle möglichen Einfügereihenfolgen an. Kann man für jede beliebige Menge an Elementen einen vollständigen Baum erzeugen? Falls ja, geben Sie eine Vorgehensweise hierfür an.

Aufgabe 3 (Binärbäume) Punkte: 3

Def.: Ein Baum heisst *k-gleichverzweigt* gdw jeder Knoten des Baumes entweder 0 oder k Kinder hat.
Beweisen Sie den folgenden Satz durch Induktion:
Ein 2-gleichverzweigter nicht leerer Baum mit n inneren Knoten hat $n + 1$ Blätter.

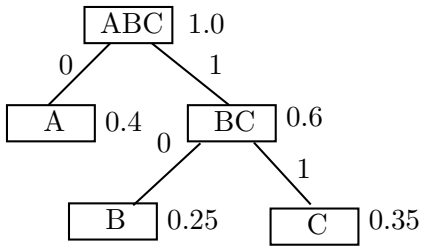
Aufgabe 4 (Binärbäume in Python) Punkte: 7

Gegeben sei ein Text, der aus den Symbolen $\mathcal{S}$ besteht. Ziel dieser Aufgabe ist es, den Text zu komprimieren, d.h. den Text so zu kodieren, dass der Code möglichst wenig Speicherplatz benötigt. Dazu wird zunächst die Häufigkeit der Symbole im Text bestimmt, d.h. wie oft jedes Symbol im Text vorkommt.
Idee: Je häufiger ein Symbol vorkommt, desto weniger Bits verwendet man für seine Kodierung. Zum Beispiel seien A,B,C die Symbole mit den relativen Häufigkeiten 0.4, 0.25, 0.35. Dann werden A mit dem Bit 0, B mit 10 und C mit 11 kodiert. (Die durchschnittlichen Speicherkosten pro Symbol betragen hier $0.4 * 1 + 0.25 * 2 + 0.35 * 2 = 1.6\ Bit$).

Algorithmus:

- Wähle die beiden Symbole mit der geringsten Häufigkeit und fasse diese zu einem neuen Symbol zusammen, die relative Häufigkeit des neuem Symbols ist die Summe der relativen Häufigkeiten der ursprünglichen Symbole
- Wiederhole Schritt 1 bis nur noch ein Symbol mit relativer Häufigkeit 1 existiert.
- Baue einen Baum auf, in dem die folgende Beziehung gilt: Symbol X und Y sind Kinder von Symbol Z gdw Symbol Z ist durch Zusammenfassen der Symbole X und Y entstanden. Ordne die Kinder so an, dass das Symbol mit der niedrigeren Häufigkeit das linke Kind ist. Die linke Kante wird mit 0 beschriftet, die rechte mit 1.

Beispiel:



Der Code eines Symbols ergibt sich aus der Konkatenation der Kantenbeschriftungen entlang des Suchpfades des Symbols. Das Verfahren nennt man Huffman-Kodierung. Anwendung findet es u.a. beim Zip-Programm.
Implementieren Sie das Verfahren in Python. Laden Sie dazu das Programm *huffman-encoding* von der Homepage runter und ersetzen Sie die Routine *calcHuffmanCode(self, symcount):* die die Berechnung des Huffman-Codes übernehmen soll. Überlegen Sie sich sinnvolle Testdateien, mit denen Sie Ihre Implementierung überprüfen können. Schauen Sie sich die komprimierten Dateien dazu mit einem Hexeditor an (z.B. khexedit).

Hinweis: Die Kantenbeschriftung können Sie in dem zugehörigen Kindknoten aufnehmen.
Das Bit der zuerst besuchten Kante wird im niedrigsten Bit der Zahl gespeichert, die einen Huffman-Code eines Zeichens repräsentiert.