TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Grundlagen: Algorithmen und Datenstrukturen
Prof. Dr. Helmut Seidl, J. Kranz, A. Reuss, R. Vogler

SS 2016
Übungsblatt 1
17.04.2016

Abgabe: 24.04.2016 (bis 23:59 Uhr)

Aufgabe 1.1 (P) Tiefe Bäume

Gegeben sei folgende induktive Definition der Datenstruktur *Baum*: Ein Baum ist

- entweder ein als *Blatt* bezeichnetes Objekt
- oder ein innerer *Knoten* zusammen mit einer Liste von (Unter-)Bäumen.

Sei $k \geq 1$. Ein Baum heißt k-verzweigt, wenn

- er ein Blatt ist oder
- er k viele k-verzweigte Unterbäume hat.

In k-verzweigten Bäumen hat also jeder innere Knoten gleich viele *Kinder*.

Wir bezeichnen einen Baum als *vollständig*, wenn

- er ein Blatt ist oder
- die Unterbäume alle gleich und ihrerseits vollständig sind.

Die *Tiefe* eines Baumes definieren wir als

- 1, wenn er ein Blatt ist, und
- 1+ die maximale Tiefe eines Unterbaumes sonst.

Die Anzahl Knoten in einem Baum ist

- 1, wenn er ein Blatt ist, und
- 1+ die Summe der Knoten der Unterbäume sonst.

Beweisen Sie folgende Behauptung mit Hilfe von Induktion **nach der Anzahl Knoten** n: Die Tiefe eines vollständigen 2-verzweigten Baumes mit n Knoten ist höchstens $log_2(n+1)$.

Hinweis zur Definition der Vollständigkeit: Wir bezeichnen zwei Bäume als *gleich*, wenn

- beide Blätter sind oder
- beide die gleichen Unterbäume haben (in der selben Reihenfolge).

Aufgabe 1.2 (P) Bubblesort mit Listen

Gegeben sei folgender Java-Code, der eine einfach verkettete Liste implementiert:

```
public class List {
  private static class Node {
    private int data;

    public int getData () {
      return data;
    }

    public void setData (int data) {
      this.data = data;
```

```
  }

  private Node next;

  public Node getNext () {
    return next;
  }

  public Node (int data, Node next) {
    this.data = data;
    this.next = next;
  }
}

private Node head;

private int size;

public int getSize () {
  return size;
}

public List () {
}

void prepend (int data) {
  head = new Node(data, head);
  size++;
}

int get (int index) {
  Node it = head;
  while (index != 0) {
    index--;
    it = it.getNext();
    if (it == null)
      throw new RuntimeException("Out of bounds");
  }
  return it.getData();
}

void swap (int indexFirst, int indexSecond) {
  if (indexFirst > indexSecond) {
    swap(indexSecond, indexFirst);
    return;
  }
  int distance = indexSecond - indexFirst;
  if (head == null)
    throw new RuntimeException("Out of bounds");
  Node it_first = head;
  while (indexFirst != 0) {
    indexFirst--;
    it_first = it_first.getNext();
    if (it_first == null)
      throw new RuntimeException("Out of bounds");
  }
  Node it_second = it_first;
  while (distance != 0) {
    distance--;
    it_second = it_second.getNext();
    if (it_second == null)
      throw new RuntimeException("Out of bounds");
  }
  int temp = it_second.getData();
```

```
    it_second.setData(it_first.getData());
    it_first.setData(temp);
  }

  @Override public String toString () {
    StringBuilder sb = new StringBuilder();
    sb.append("[");
    boolean first = true;
    Node it = head;
    while (it != null) {
      if(first)
        first = false;
      else
        sb.append(", ");
      sb.append(it.getData());
      it = it.getNext();
    }
    sb.append("]");
    return sb.toString();
  }
}
```

a) Betrachten Sie die Implementierung und beantworten Sie dabei folgende Fragen:

- Was hat es mit der Schachtelung der Klassen auf sich?
- Was bedeutet es, dass die innere Klasse statisch ist? Wieso ist dies hier sinnvoll?
- Wozu wurde der Wrapper `List` implementiert, statt den Benutzer direkt auf Knoten arbeiten zu lassen? Was hat dies mit *abstrakten Datentypen* zu tun?
- Welchen Zweck erfüllt `throw`? Wieso wird eine `RuntimeException` statt einer normalen `Exception` geworfen? Wo liegt der Unterschied?
- Welche der Operationen sind langsam, welche schnell? Wieso?
- Die `swap`-Methode ruft sich selbst auf. Wie nennt man Methoden, die sich so verhalten? Welche Motivationen gibt es, derart zu programmieren?
- Wieso wird in `toString()` ein `StringBuilder` verwendet? Welche Operation implementiert dieser effizient?

b) Die Liste soll nun genutzt werden, um Daten sortiert zu speichern. Um die Liste zu sortieren, wird folgende Methode verwendet, die den *Bubblesort*-Algorithmus implementiert:

```
void bubblesort(List l) {
  for(int i = 0; i < l.getSize(); i++)
    for(int j = 1; j < l.getSize() - i; j++)
      if(l.get(j - 1) > l.get(j))
        l.swap(j - 1, j);
}
```

Beantworten Sie folgende Fragen:

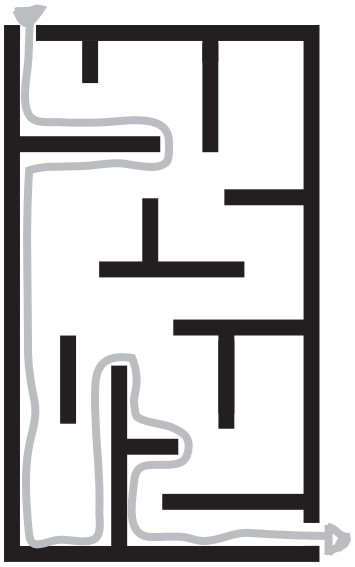
- Die Methode hat keinen Rückgabewert; funktioniert sie dennoch? Wenn ja, wieso? Rufen Sie sich allgemein das Konzept von *Referenzen* ins Gedächtnis.
- Wie funktioniert der Algorithmus, warum liefert er stets ein sortiertes Ergebnis?
- Wie oft *ungefähr* wird eine Liste der Größe n durchlaufen? Beantworten Sie diese Frage zunächst, ohne dabei an die Laufzeit der aufgerufenen Listenmethoden zu denken.
- Eignet sich unsere Implementierung einer Liste hier besonders gut oder besonders schlecht? Ziehen Sie nun die Implementierung der Listenmethoden in Ihre Laufzeitüberlegung mit ein. Zu welch verheerendem Ergebnis kommen Sie?

Allgemeine Hinweise zur Hausaufgabenabgabe

Die Hausaufgabenabgabe erfolgt ausschließlich über Moodle. Bitte vergewissern Sie sich nach der Abgabe selbstständig, dass alle Dateien erfolgreich auf Moodle eingestellt wurden. Programmieraufgaben sind in Java zu lösen. Quelltexte sind als UTF-8 kodierte `.java`-Textdateien abzugeben; die Dateien dürfen sich in einem Archiv im Zip-Format befinden. Schriftliche Abgaben müssen in Form einer einzigen PDF-Datei erfolgen. Nutzen Sie z.B. ImageMagick[1], um aus eingescannten Bildern ein PDF zu erzeugen. Achten Sie bei eingescannten Bildern darauf, dass diese im PDF-Dokument eine korrekte Orientierung haben. **Abgaben, die sich nicht in beschriebenen Formaten befinden, können nicht korrigiert oder bewertet werden!**

Aufgabe 1.3 [10 Punkte] (H) Labyrinth

In der Vorlesung haben Sie verschiedene Algorithmen kennengelernt, einen Weg aus einem Labyrinth zu finden. Der einfachste vorgestellte Algorithmus funktioniert dabei nur, wenn man selbst auf keiner *Insel* das Labyrinth betritt und auch der Ausgang sich auf keiner Insel befindet. Bei dieser Variante tastet man sich so an den Wänden entlang, dass man immer zu seiner rechten Seite eine Wand spürt (siehe Abbildung). Implementieren Sie das Durchwandern des Labyrinths mit Hilfe der Klasse `Maze`, die Ihnen zusammen mit diesem Blatt zur Verfügung gestellt wird. Die Klasse `Maze` bietet folgende Methoden:



- `boolean[][] generateMaze(int width, int height)`:
  Diese Methode erzeugt ein zufälliges Labyrinth. Es wird als ein zweidimensionales boolesches Feld mit `width` Spalten und `height` Zeilen repräsentiert. Hat dieses Array an der Stelle $[x][y]$ den Wert `true`, befindet sich im Labyrinth bei den Koordinaten (x, y) eine Wand.
- `draw(int x, int y, boolean[][] maze)`:
  Diese Methode gibt die eine Darstellung des Labyrinths `maze` mit der Person (markiert als roter Punkt) an den Koordinaten (x, y) aus.

Der Eingang befindet sich stets links oben im Labyrinth bei den Koordination $(1, 0)$, der Ausgang rechts unten im Labyrinth bei den Koordinaten `(width-1, height-2)`. Ein Labyrinth ist niemals kleiner als 3×3 Zellen.

Benutzen Sie die `draw`-Methode, um den Weg der Person durchs Labyrinth zu illustrieren. Wenn im generierten Labyrinth kein Weg zum Ausgang existiert, was durchaus vorkommen kann, soll Ihr Programm dies bemerken und sich beenden.

- Implementieren Sie eine Funktion `void walk(int x, int y, int direction)`, welche vom Startpunkt `(1,0)` aus immer entlang der rechten Wand weitergeht, die neue Position jeweils mittels `draw` ausgibt, und schließlich ganz am Ende ausgibt, ob das Ziel erreicht werden konnte oder nicht.
- Implementieren Sie eine `main`-Methode, welche mittels `generateMaze` ein Labyrinth erzeugt und dann mittels `walk` nach dem Ausgang sucht.

Hinweis: Zum Testen ist es nützlich, anstelle von `generateMaze` eine eigene Funktion zu verwenden, die immer dasselbe Labyrinth liefert.

[1] `http://www.imagemagick.org/script/index.php`

Aufgabe 1.4 [10 Punkte] **(H) Induktion**

a) Zeigen Sie für alle natürlichen Zahlen $n \geq 1$ **mittels Induktion** die folgende Behauptung:
Die Summe der ersten n ungeraden Zahlen ist n^2 (als Formel: $\sum_{i=1}^{n}(2i-1) = n^2$).
Kennzeichnen bzw. benennen Sie in Ihrem Beweis den Induktionsanfang, die Induktionsvoraussetzung und den Induktionsschritt.

b) Zeigen Sie $F_{n+1}F_{n-1} - F_n^2 = (-1)^n$, wobei F_n die n-te Fibonaccizahl nach der rekursiven Definition $F_n = F_{n-1} + F_{n-2}$ mit den Anfangswerten $F_0 = 0$ und $F_1 = 1$ ist.