Aufgaben zur Vorbereitung auf die Testate 5 und 6

Als Vorbereitung auf die Testate 5 und 6 solltest Du unbedingt diese Aufgaben bearbeiten.

Methoden für binäre Suchbäume

Erweitere den aus der Vorlesung bekannten binären Suchbaum, dessen Knoten Objekte der Klasse `HuffmannTriple` beinhalten. Dieser Suchbaum wird von der Klasse `CharacterSearchTree` realisiert.

Beachte, dass die Lösungen direkt die Klasse `CharacterSearchTree` ergänzen sollen. Bei der Bearbeitung der Klausur wirst Du nur die Konstruktoren und die beiden Methoden `isEmpty()` und `isLeaf()` nutzen dürfen. Im Testat werden Dir 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 in der Klasse `CharacterSearchTree` keine zusätzlichen Attribute angelegt werden.

Erweitere schrittweise auch die in der Klasse `TreeTest` vorgegebene Testmethode. Die Testmethode soll die nachfolgend beschriebenen Methoden aufrufen und geeignete Ausgaben machen, um die korrekte Implementierung der Methoden zu überprüfen.

Die Klassen `HuffmanTriple`, `CharacterSearchTree` (in der für das Testat eingeschränkten Fassung) und `TreeTest` sind zusammen mit diesem Praktikumsblatt heruntergeladen worden.

Ergänze die Klasse `CharacterSearchTree` um folgende Methoden:

1 - Konstruktor mit Feld von `char`

Implementiere 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.

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

Implementiere 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.

3 - Methode `void showPreOrder()`

Implementiere 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. Nutze die Methode `toString` der Klasse `HuffmanTriple`. Wird der Inhalt eines Blatts ausgegeben, soll ein `'*'` vorangestellt werden.

4 - Methode `int height()`

Implementiere 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`.

5 - Methode `int countCharacters()`

Implementiere 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`.

6 - Methode `int longestCode()`

Implementiere 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`.

7 - Methode `HuffmanTriple minimum()`

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

8 - Methode `boolean hasOnlyCompleteNode()`

Implementiere 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.

9 - Methode `boolean containsCharacter( char t )`

Implementiere 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.

10 - Methode `boolean equalStructure( CharacterSearchTree cst )`

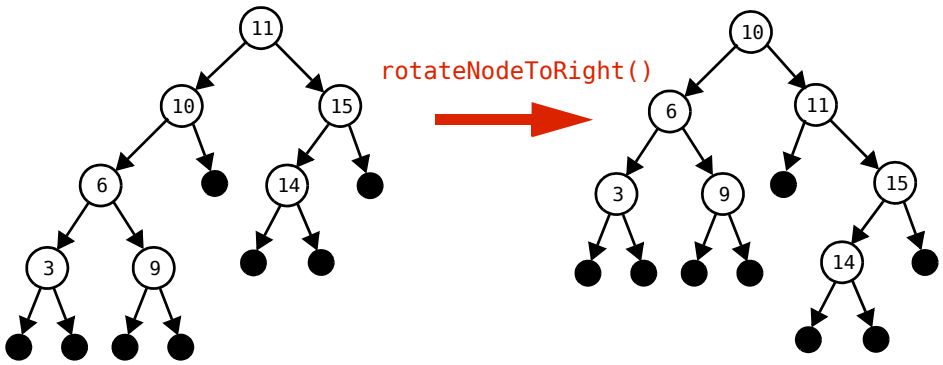
Implementiere eine Methode `equalStructure`, die einen Parameter des Typs `CharacterSearchTree` besitzt. Die Methode soll `true` zurückgeben, falls der aufrufende Baum und der als Argument übergebene Baum die gleiche Struktur besitzen, also an den gleichen Positionen in den Bäumen Knoten bzw. Nachfolger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

11 - Methode `CharacterSearchTree rotateNodeToRight()`

Implementiere 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: Beachte, dass das folgende Beispiel nur einen Sonderfall der Rotation behandelt.

Beispiel:



12 - Methode `boolean samePath( char t1, char t2 )`

Implementiere eine Methode `boolean samePath( char t1, char t2 )`, die `true` liefern soll, wenn es einen direkten Pfad zwischen der Wurzel und einem (beliebigen) Blatt gibt, auf dem (in beliebiger Reihenfolge) ein `HuffmanTriple`-Objekt dem dem Zeichen `t1` als Attributwert von `token` und ein `HuffmanTriple`-Objekt dem dem Zeichen `t2` als Attributwert von `token` liegen. Sonst soll `false` zurückgegeben werden.