Datenstrukturen und effiziente Algorithmen I

Sommersemester 2023
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Lehrstuhl für Datenstrukturen und effiziente Algorithmen

Übungsblatt 8

Abgabe: 06. Juni 2023

Aufgabe 1: Erkennen von Klammerausdrücken (3+3 Punkte)

Ein *korrekter* Klammerausdruck K sei über folgende kontextfreie Grammatik beschrieben:

$$K \to (K) \tag{1}$$
$$K \to KK \tag{2}$$
$$K \to \varepsilon \tag{3}$$

Betrachten Sie folgenden Algorithmus (Folie 207) zur Erkennung von Klammerausdrücken:

Algorithmus (unter Verwendung eines Stacks):

- Lese die Folge der Klammern von links nach rechts.
- Betrachte die aktuell gelesene Klammer:
  - Falls „(“ gelesen, so lege diese auf den Stack.
  - Falls „)“ gelesen, so nimm eine „(“ vom Stack, erkläre diese als zur momentan gelesenen „)“ gehörig.
- Erkläre den Klammerausdruck als korrekt, falls der Stack am Ende leer ist, jedoch zwischendurch niemals versucht wurde, auf den leeren Stack `pop()` anzuwenden.

Beweisen Sie mittels vollständiger Induktion über die Länge n eines Klammerausdrucks K, dass der Algorithmus...

(a) eine korrekte Klammerung meldet, wenn K korrekt geklammert ist,

(b) eine nicht-korrekte Klammerung meldet, wenn K nicht korrekt geklammert ist.

Aufgabe 2: Klammerausdrücke mit mehr als einem Klammertyp (4 Punkte)

Überlegen Sie sich, wie Sie den Algorithmus aus Aufgabe 1 erweitern müssen, damit dieser Klammerausdrücke mit mehr als einem Klammertyp (z.B. `'('` und `')'`, `'{'` und `'}'`, sowie `'['` und `']'`) korrekt identifizieren kann. Begründen Sie Ihre Antwort.

Aufgabe 3: Klammerausdrücke: Implementierung (3+4 Punkte)

Betrachten Sie das Problem zur Erkennung von Klammerausdrücken aus Aufgabe 1.

(a) Implementieren Sie eine generische Klasse `Stack<T>` mittels einer verketteten Liste (siehe Folie 205). Ihre Klasse soll folgende Methoden besitzen:

```
public class Stack<T> {
    public void push(T t) { ... };
    public void pop() { ... };
    public T top() { ... };
    public boolean isEmpty() { ... };
    public int size() { ... };
}
```

(b) Erstellen Sie ein Programm `IdentifyBrackets`, das eine Textdatei einliest und die darin enthaltenen Klammerpaare identifiziert. Dabei soll zwischen den drei Klammertypen `'('` und `')'`, `'{'` und `'}'`, sowie `'['` und `']'` unterschieden werden.

Beispielsweise soll Ihr Programm in dem Beispiel `example1.txt` mit folgendem Inhalt alle vier Klammerpaare identifizieren und einander zuordnen:

```
int f(int[] c) {
    return c[0];
}
```

Die Ausgabe Ihres Programmes soll sich an folgendem Beispiel orientieren:

```
java IdentifyBrackets example1.txt
Matching brackets: '[' at line 1, column 10 and ']' at line 1, column 11
Matching brackets: '(' at line 1, column 6 and ')' at line 1, column 14
Matching brackets: '[' at line 2, column 13 and ']' at line 2, column 15
Matching brackets: '{' at line 1, column 16 and '}' at line 3, column 1
```

(c) Bonus (+3): Erweitern Sie Ihren Klasse um eine Fehlerbehandlung falsch gesetzter Klammern. Sobald ein Fehler gefunden wurde, soll dieser ausgegeben und Ihr Programm beendet werden. Es soll also auch nur der erste Fehler ausgegeben werden.

Beispielsweise soll Ihr Programm in dem Beispiel `example2.txt` mit folgendem Inhalt die falsche Klammerung in Zeile 2 und Spalte 13 finden:

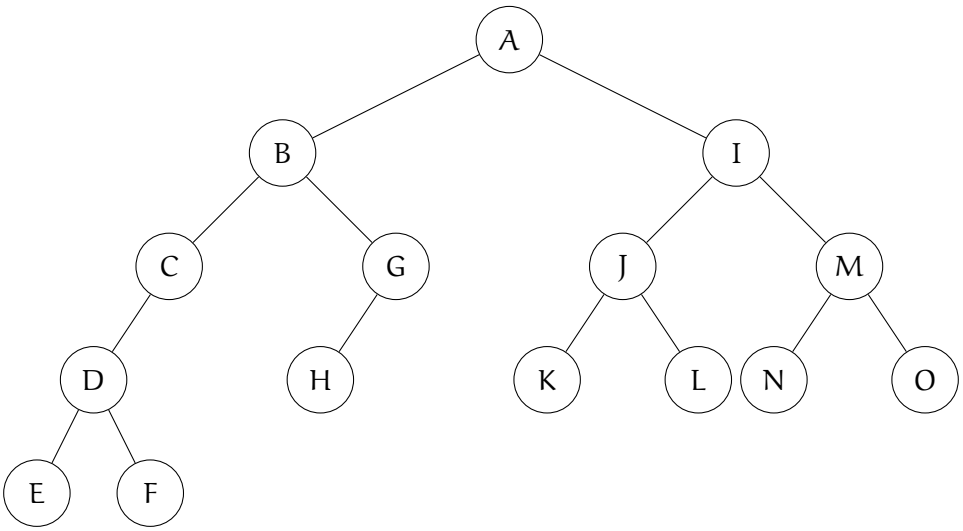
```
int f(int[] c)(
    return c[0);
}
```

Die Ausgabe Ihres Programmes könnte für `example2.txt` wie folgt aussehen.

```
java IdentifyBrackets example2.txt
Matching brackets: '[' in line 1, column 10 and '] at line 1, column 11
Matching brackets: '(' in line 1, column 6 and ') at line 1, column 14
Error at line 2, column 13: Reading ')' but expecting ']' because of '['
at line 2 column 11.
```

Aufgabe 4: Traversierung von Binärbäumen (3 Punkte)

Geben Sie für den folgenden Binärbaum



die Knoten in

a) Pre-Order (WLR),

b) In-Order (LWR),

c) und Post-Order (LRW) an.