Praktikumstutorium – Blatt 5

Als Vorbereitung auf das Testat 5 sollten Sie unbedingt diese Aufgaben bearbeiten.
Sofern dabei Schwierigkeiten auftreten, sollten Sie unbedingt das **Tutorium des Programmierpraktikums** besuchen

Tutoriumszeiten im Raum OH 12/4.030

Montag	**Dienstag**	**Mittwoch**	**Donnerstag**	**Freitag**
12.00 – 14.00 Uhr	10.00 – 16.00 Uhr	10.00 – 14.00 Uhr	10.00 – 14.00 Uhr	*Testat*

Methoden für binäre Suchbäume

Erweitern Sie den aus der Vorlesung bekannten binären Suchbaum, dessen Knoten Objekte der Klasse `HuffmannTriple` beinhalten. Dieser Suchbaum wird von der Klasse `CharacterSearchTree` realisiert. Die Klassen `HuffmanTriple` und `CharacterSearchTree` können Sie aus der Datei *06-Binärer-Suchbaum.zip* aus dem Ordner *Programmbeispiele* des ***moodle***-Kursbereichs herunterladen.

Beachten Sie, dass Sie Ihre Lösungen direkt in der Klasse `CharacterSearchTree` ergänzen sollen. Bei der Bearbeitung der Klausur werden Sie nur die Konstruktoren und die beiden Methoden `isEmpty()` und `isLeaf()` nutzen dürfen. Im Testat werden Ihnen zusätzlich die beiden Methoden `iterativeAdd( char t )` und `show()` zur Verfügung stehen, um Bäume zum Testen anlegen und ansehen zu können. Bei der Lösung dürfen Sie in der Klasse `CharacterSearchTree` keine zusätzlichen Attribute anlegen.

1 - Testumgebung

Erweitern Sie schrittweise die in der Klasse `Testumgebung` vorgegebene Testmethode. Die Testmethode soll die nachfolgend beschriebenen Methoden aufrufen und geeignete Ausgaben machen, um die Methoden zu überprüfen.

Ergänzen Sie die Klasse `CharacterSearchTree` um folgende Methoden:

2 - Konstruktor mit Feld von `char`

Implementieren Sie einen Konstruktor, der ein Feld von Zeichen (also des Typs `char`) als Parameter besitzt und der für die im Feld abgelegten Zeichen Knoten in den Baum in der Reihenfolge ihres Auftretens im Feld einfügt oder – falls bereits ein entsprechender Knoten vorhanden ist – die zugehörige Häufigkeit erhöht.

3 - Methode `void add( char t, int q, String c )` mit drei Parametern

Implementieren Sie eine Methode `add( char t, int q, String c )`, die ein Zeichen `t` zusammen mit der Häufigkeit `q` und der Kodierung `c` in den Baum einträgt. Ist für das Zeichen `t` im Baum noch kein Knoten vorhanden, soll dieser ergänzt werden. Ist für das Zeichen `t` im Baum bereits ein Knoten vorhanden, soll dessen Häufigkeit um den Wert von `q` erhöht und die Kodierung auf `c` gesetzt werden.

4 - Methode `void showPreOrder()`

Implementieren Sie eine Methode `showPreOrder`, die die Inhalte der Knoten des Baums in der Folge eines *PreOrder*-Durchlaufs anzeigt. Bei einem *PreOrder*-Durchlauf wird zuerst der Inhalt der Wurzel ausgegeben und danach werden der linke und danach der rechte Teilbaum in dieser Reihenfolge behandelt. Nutzen Sie die Methode `toString` der Klasse `HuffmanTriple`. Wird der Inhalt eines Blatts ausgegeben, soll ein `'*'` vorangestellt werden.

5 - Methode `int height()`

Implementieren Sie eine Methode `height`, die die Höhe des Baums liefert. Die Höhe des Baums ist die Anzahl der Knoten auf dem längsten möglichen Weg von der Wurzel zu einem Blatt. Ein leerer Baum hat die Höhe `0`. Ein Baum, der nur aus der Wurzel besteht, hat die Höhe `1`.

6 - Methode `int countCharacters()`

Implementieren Sie eine Methode `countCharacters`, die die Summe der `quantity`-Werte aller `HuffmanTriple`-Objekte im Baum bildet. Ein leerer Baum besitzt kein `HuffmanTriple`-Objekt, liefert also als Ergebnis `0`.

7 - Methode `int longestCode()`

Implementieren Sie eine Methode `longestCode`, die die Länge der längsten Kodierung aus allen `HuffmanTriple`-Objekten des Baums bestimmt. Ein leerer Baum besitzt kein `HuffmanTriple`-Objekt und damit auch keine Kodierung, liefert also als Ergebnis `0`.

8 - Methode `HuffmanTriple minimum()`

Implementieren Sie eine Methode `minimum`, die das `HuffmanTriple`-Objekt mit dem kleinsten im Baum gespeicherten `token`-Zeichen liefert. Implementieren Sie `minimum` mit einem nicht-rekursiven Algorithmus. Beachten Sie dabei, dass ein binärer Suchbaum vorliegt. Wird die Methode für den leeren Teilbaum aufgerufen, soll `null` zurückgegeben werden.

9 - Methode `boolean hasOnlyCompleteNode()`

Implementieren Sie eine Methode `hasOnlyCompleteNodes`, die prüft, ob in einem Baum keine Knoten mit nur einem Nachfolger vorkommen. In diesem Fall soll `true` zurückgegeben werden, sonst `false`. Ein leerer Baum soll `true` liefern.

10 - Methode `boolean containsCharacter( char t )`

Implementieren Sie eine Methode `containsCharacter`, die prüft, ob ein als Parameter übergebenes Zeichen im Baum als Wert des Attributs `token` eines `HuffmanTriple`-Objekts auftritt. In diesem Fall soll `true` zurückgegeben werden, sonst `false`. Ein leerer Baum soll `false` liefern.

11 - Methode `boolean equalStructure( CharacterSearchTree cst )`

Implementieren Sie eine Methode `equalStructure`, die einen Parameter des Typs `CharacterSearchTree` besitzt. Die Methode soll `true` zurückgeben, falls der aufrufende Baum und der als Argument übergebene Baum die gleiche Struktur besitzen, also an den gleichen Positionen in den Bäumen Knoten bzw. Nachfolger auftreten. Die Inhalte der Knoten sollen dabei unberücksichtigt bleiben.

12 - Methode `CharacterSearchTree rotateNodeToRight()`

Implementieren Sie eine Methode `rotateNodeToRight`, die eine Rückgabe vom Typ `CharacterSearchTree` liefert. Die Methode soll den Baum derart umformen, dass das linke Kind der Wurzel zur (neuen) Wurzel wird. Die Teilbäume der betroffenen Knoten sollen unverändert bleiben. Die neue Wurzel soll zurückgegeben werden. Ist der Baum leer oder besitzt die Wurzel kein linkes Kind, soll nichts geschehen und die (alte) Wurzel zurückgegeben werden.

Hinweis: Beachten Sie, dass das folgende Beispiel nur einen Sonderfall der Rotation behandelt.

Beispiel: