TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK
Lehrstuhl für Angewandte Informatik / Kooperative Systeme
Prof. Dr. A. Brüggemann-Klein, R. Palenta, J. Kranz, Dr. A. Reuss

WS 15/16
Übungsblatt 9
11.12.2015

Praktikum Grundlagen der Programmierung

Abgabe: 21.12.2015 (vor 5:00 Uhr)

Hinweis: Implementieren Sie alle Hausaufgaben in einem Package namens **blatt09**.

Aufgabe 9.1 (P) Doppelt verkettete Liste

Themengebiet: Dynamische Datenstrukturen, Rekursion

In dieser Aufgabe soll eine doppelt verkettete Liste erstellt werden. Eine doppelt verkettete Liste unterscheidet sich von einer einfach verketteten Liste nur dadurch, dass jedes Listenelement zusätzlich eine Referenz auf das vorherige Listenelement hat.

Erstellen Sie eine generische Klasse `Element`, die ein Listenelement repräsentiert. Ein Objekt der Klasse `Element` hat folgende drei Attribute

- `private Element<T> prev` - die Referenz auf das vorherige Listenelement
- `private Element<T> next` - die Referenz auf das nachfolgende Listenelement
- `T info` - den eigentlichen Wert des Listenelements, wobei T den generischen Datentyp repräsentiert

Erstellen Sie die folgenden Methoden

- get- und set-Methoden für `prev`, `next` und `info`.
- einen Konstruktor `public Element(T info, Element prev, Element next)`, der ein Listenelement mit dem Wert `info`, mit einer Referenz auf das vorherige Listenelement `prev` und mit einer Referenz auf das nachfolgende Listenelement `next` erstellt.
- `public boolean search(T info)` - es wird *rekursiv* nach einem Listenelement mit dem Wert `info` gesucht. Wird ein solches Listenelement gefunden, wird `true` zurückgegeben, sonst `false`.

Erstellen Sie eine generische Klasse `DoublyLinkedList`, die eine doppelt verkettet Liste repräsentiert. Ein Objekt der Klasse `DoublyLinkedList` hat folgende Attribute

- `Element<T> head` - die Referenz auf das erste Listenelement
- `int size` - die Größe der Liste

Erstellen Sie die folgenden Methoden

- einen Konstruktor `public DoublyLinkedList()`, der eine leere Liste erzeugt.
- `public void add(T info, int pos)` - es wird ein Listenelement mit dem Wert `info` an der Position `pos` hinzugefügt. Ist `pos` 0 wird das neue Element vor die Liste gehangen, ist `pos` die Größe der Liste, wird das neue Element am Ende der Liste angehangen. Ist die Position ungültig wird kein Element hinzugefügt.

- `public int size()` - gibt die Größe der Liste zurück.
- `public T getPosition(int pos)` - gibt den Wert des Listenelements an Position `pos` zurück.
- `public void remove(int pos)` - entfernt das Listenelement an der Position `pos`. Ist die übergebene Position ungültig wird kein Listenelement entfernt.
- `public String toString()` - gibt die Listenelemente durch Kommata getrennt in eckigen Klammern aus, z.B. [1, 2, 3] oder [eins, zwei].

Testen Sie die erstellten Klassen!

Aufgabe 9.2 (P) Mergesort
Themengebiet: Rekursion

In dieser Aufgabe soll der Sortieralgorithmus Mergesort implementiert werden. Der Algorithmus basiert auf dem Verfahren *Teile und Herrsche*. Das bedeutet, dass ein Problem immer wieder rekursiv in kleinere/einfachere Teilprobleme zerlegt wird, bis es auf den einzelnen Teilen gelöst (beherrscht) werden kann. Anschließend wird aus den gelösten Teilproblemen eine Gesamtlösung zusammengestzt.

Bei Mergesort wird eine Liste von Daten solange rekursiv halbiert bis die resultierenden Listen nur noch ein Datum enthalten. Beim rekursiven Aufstieg werden die kleinen Listen sortiert zusammengefügt (merge). Es werden immer zwei bereits sortierte Listen (einelementige Listen sind per se sortiert) zu einer sortierten Liste zusammengefügt, bis wieder alle Daten in einer Liste liegen.

Implementieren Sie den Sortieralgorithmus Mergesort für ein Array von Integers. Verwenden Sie dazu die beiden folgenden Methoden

- `public static void mergesort(int[] numbers)` - Implementierung des rekursiven Mergesort-Algorithmus.
- `public static void merge(int[] result, int[] left, int[] right)` - fügt die beiden sortierten Arrays `left` und `right` in `result` sortiert zusammen.

Tipps:

- Spielen Sie den Algorithmus auf dem Papier an kleinen Beispielen durch.
- Überlegen Sie, was die Abbruchbedingung für die Rekursion ist.
- In wie viele Teilprobleme wird das Sortierproblem in einem Rekursionsschritt zerlegt?
- Die Methode `merge` kann unabhängig vom restlichen Algorithmus implementiert werden. Testen Sie die Implementierung dieser Methode bevor Sie die Methode im Algorithmus verwenden.

Aufgabe 9.3 (P) Tic Tac Toe
Themengebiet: Arrays

In dieser Aufgabe wollen wir das Spiel *Tic Tac Toe* implementieren. Es spielen 2 Spieler auf einem 3×3 Spielfeld gegeneinander. Abwechselnd setzen die Spieler ihre Zeichen. Für die Implementierung nehmen wir an, dass Spieler 1 seine Felder mit einer 1 markiert und Spieler 2 seine Felder mit der Zahl 2 markiert.

Damit die Spieler angeben können, auf welches Feld sie setzen wollen, gehen Sie davon aus, dass die Felder des Spielfelds folgendermaßen durchnummeriert sind:

0	1	2
3	4	5
6	7	8

Der Spieler, der als erstes drei seiner Zeichen in einer Reihe, einer Spalte oder Diagonale setzen kann, gewinnt dieses Spiel.
Berücksichtigen Sie, dass das Spiel auch unentschieden ausgehen kann, wie bei folgender Situation:

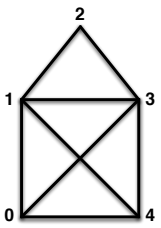
1	1	2
2	2	1
1	2	1

Hinweis: Überlegen Sie sich alternative Möglichkeiten, die Felder des Spielfelds zu kodieren. Welche Teile des Programms könnten dadurch leichter umgesetzt werden, welche Teile würden schwieriger werden?

Aufgabe 9.4 [4 Punkte] **(H) Arrays**
Implementieren Sie in einer Klasse `Arrays` die folgenden Methoden:

- [1] `public static void minUndMax(int[] feld)` - gibt Minimum und Maximum des Felds `feld` auf der Konsole aus. Das Feld wird dabei nur einmal durchlaufen.
- [1] `public static int[] invertieren(int[] feld)` - gibt ein neues Feld zurück, das die Elemente von `feld` in umgekehrter Reihenfolge enthält.
- [1] `public static int[] schneiden(int[] feld, int laenge)` - gibt ein neues Feld zurück, dass `laenge` Zeichen lang ist und die Elemente von `feld` soweit möglich enthält. Sollte das zurückgegeben Feld größer sein als das übergebene, sollen die zusätzlichen Positionen den Wert 0 haben.
- [1] `public static int[] linearisieren(int[][] feld)` - gibt ein neues eindimensionales Feld zurück, das die Werte des übergebenen zweidimensionalen Feld `feld` enthält, zurück. Die Zeilen des Felds `feld` sollen dabei in der ihrem Index entsprechenden Reihenfolge in dem eindimensionalem Feld abgelegt werden. Gehen Sie davon aus, dass das mehrdimensionale Feld für jede Zeile ein Feld der Spaltenwerte speichert. Beachten Sie, dass Zeilen nicht gleich lang sein müssen.

Aufgabe 9.5 [5 Punkte] **(H) Das Haus vom Nikolaus**
Themen: Rekursion



In dieser Aufgabe soll rekursiv die Anzahl der Möglichkeiten bestimmt werden, das Haus vom Nikolaus zu zeichnen. Dabei kann mit dem Zeichnen des Hauses an jeder beliebigen Stelle begonnen werden, jede Linie darf jedoch nur genau einmal gezeichnet werden.

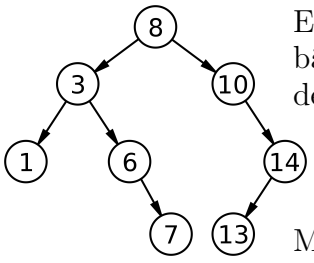
Mithilfe eines zweidimensionalen Arrays `int[][] edges` kann man sich leicht merken, welche der Punkte 0 bis 4 durch eine Linie verbunden sind. Wenn gilt `edges[i][j] != 0` bzw. `edges[j][i] != 0`, muss eine Linie zwischen den Punkten `i` und `j` gezeichnet werden, sonst nicht. Der folgende Algorithmus versucht, das Haus vom Nikolaus abzulaufen und dabei jede abgelaufene Linie in `edges` auf 0 zu setzen.

- Laden Sie die Datei `Nikolaus.java` von der Praktikums-Webseite herunter und fügen Sie diese zu einem neuen, leeren Java-Projekt „Nikolaus“ hinzu.
- [3] Implementieren Sie die vorgegebene Methode
  `public static int countSolutions(int pos, int linesLeft)`,
  die als Parameter `pos` die Nummer des Punktes übergeben bekommt, an dem man sich beim Ablaufen gerade befindet sowie den Parameter `linesLeft`, der angibt, wieviele Linien noch nicht abgelaufen sind.
  Hinweis:
  - Überlegen Sie zunächst, was es bedeutet, wenn keine Linie mehr abzulaufen ist und geben Sie ein entsprechendes Ergebnis zurück.
  - Falls noch Linien abzulaufen sind und Sie am Punkt `i` angekommen sind, finden Sie heraus, welche Linien vom Punkt `i` aus noch abzulaufen sind. Gibt es keine solche Linie mehr, geben Sie ein entsprechendes Ergebnis zurück.
    Für jede mögliche Linie fahren Sie rekursiv mit dem Aufsummieren der Möglichkeiten fort. Denken Sie daran, *vor* dem rekursiven Aufruf die Linie durch `edges[i][j] = edges[j][i] = 0` als bereits abgelaufen zu markieren und *nach* dem Aufruf die Markierung wieder zu entfernen.
- [2] Implementieren Sie nun die vorgegebene Methode
  `public static void main(String[] args)`.
  Zählen Sie für jeden der Punkte, wieviele Möglichkeiten es gibt, von diesem Punkt aus das Haus zu zeichnen und geben Sie anschliessend die Summe aus.

Aufgabe 9.6 [10 Punkte] (H) Binärbaum

Themen: dynamische Datenstrukturen, Rekursion

- Dynamische Datenstrukturen
- Rekursion



Ein *Binärbaum* ist ein Baum, in dem alle Knoten einen Wert und 2 Teilbäume haben. Bei einem *Binären Suchbaum* (BSB) gilt für jeden Knoten des Baumes, dass sein Wert

- größer ist als die Werte aller Knoten in seinem linken Teilbaum;
- kleiner ist als die Werte aller Knoten in seinem rechten Teilbaum;
- maximal einmal im ganzen BSB vorkommt.

Man bezeichnet dies auch als *Bauminvariante*.

***Beispiel**: In der obigen Zeichnung finden Sie einen BSB für die Zahlen* $1, 3, 6, 7, 8, 10, 13, 14$.

a) [1] Entwerfen und realisieren Sie eine Datenstruktur `BinaryTree` für binäre Suchbäume für ganze Zahlen (`int`). Auf die Objektattribute soll von außen nicht zugegriffen

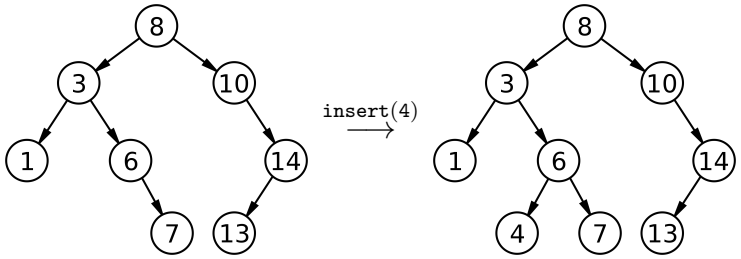
werden können. Die einzelnen Elemente des Baumes sollen in Objekten der Klasse `BinaryTreeNode` gespeichert werden. Die Klasse `BinaryTree` verwaltet den Baum und hält dazu eine Referenz auf den Wurzelknoten des Baumes.

b) [1] Implementieren Sie Konstruktoren für die Klasse der Baumknoten `BinaryTreeNode`. Ein Baumknoten soll entweder alleinstehend aus einer einzelnen Zahl gebaut werden oder zusätzlich einen rechten und linken Teilbaum erhalten. Implementieren Sie außerdem einen Kontruktor der Klasse `BinaryTree`, der einen leeren Binärbaum erstellt.

Hinweis: Sie können zum Testen zusätzlich einen Konstruktor der Klasse `BinaryTree` implementieren, der manuell zusammengesetzte Baumknoten als Parameter übernimmt. Einen solchen Konstruktor sollte die fertige Datenstruktur allerdings nicht enthalten, da so fehlerhafte Binärbäume konstruiert werden können.

Implementieren Sie nun die folgenden Methoden in den Klassen `BinaryTree` und `BinaryTreeNode`

c) [1] Implementieren Sie eine Methode `void insert(int newElem)`, die ein Element in den Binärbaum einfügt. Achten Sie dabei auf die Einhaltung der Bauminvariante. ***Beispiel**: Die folgende Zeichnung illustriert einen Aufruf von* `insert(4)`:



d) [1] Implementieren Sie eine Methode `boolean contains(int elem)`, die `true` zurückgibt, falls das übergebene Element im Baum vorkommt und `false`, falls nicht.

e) [2] Implementieren Sie eine Methode `String toString()`, die den Baum als Text ausgibt. Die Methode muss in der Klasse `BinaryTreeNode` rekursiv implementiert werden. Es soll hierfür das sogenannte *DOT*-Format[1] verwendet werden. In diesem Format beginnt die Ausgabe stets mit der Zeile „`digraph g {`“. Es folgt die Zeile „`graph [ordering=out]`“, die die Reihenfolge der Kanten festlegt. Nach dieser Zeile gibt man die Kanten im Baum aus, wobei eine Kante pro Zeile ausgegeben wird. Man muss darauf achten, die Kante zum linken Teilbaum vor der Kante zum rechten Teilbaum auszugeben. Eine Kante besteht aus den beiden Knotenbezeichnern (hier einfach ihr gespeicherter Wert) und einem verbindenden Pfeil. Hinter der Kante gibt man noch einen Namen der Kante an, sodass die Richtung klar wird. So ergibt sich z.B. „`5 -> 7 [label=right]`“ für eine Rechtskante vom Knoten 5 zum Knoten 7. Nach allen Kanten folgt noch eine abschließende Zeile, die lediglich eine schließende geschleifte Klammer enthält. Als Beispiel ergibt sich folgende DOT-Repräsentation des im Einführungsbeispiel gezeigten Binärbaumes:

[1] `http://www.graphviz.org/doc/info/lang.html`

```
digraph g {
  graph [odering=out]
  8 -> 3 [label=left]
  8 -> 10 [label=right]
  10 -> 14 [label=right]
  14 -> 13 [label=left]
  3 -> 1 [label=left]
  3 -> 6 [label=right]
  6 -> 7 [label=right]
}
```

Es gibt Software, die aus Graphen im DOT-Format automatisch ein Bild erzeugt, sodass man den Graphen betrachten kann. Dies ist sehr praktisch zum Debugging. Ohne Installation erlaubt es die Webseite `http://sandbox.kidstrythisathome.com/erdos/`, DOT-Graphen zu zeichnen.

Hinweis: Das DOT-Format bietet noch viele weitere Möglichkeiten, mit denen sich auch die hier verwendeten Binärbäume schöner darstellen lassen. Eigene Verbesserungen an der Darstellung sind ausdrücklich erwünscht!

f) [1] Implementieren Sie eine Methode `int size()`, die die Anzahl der Elemente des Baums zurückgibt.

g) [1] Implementieren Sie eine Methode `int depth()`, die die Tiefe des Baums zurückgibt, sprich die Anzahl der Kanten des längsten Pfades von der Wurzel bis zu einem Blatt (Blätter sind kinderlose Knoten). Ein einelementiger Baum hat also die Tiefe 0.

h) [1] Implementieren Sie eine Methode `String toStringDescending()`, die die Elemente des Baums in absteigender Reihenfolge durch Kommata getrennt als Zeichenkette zurückgibt.

i) [1] Implementieren Sie eine Methode `boolean isBinarySearchTree()`, die `true` zurückgibt, falls die Bauminvariante für diesen Baum gilt und `false`, falls nicht. Wenn Sie alles richtig implementiert haben, sollte diese Methode stets `true` liefern; sie eignet sich also dazu, ihre Implementierung zu testen.

Hinweis: Wir empfehlen, alle Methoden rekursiv zu implementieren. Um die volle Punktzahl zu erhalten, muss jedoch lediglich die Methode `toString()` rekursiv implementiert werden.

Beispiel: Um eine Methode `String toStringAscending()` zu implementieren, die die Elemente des Baumes in aufsteigender Reihenfolge zurückgibt, müssen an einen String – ausgehend von einem Baumknoten – zuerst die Werte der Baumknoten, die kleiner dem Wert des aktuellen Baumknoten sind, anschließend der Wert des aktuellen Baumknotens und zuletzt die Werte der Baumknoten, die größer als der Wert des aktuellen Baumknotens sind, angehängt werden. Aufgrund der Bauminvariante sind alle Baumknoten, deren Werte kleiner sind als der Wert des aktuellen Knotens, im linken Teilbaum und alle Baumknoten, deren Werte größer als der Wert des aktuellen Knotens sind, im rechten Teilbaum. Damit ergibt sich folgende Methode für die Klasse der Baumknoten, wobei `left` die Referenz auf das linke Kind, `right` die Referenz auf das rechte Kind ist und `value` der Wert des Baumknoten ist.

```
public String toStringAscending () {
    String result = "";
    if ( left != null ) {
        result = result + left.toStringAscending () ;
        result = result + ", " ;
    }
    result = result + value ;
    if ( right != null ) {
        result = result + ", " + right.toStringAscending () ;
    }
    return result ;
}
```