Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 21.4.2015

Algorithmen und Datenstrukturen

SS 2015

Übungsblatt 2: O-Notation und Allgorithmen im Alltag

Besprechung: 30.4.–8.5.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 29.4.2015 13:25

Aufgabe 2-1 Allgemeine Fragen — nicht bewertet

(a) Erklären Sie mit eigenen Worten, was ein Algorithmus ist!

(b) Erklären Sie die Begriffe Finitheit, Terminierung, Determinismus, Determiniertheit.
Heben Sie den Unterschied zwischen den beiden letzten Begriffen hervor.

(c) Nennen Sie die grundlegenden Bestandteile eines Algorithmus.

(d) In Abhängigkeit von welcher Größe wird die Komplexität eines Algorithmus meist angegeben?

(e) Wofür verwendet man die O-Notation? Was drückt sie aus?

Aufgabe 2-2 Algorithmen im Alltag — nicht bewertet

Wir verwenden im Alltag ständig Algorithmen, ohne dass uns dies bewusst ist.
Überlegen Sie sich "Algorithmen" für folgende Problemstellungen.

(a) Die Suche mehrerer Produkte in einem Supermarkt, dessen Regale nach Produktgruppen geordnet sind.

(b) Versand einer Email mit Dateianhängen an mehrere Empfänger, die vielleicht nicht alle im Adressbuch sind.

(c) Die Suche des Ausgangs in einem einfachen (zyklenfreien) Labyrinth.
(Verwenden Sie zum Beispiel den Lösungsweg "Rechte Hand an rechter Wand")

(d) Abbiegen an einer Kreuzung ohne explizite Vorfahrtsregelung.

(e) Das Lösen von Rubiks Würfel.
(Es muss kein effizienter oder optimaler Algorithmus sein!)

(f) Das Finden des Heimwegs aus einem Wald mit Hilfe von einer Scheibe Brot.

Aufgabe 2-3 Verkettete Listen – Einfügen an anderen Positionen 5 Punkte

Auf langen Listen benötigt die Methode `append(daten)` wesentlich mehr Zeit zum Einfügen am Ende, als die Methode `prepend(daten)` zum Einfügen am Anfang der Liste.

- Erklären Sie diesen Effekt, und machen Sie einen einfachen Verbesserungsvorschlag, wie dieses Problem behoben werden kann.
- Vergleichen Sie die Laufzeitkomplexität des Einfügens und Anhängens, *vor und nach* der Verbesserung, in O-Notation.
- Warum funktioniert die gleiche Verbesserung nicht für Funktionen wie `elementAnPosition(index)` und `einfuegenAnPosition(index, daten)`?

Sie müssen diese Aufgabe nicht *implementieren*. Geben Sie Ihre Antwort zu dieser Aufgabe als `.pdf` ab.

Aufgabe 2-4 Verkettete Listen – Zugriff per Index 4 Punkte

Implementieren Sie eine Methode `get(int i)`, die das Element an der `i`-ten Position der Liste zurückliefert.

Das erste Element der Liste habe die Nummer `0`.

Wenn die Liste weniger als `i` Elemente hat, soll `null` zurückgeliefert werden.

Ihre Implementierung soll die Komplexität $O(n)$ (bei $i = \infty$) bzw. $O(i + 1)$ (bei $n = \infty$) haben.

(Hinweis: Verwenden Sie daher *nicht* die Funktion `size()`!)

Erweitern Sie die Klasse `Liste` vom ersten Übungsblatt um eine Methode:

```
/** Unterklasse, mit index-basiertem Zugriff */
public class ListeGet<INHALT> extends Liste<INHALT> {
  /** Konstruktor für eine leere Liste */
  public ListeGet() {
    super();
  };

  /* Element an Position pos (beginnend bei 0) suchen. */
  public INHALT get(int pos) {
    // TODO: Implementieren
  }
}
```

Sie können dafür die `Liste.java` von der Homepage, bei Blatt 2 verwenden.

Testen Sie ihr Programm geeignet, bevor Sie es abgeben.

Aufgabe 2-5 Verkettete Listen – Minimum und Maximum 5 Punkte

Gegeben Sei die **einfach verankerten verketteten Liste** vom ersten Übungsblatt.
Für diese Aufgabe spezialisieren wir die Klasse `Liste` mittels Vererbung auf `Integer` Werte.

- Ergänzen Sie die Methoden `min` und `max`:

```
/** Verkettete Liste für Daten vom Typ Integer */
public class IntegerListe extends Liste<Integer> {
  /** Konstruktor für eine leere Liste */
  public IntegerListe() {
    super();
  };

  /** Kleinstes Element der Liste suchen */
  public Integer min() {
    // TODO: Implementieren
  }

  /** Größtes Element der Liste suchen */
  public Integer max() {
    // TODO: Implementieren
  }
}
```

- Zum Testen ihrer Implementierung verwenden Sie (mindestens) folgende Methode:

```
/** Verkettete Liste für Daten vom Typ Integer */
public class IntegerListe extends Liste<Integer> {
  // ...

  /** Methode zum Testen der Implementierung */
  public static void main(String[] args) {
    IntegerListe liste = new IntegerListe();
    liste.append(2);
    liste.append(1);
    liste.append(3);
    liste.append(4);
    liste.append(0);
    System.out.println("Das Minimum ist: "+liste.min()+" erwartet: 0");
    System.out.println("Das Maximum ist: "+liste.max()+" erwartet: 4");
  }
}
```

- Warum haben wir für diese Aufgabe die Liste auf `Integer` spezialisiert?

- **Optional:** Implementieren Sie eine Klasse `ComparableList` die Funktionen `min()` und `max()` für beliebige Objekte, die das Java-Interface `Comparable` implementieren.

```
import java.util.*

/** Verkettete Liste für Daten vom Typ {@code Comparable<?>} */
public class ComparableListe<INHALT extends Comparable<? super INHALT>>
    extends Liste<INHALT> {
  /** Konstruktor für eine leere Liste */
  public ComparableListe() {
    super();
  };

  /** Kleinstes Element der Liste suchen */
  public INHALT min() {
    // TODO: Implementieren
  }

  /** Größtes Element der Liste suchen */
  public INHALT max() {
    // TODO: Implementieren
  }
}
```