ulm university universität
uulm

1. Klausur zur Veranstaltung

Einführung in die Informatik I - Grundlagen

und **Allgemeine Informatik 2**

im Sommersemester 2022

Dr. Jens Kohlmeyer, Stefan Höppner

25.07.2022

Musterlösung

Aufgabe 1 - Verständnisfragen **3 + 5 + 4 = 12 Punkte**

a) Erläutern Sie, **in ganzen Sätzen**, was unter der "bad-character"-Strategie im Boyer-Moore Algorithmus verstanden wird. Gehen Sie in Ihrer Erläuterung auf die verschiedenen Fälle, welche die Strategie abdeckt, mittels Beispielen ein.

Lösung

Die "bad-character"-Strategie definiert um wie viel ein gesuchtes Pattern beim Auftreten eines Zeichenmissmatches (bad-character) verschoben werden kann. Dabei werden 2 Fälle unterschieden:

i. Das Zeichen kommt gar nicht im Pattern vor → es kann ein Shift bis hinter den bad-character vorgenommen werden

ii. Das Zeichen kommt im Pattern vor, aber an einer anderen Stelle → es kann ein Shift gemacht werden, so dass das letzte Vorkommen des Zeichens im Pattern mit der Position des Zeichens im Suchstring übereinstimmt.

b) Erläutern Sie, **in ganzen Sätzen**, das Prinzip, die Voraussetzungen und die Vorgehensweise des *LSD-Radixsort*.

Lösung

Prinzip: statt Vergleich von Elementen werden einzelne Symbole (bspw. Ziffern) der Elemente verglichen.
Voraussetzungen:

- Die zu sortierenden Elemente müssen sich bezogen auf das Sortierkriterium (Schlüssel) aus einzelnen Symbolen zusammen setzen
- einzelne Symbole der Elemente (aus einem endlichen Alphabet) müssen vergleichbar sein (Ordnungsrelation).

Vorgehen: Partitionierung und Sammlung im wechsel von niederwertigster Stelle aufwärts.

- Partitionierung: Für jedes Element a_i : lege a_i in Fach F_y, falls das betrachtete Symbol von a_i dem Symbol $y \varepsilon A$ entspricht
- Sammlung: Erzeuge eine neu angeordnete Folge $a_1, ..., a_n$ durch hintereinander fügen der Inhalte der Fächer $F_1, ..., F_m$. Beginne mit dem Fach mit der niedrigsten Wertigkeit und ändere die Reihenfolge der Elemente darin nicht

c) Nennen und Beschreiben Sie, **in ganzen Sätzen**, **zwei** Maßnahmen zur Qualitätssicherung von Softwaresystemen.

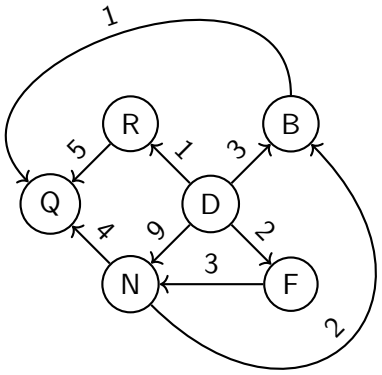
Lösung

- Tests: Ausführen der Software unter vordefinierten Bedingungen und Inputs zur Auffindung von Fehlverhalten.
- Reviews: Statische Prüfung eines Dokuments (z.B Lastenheft oder Code) allein oder mit Partner zur manuellen Aufdeckung von Defekten.

Aufgabe 2 - Graphen

2 + 3.5 + 3.5 + 3 = 12 Punkte

Betrachten Sie folgenden gerichteten, gewichteten Graphen G:



a) Bewerten Sie folgenden Aussage bezüglich ihrer Richtigkeit und begründen Sie Ihre Antwort: "Jeder gerichtete azyklische Graph ist ein Baum."

Lösung

Diese Aussage ist falsch, da ein azyklischer Graph nicht ausschließt, dass sich zwei Knoten einen Kindknoten teilen. Dies muss bei Bäumen jedoch gegeben sein.

b) Wenden Sie den Dijkstra-Algorithmus mit Startknoten D auf G an.

Lösung

Schritt	**Menge** S	**Menge** $V§$
0	$\emptyset$	(D,0)
1	(D,0)	(B,3), (F,2), (N,9), (Q,∞), (R,**1**)
2	(D,0), (R,1)	(B,3), (F,**2**), (N,9), (Q,6)
3	(D,0), (R,1), (F,2)	(B, **3**), (N,5), (Q,6)
4	(D,0), (R,1), (F,2), (B,3)	(N,5), (Q,**4**)
5	(D,0), (R,1), (F,2), (B,3), (Q,4)	(N,5)
6	(D,0), (R,1), (F,2), (B,3), (Q,4), (N,5)	$\emptyset$

c) Geben Sie die Adjazenzliste für G an.

Lösung

B $\rightarrow [Q]$
D $\rightarrow [B, F, N, R]$
F $\rightarrow [N]$
N $\rightarrow [B, Q]$
Q $\rightarrow []$
R $\rightarrow [Q]$

d) Erläutern Sie eine Änderung an G die Sie vornehmen müssten um den Graph nicht mehr topologisch sortierbar zu machen und erklären Sie warum dies der Fall ist.

Lösung

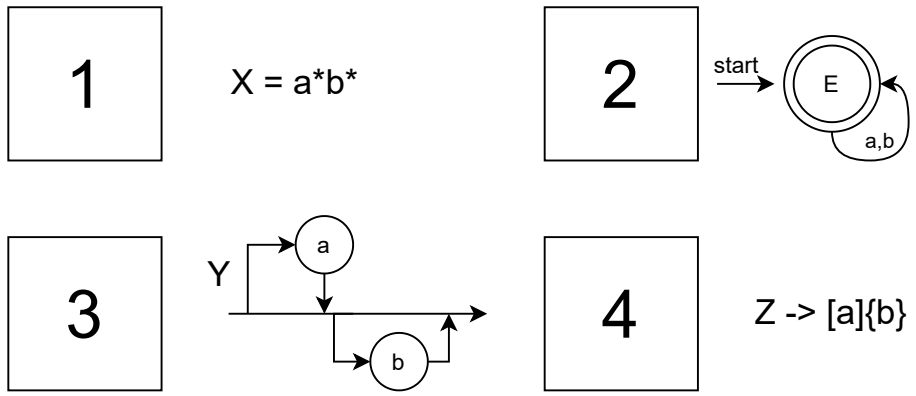
z.B. Kante von Q nach D, dadurch entsteht ein Zyklus und ein topologisch sortierbarer Graph darf keine Zyklen enthalten.

Aufgabe 3 - Formale Sprachen

4 + 3 + 5 = 12 Punkte

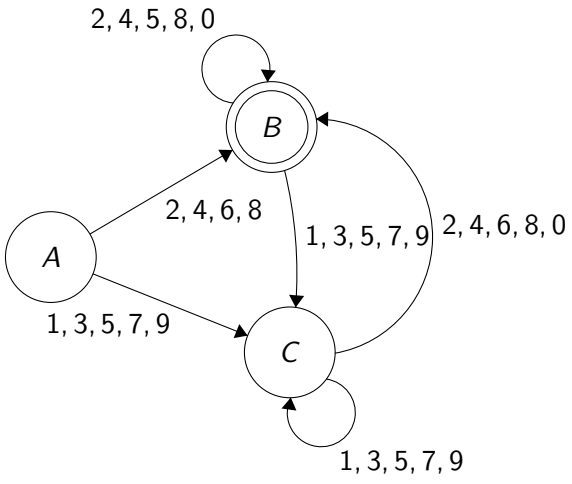
a) Im Folgenden sehen Sie einen regulären Ausdruck, einen endlichen Automaten, ein Syntaxdiagramm und eine EBNF. Markieren Sie, welche davon jeweils die gleiche Sprache erzeugen bzw. beschreiben, indem Sie in die Box eine Zahl eintragen. *Wären Sie zum Beispiel der Meinung, dass alle unterschiedliche Sprachen beschreiben bzw. erzeugen, dann tragen Sie in jede Box eine andere Zahl ein.*

Lösung



b) Geben Sie einen **deterministischen** endlichen Automaten an, der alle positiven, geraden Dezimalzahlen, ohne führende Nullen akzeptiert.

Lösung



c) Die Programmiersprache LISP zeichnet sich durch eine sehr simple Syntax aus die sich leicht in eine Grammatik und, daraus resultierend, leicht parsen lässt.
In dieser Aufgabe betrachten wir ein Subset valider LISP Ausdrücke, hier genannt LISP^-:
Jeder Ausdruck in LISP^- beginnt mit einer öffnenden und endet mit einer schließenden Klammer. Ein LISP^- Ausdruck ist eine Operation mit 2 bzw. 3 Argumenten in Präfixnotation (d.h der Name der Operation kommt vor den Operanden). Operationen bekommen immer entweder eine ganze Zahl > 0 oder weitere LISP^- Ausdrücke als Operanden übergeben. Die erlaubten 2-Argument Operationen in LISP^- sind: $+$, -, * und $/$. Die einzige erlaubte 3-Argument Operation in LISP^- ist: *if*.
Definieren Sie eine Grammatik die alle gültigen LISP^- Ausdrücke akzeptiert.
Beispiel: Folgender Ausdruck ist ein valider LISP^- Ausdruck: $(* \ (+ \ 1 \ 2) \ (\mathit{IF} \ 0 \ (+ \ 1 \ 2) \ (- \ 900 \ 10)))$

Lösung

$V = \{S, OP, 2OP, C, 3OP, EXP, NBR\}$
$\Sigma = \{0,1,2,3,4,5,6,7,8,9,+,-,*,/,if,(,)\}$
$S = \{S\}$
$P = \{S \rightarrow (OP)$
$OP \rightarrow 2OP \mid 3OP$
$2OP \rightarrow C \ EXP \ EXP$
$C \rightarrow + \mid - \mid * \mid /$
$3OP \rightarrow if \ EXP \ EXP \ EXP$
$EXP \rightarrow S \mid NBR$
$NBR \rightarrow 0 \mid ... \mid 9 \mid 1NRB \mid ... \mid 9NBR$
$\}$

Aufgabe 4 - Java - Datenstrukturen **3 + 4 + 5 = 12 Punkte**

Betrachten Sie folgende Klassen und Beschreibung:

```java
public class Library {
    private X books;
}
```

```java
public class Book {
    public String title;
    protected Y pages;
}
```

```java
public class Page {
    public String content;
}
```

```java
public interface Readable<T> {
    public String read();
}
```

In einer Bibliothek (`Library`) werden Bücher nach Genre sortiert abgelegt. Genres werden direkt über ihren Namen identifiziert.
Bücher (`Books`) enthalten Seiten (`Pages`) in einer sortierten Reihenfolge, d.h. Seite 2 folgt auf Seite 1, Seite 3 folgt auf Seite 2 usw.

a) Geben Sie sinnvolle Typen für die Platzhalter `X` und `Y` an. Begründen Sie, warum Sie sich für diese Typen entschieden haben. Es ist Ihnen erlaubt selbst neue Typen zu definieren und zu verwenden. Geben Sie hierfür die komplette Klassendefinition der neu konzipierten Typen an.
Sie dürfen jeden Datentyp nur **einmal** verwenden.

Lösung

X: *Map* $<$ *String*, *Collection* $<$ *Book* $>>$: Genere wird als String repräsentiert, Map erlaubt direkte Zuordnung, Collection erlaubt eine Sammlung von Büchern pro Genere
Y: *Stack* $<$ *Page* $>$: stellt das aufeinanderfolgen von Buchseiten direkt dar
Alternativen sind zulässig, z.B *List* $<>$ für Y oder *List* $<$ *Tuple* $<>>$ für X wenn eine Tuple Definition existiert.

b) Beschreiben Sie, wie, auf Basis Ihrer in **a)** gewählten Datentypen folgende Aufgabe zu lösen ist. Nummerieren Sie die einzelnen Schritte Ihrer Lösung so, dass eine klare Reihenfolge erkennbar ist:
Finden Sie, ausgehend von einer Library, heraus, ob ein Book mit dem Titel "LISP für Anfänger" in den Büchern des Genre "Sachbücher" existiert.
Beispiel für die Darstellung *Prüfen Sie, ob die Anzahl der Zeichen auf einer Page größer als 1000 ist:*
1. Länge von `content` mittels content.length() ermitteln
2. Vergleich der ermittelten Länge mit 1000, falls größer `true` zurückgeben, sonst `false`

Lösung

1. Alle "Sachbücher" aus der Map mittels books.get('Sachbücher') holen
2. Über Collection iterieren
3. in jeder Iteration prüfen ob buch.title gleich "LISP für Anfänger" ist, falls ja `true` zurück liefern
4. falls Iteration terminiert `false` zurück geben.

c) Welche Änderungen müssen Sie an der Klasse `Book` vornehmen, so dass diese das Interface `Readable<Page>` implementiert wenn folgende Eigenschaften für das Interface definiert sind:

- Mit `Readable<T>` wird angegeben, dass der Inhalt von `T` nach und nach stückweise gelesen werden kann.
- `read` liefert bei jedem Aufruf weiteren, bis jetzt noch nicht gelesen Inhalt, von `T` in Form eines Strings zurück.
- Wie `read` die Daten intern aufteilt, um stückweises Lesen zu ermöglichen, ist beliebig, muss aber sicherstellen, dass niemals 2 mal der selbe Inhalt zurückgeliefert wird.
- Sollte kein weiterer Inhalt lesbar sein so muss `read` `null` zurückliefern.

Geben Sie hierfür die neue Definition der gesamten Klasse `Book` mit allen Attributen und Methoden an.

Lösung

```java
public class Book implements Readable<Page>{
    public String title;
    private int pos = 0;
    protected List<Page> pages;

    public String read() {
        var nextPage = pages.get(pos);
        if(nextPage != null) {
            pos++;
            return nextPage.content;
        } else {
            return null;
        }
    }
}
```

Aufgabe 5 - Java - Familien

3 + 4 + 5 = 12 Punkte

Betrachten Sie folgende Klassen, die Familienrelationen darstellen:

```java
public class Person {
    boolean female;
    String name;
    Female mother;
    Male father;
    Person[] siblings;
    Person[] kids;
}
```

```java
public class Male extends Person {
    public Male() {
        this.female = false;
    }
}
```

```java
public class Female extends Person {
    public Female() {
        this.female = true;
    }
}
```

```java
public enum Direction {
    UP, DOWN, RIGHT;
}
```

a) Implementieren Sie die Methode `isAncestor(Person p)` der Klasse `Person` welche prüft, ob eine übergebene Person `p` Vorfahre der aktuellen Person ist.

Lösung

```java
public boolean isAncestor(Person p) {
    if (this == p) {
        return true;
    } else {
        return this.father.isAncestor(p) || this.mother.isAncestor(p);
    }
}
```

b) Implementieren Sie die Methode `parseDirections(String str)` der Klasse `Person`. Die Methode soll aus einem übergebenen String ein Array mit Richtungsangaben erzeugen. Dabei gelten die folgenden Regeln:

- Richtungsangaben sind mit dem Symbol ',' getrennt.
- Die Richtung `UP` wird mit dem Wort "Hoch" angegeben.
- Die Richtung `DOWN` wird mit dem Wort "Runter" angegeben.
- Die Richtung `RIGHT` wird mit dem Wort "Side" angegeben.
- Angaben von anderen Wörtern werden ignoriert.

Hinweis: Das zurückgegebene Array darf leere Felder enthalten.

Lösung

```
1  public Direction[] parseDirection(String str) {
2      String[] directions = s.splitAt(",");
3      Direction[] dirs = new Direction[directions.length];
4      for (int i = 0; i < directions.length; i++) {
5          if(s.equals("Hoch")) {
6              dirs[i] = Directions.UP;
7          } else if (s.equals("Runter")) {
8              dirs[i] = Directions.DOWN;
9          } else if (s.equals("Side")) {
10             dirs[i] = Directions.RIGHT;
11         }
12     }
13     return dirs;
14 }
```

c) Implementieren Sie die Methode `traverseFamily(String str)` der Klasse `Person`. Die Methode soll, auf Basis eines Strings der der Formatierung aus Aufgabe b) entspricht durch den Stammbaum der Person navigieren und die Person am Ende der Traversierung zurückgegeben. Dabei gilt:

- UP navigiert zum Vater, insofern dieser existiert.
- DOWN navigiert zum ersten Kind (an der ersten Position im kids-Array), insofern dieses existiert.
- RIGHT navigiert zum letzten Bruder oder zur letzten Schwester (an der letzten Position im siblings-Array), insofern diese/r existiert.

Hinweis: Sie dürfen Ihre parse-Methode aus Aufgabe b) verwenden. Achten Sie auf den Aufbau des zurückgegebenen Arrays!

Lösung

```
1  public Person traverseFamily(String str) {
2      Direction[] dirs = parseDirection(str);
3      Person p = this;
4      for (Direction d: dirs) {
5          if (d != null) {
6              switch (d) {
7                  case UP:
8                      p = p.father != null?p.father:p;
9                      break;
10                 case DOWN:
11                     p = p.kids != null && p.kids[0] != null?p.kids[0]:p;
12                     break;
13                 case RIGHT:
14                     p = p.siblings != null
15                       && p.siblings[p.siblings.length -1] != null
16                          ?p.siblings[p.siblings.length -1]:p;
17                     break;
18             }
19         }
20     }
21     return p;
22 }
```