Karlsruher Institut für Technologie

12. Tutorium
Wiederholung, Parsen, Suchen, Sortieren

Tutorium 14

Péter Bohner | 08.02.2023

Inhaltsverzeichnis

Wiederholung

Es gibt 19 Fragen, wir gehen der Reihe nach durch
Mit welchem Schlüsselwort werden Variablen unveränderbar gemacht? `final`
Sei int i = 1. Welchen Wert hat i nach der Ausführung folgendes Ausdrucks? (1 == 1) || (i++ == 5)
1, wegen der Kurzschlussauswertung wird die zweite Bedinung nie ausgeführt
Welches Schlüsselwort macht aus einer Methode eine Klassenmethode?
`static`
Wie macht man einen Down-Cast?

```
Kraftfahrzeug k = new Motorrad();
if (k instanceof Motorrad) { // Immer mit Typenüberprüfung
    Motorrad m = (Motorrad) k;
}
```

Wiederholung

Wahr oder Falsch: Eine Klasse kann nur einen Konstruktor haben.

Falsch, man kann die Parameterliste überladen

Wie schreibt man einen Getter für eine ArrayList von Strings?

Shallow copy reicht, da Strings immutable sind.

```
public List<String> getList() {
    return this.list.clone();
}
```

Wie deklariert man in Java die Verwendung eines Generics `T` in einer Klasse?

```
public class ClassName<T> { ... }
```

Welche Annotation benutzt man, um eine Methode for jedem JUnit5 Testfall auszuführen?

```
@BeforeEach
```

Wiederholung

Was ist die Standardsichtbarkeit von Attributen und Methoden?

package-private, also nur innerhalb des Paketes sichtbar

Was kann bei der Widening Conversion auftreten?

- Informationsverlust bezüglich Genauigkeit
- z.B. `float a = 23456789; // a == 23456788.0`

Wie schreibt man einen Getter für eine ArrayList von veränderbaren Person-Objekten?

```
public List<Person> getList() {
    List<Person> copy = new ArrayList<>();
    for(var p : this.list)
        copy.add(new Person(p)); // copy constructor
    return copy;
}
```

Wiederholung

Was berechnet diese Methode?

```
public int something(int x) {
    if(x <= 1)
        return 1;
    return something(x - 2) + something(x - 1);
}
```

Die x-te Element der Fibonacci Folge (von 0 indiziert)

Was ist ein Beispiel für Narrowing Conversion?

```
byte a = (byte) 1234; // a == 214
```

Wiederholung

Wie schreibt man einen Copy-constructor für die Folgende Klasse?

```
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
public Person(Person p) {
    return new Person(p.name, p.age);
}
```

Wie kann man diese Klasse effizienter Schreiben? Mit Records aus Java 14

```
public record Person(String name, int age){}
```

Wiederholung

Was wisst ihr über Arrays? (`int[] a`)

- Haben immer eine feste Länge
- Werden in Java wie Objekte behandelt (Pass by Reference!)
- Können nur Instanzen eines Datentyps speichern (in diesem Beispiel ints)

Durch welches Schlüsselwort kann man die Ableitung einer Klasse verbieten?

```
final class C
```

Wie macht man einen Up-Cast?

```
Kraftfahrzeug k = new Motorrad(); // Harmlos!
```

Wie gibt man den konkreten Datentyp bei der Instanzierung einer Klasse die Generics nutzt an?

```
ClassName<String> objectName = new ClassName<>();
```

Parsen

Problem:
Wie kann eine Zeichenkette in semantisch sinnvolle Teile zerlegt werden ?

Crashkurs: Formale Grammatiken

Ganz formal...

Definition

Eine Grammatik ist ein 4-Tupel $\mathcal{G} = (\mathcal{V}, \Sigma, \mathcal{P}, \mathcal{S})$ bestehend aus:

- $\mathcal{V}$, endliche Menge von **Variablen**
- Σ, endliche Menge von **Terminalsymbolen** (mit $\Sigma \cap \mathcal{V} = \emptyset$)
- $\mathcal{P} \subset (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$, endliche Menge von **Produktionsregeln**
- $\mathcal{S} \in \mathcal{V}$, **Startsymbol**

Crashkurs: Formale Grammatiken

Jetzt ein konkretes Beispiel für eine Grammatik...

Klammerausdruck

- $\mathcal{G} = (\mathcal{V} = \{X, Y\}, \quad \Sigma = \{(,), expr\}, \quad \mathcal{P}, \quad \mathcal{S} = X)$
- $\mathcal{P} = \{$
  $X \longrightarrow (X)$
  $X \longrightarrow XX$
  $X \longrightarrow Y$
  $Y \longrightarrow expr$
  $\}$

Crashkurs: Formale Grammatiken

- *Dann sind folgende Wörter gültige Klammerausdrücke:*
  - `expr`
  - `(expr)`
  - `(expr)(exprexpr)`
  - `expr(expr(expr)(expr(expr)))expr((expr)expr)`
  - `...`
- *Aber nicht sowas:*
  - `)(`
  - `expr()`
  - `)(expr)`
  - `term`
  - `...`

Top-down Parsing

Idee:

- Verwende eine **formale Grammatik** um die gewünschte Sprache zu beschreiben.
- Spalte die Eingabe in einzelne **Symbole** oder **Tokens**.
- Ausgehend vom **Anfangssymbol** bestimme die nächste anzuwendende Produktion.

LL(1)-Parser

- Ein **LL(1)-Parser** ist ein einfacher **Top-down-Parser**.
- Betrachte immer nur *ein einziges Zeichen* (ganz links) um die nächste Regel zu entscheiden.
- Grammatik darf dementsprechend **keine Konflikte bzgl. des ganz linken Symbols** haben (sonst siehe Links-Faktorisierung), da sonst mehrere Produktionen in Frage kommen und der Parser nicht mehr entscheiden kann, welche davon als nächstes angewandt werden soll.

LL(1)-Parser: Beispiel

Ein konkretes Beispiel...

LL(1)-Grammatik

- $\mathcal{G} = (\mathcal{V} = \{X, Y\}, \quad \Sigma = \{(,), +, a\}, \quad \mathcal{P}, \quad \mathcal{S} = X)$
- $\mathcal{P} = \{$
  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$
  $\}$

Eingabe:

```
((a+a)+a)
```

LL(1)-Parser: Beispiel

- Eingabe:

  ```
  ((a+a)+a)
  ```

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X

LL(1)-Parser: Beispiel

- Eingabe:

  ((a+a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X

LL(1)-Parser: Beispiel

- Eingabe:

  ```
  ((a+a)+a)
  ```

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  (
  X
  $+$
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  ```
  ((a+a)+a)
  ```

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  (
  X
  $+$
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  `(a+a)+a)`

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  `(a+a)+a)`

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  `(a+a)+a)`

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  (
  X
  $+$
  Y
)
  $+$
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  (a+a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  (
  X
  +
  Y
)
  +
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  `a+a)+a)`

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X
  $+$
  Y
  $)$
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  a+a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  X
  $+$
  Y
  $)$
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  a+a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  Y
  $+$
  Y
  $)$
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  a+a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  a
  $+$
  Y
  $)$
  $+$
  Y
  $)$

LL(1)-Parser: Beispiel

- Eingabe:

  +a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  \+
  Y
)
  \+
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  Y
)
  +
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  a)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  a
)
  +
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

)+a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

)
  +
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  +a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  +
  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  Y
)

LL(1)-Parser: Beispiel

- Eingabe:

  a)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  a
)

LL(1)-Parser: Beispiel

- Eingabe:

)

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

)

LL(1)-Parser: Beispiel

- Eingabe:

  $\emptyset$

- Produktionen:

  $X \longrightarrow Y$
  $X \longrightarrow (X + Y)$
  $Y \longrightarrow a$

- Symbol-Stack:

  $\emptyset$

Suchen

Problem:
Wie finde ich ein bestimmtes Element in einer Liste ?

4	2	3	9	7	13	6	10

Lineare Suche

- Erster (naiver) Ansatz: **Lineare Suche**
- Idee: Ein Element nach dem anderen anschauen und vergleichen.

Lineare Suche: Java-Code

```java
int linearSearch(int needle, int[] haystack) {
    for (int i = 0; i < haystack.length; i++) {
        if (haystack[i] == needle) {
            return i;
        }
    }
    return -1;
}
```

Lineare Suche

- Laufzeit: $O(n)$
- Für beliebige Listen geht es nicht wirklich besser.

Rätsel

Es wurde ein neuer Info-Bau an der Uni errichtet. Das neue Gebäude besitzt ganze 256 Stockwerke. Die Informatiker sind zunächst begeistert, doch schon bald gibt es ein Problem:
Da bekanntlich alle Informatiker Keller gewohnt sind, können sie die Gefahr von offenen Fenstern nicht einschätzen und fallen gelegentlich hinaus.
Daraufhin sollten zunächst alle Fenster des Gebäudes verriegeln werden, doch man hat festgestellt, dass einige Informatiker den Sturz munter überstanden haben.
Nun geht man davon aus, dass es ein Stockwerk n gibt, ab dem die Fenster keine Gefahr mehr darstellen. Zu diesem Zweck sollen nun Tests durchgeführt werden, bei denen man jeweils einen Informatiker aus dem Fenster springen lässt und beobachtet, ob der Sturz munter überstanden wurde. Da die Informatiker aber lieber die Anzahl der Tests als die Anzahl der unglücklichen Landungen minimieren wollen, kommt eine lineare Suche (Stockwert 1, Stockwerk 2, ...) nicht in Frage.
Wieviele Testfälle braucht man im ungünstigsten Fall ?

Lösung

8

. . . Aber warum ?

Binäre Suche

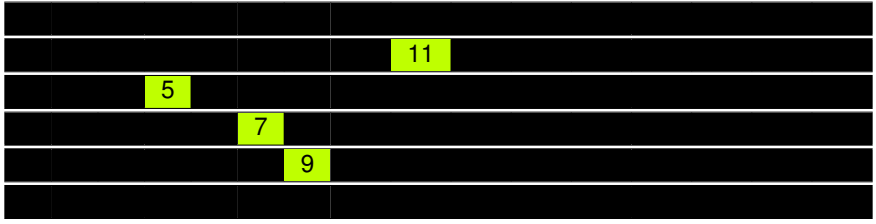
- Ansatz: Suche auf einer vorsortierten Liste.
- Idee: Teile die Liste in zwei Hälften und entscheide, in welcher der beiden Teillisten das gesuchte Element liegt und verwerfe die andere.
  Dann wiederhole den Vorgang auf der übrigen Liste.

Binäre Suche: Java-Code

```java
int binarySearch(int needle, int[] haystack, int from, int to) {
    if (to > from) {
        int mid = (from + to) / 2;
        if (needle < haystack[mid]) {
            return binarySearch(needle, haystack, from, mid - 1);
        } else if (needle > haystack[mid]) {
            return binarySearch(needle, haystack, mid + 1, to);
        } else {
            return mid;
        }
    }
    return -1;
}
```

Binäre Suche: Beispiel

Suche: **9**



Binäre Suche

- Laufzeit: $O(log(n))$
- Die Liste muss **sortiert** sein.

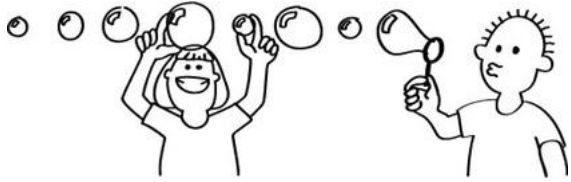
Sortieren

Problem:
Wie sortiere ich eine gegebene Liste ?

4	2	3	9	7	13	6	10

Bubble Sort

- Ansatz: Vertausche zwei unsortierte benachbarte Elemente, solange bis die Liste sortiert ist.



Bubble Sort: Java-Code

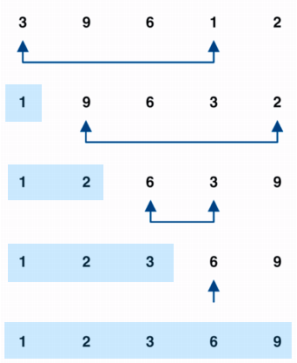
```java
void bubbleSort(int[] list) {
    int n = list.length;
    int swap;
    boolean swapped = true;
    while (swapped) {
        swapped = false;
        for (int i = 1; i < n; i++) {
            if (list[i] < list[i-1]) {
                swap = list[i];
                list[i] = list[i - 1];
                list[i - 1] = swap;
                swapped = true;
            }
        }
        n--;
    }
}
```

Bubble Sort

- Laufzeit: $O(n^2)$
- Sehr naiver Algorithmus, allerdings gut, wenn die Liste quasi schon sortiert ist.

Selection Sort

- Ansatz: Suche immer das kleinste Element der unsortierten Liste und füge es an den Rand der sortierten Liste ein.



Selection Sort: Java-Code

```java
void selectionSort(int[] list) {
    int min, swap;
    for (int i = 1; i < list.length; i++) {
        min = i - 1;
        for (int j = i; j < list.length; j++) {
            if (list[j] < list[min]) {
                min = j;
            }
        }
        if (min != i - 1) {
            swap = list[i - 1];
            list[i - 1] = list[min];
            list[min] = swap;
        }
    }
}
```

Selection Sort

- Laufzeit: $O(n^2)$
- Braucht *immer* $\frac{n(n-1)}{2}$ Vergleiche !

Insertion Sort

- Ansatz: Füge jedes Element der unsortierten Liste an die passende Stelle innerhalb der sortierten Liste.

Insertion Sort: Java-Code

```java
void insertionSort(int[] list) {
    int j, current;
    for (int i = 1; i < list.length; i++) {
        current = list[i];
        j = i - 1;
        while (j >= 0 && list[j] > current) {
            list[j + 1] = list[j--];
        }
        list[j + 1] = current;
    }
}
```

Insertion Sort

- Laufzeit: $O(n^2)$
- Das Vorgehen ist mit der Sortierung eines Spielkartenblatts vergleichbar.

Danke für Eure Mitarbeit!

Viel Erfolg im weiteren Studium!