ulm university universität
uulm

2. Klausur zur Veranstaltung

Einführung in die Informatik I - Grundlagen

und **Allgemeine Informatik 2**

im Sommersemester 2021

Dr. Jens Kohlmeyer, Stefan Höppner, Dr. Markus Maucher

25.09.2021

Musterlösung

Aufgabe 1 - Verständnisfragen

3 + 4 + 5 = 12 Punkte

a) Erklären Sie, was der *Rand* eines Wortes und dessen *Dicke* sind, und geben Sie die Ränder der Worte *erdbeere* und *fffff* an.

Lösung

Der Rand eines Wortes $w_0 w_1 ... w_{L-1}$ mit Länge L ist das längste Teilwort $w_0 w_1 ... w_{r-1}$ der Länge $r < L$, das identisch ist mit dem Wortende $w_{L-r} w_{L-r-1} ... w_{L-1}$ ist. r ist dabei die Dicke des Randes.
erdbeere: Rand = e
fffff: Rand = ffff, da $r < L$ gelten muss.

b) Erläutern Sie die Vorgehensweise des *Boyer-Moore Algorithmus* zur Suche in Texten unter Verwendung der *Bad-Character-Heuristik*. Gehen Sie dabei auch auf die Unterschiede zum *Knuth-Morris-Pratt Algorithmus* ein.

Lösung

Das Muster wird am Anfang linksbündig unter den Text "geschrieben" und dann von rechts nach links Zeichen für Zeichen mit dem Text verglichen. Sobald ein Mismatch auftritt, berechnet die BCH, wie weit das Suchmuster nach rechts verschoben werden kann.
Stimmt beim Vergleich des Musters mit dem Text von rechts nach links ein Zeichen des Musters nicht mit dem Zeichen des Textes überein (Bad-Character), wird im Muster nach dem letzten Vorkommen dieses Bad-Characters gesucht und das Muster so weit verschoben, bis beide Buchstaben übereinander liegen. Existiert dieser Bad-Character nicht im Muster, wird das Muster soweit verschoben, dass sein erstes Zeichen unter dem Nachfolger des Bad-Character liegt. Es kann vorkommen, dass die Bad-Character-Heuristik eine Verschiebung des Musters nach links vorschlägt. In diesem Fall wird um eine Position nach rechts geschoben.
Unterschiede zu KMP:
Vergleich von rechts nach Links
Zeichen im Text werden Berücksichtigt nicht nur die im Muster

c) Nennen Sie die **5** Phasen der Softwareentwicklung. Beschreiben Sie für jede Phase kurz deren Funktion für die Softwareentwicklung.

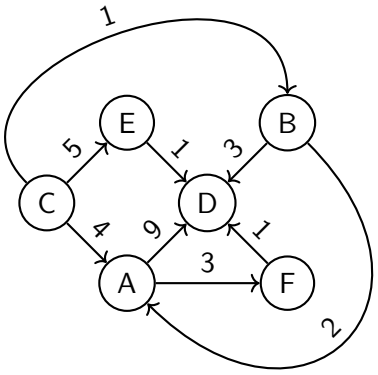
Lösung

Requirements Engineering: Tätigkeiten zur Klärung der Aufgabenstellung (Ermittlung, Beschreibung und Überprüfung der Anforderungen)
Entwurf: Präzise, konsistente und vollständige Beschreibung der Lösungskonzeption für ein neues (Software-)System
Implementation: Umsetzung eines Entwurfs in Code
Integration, Test und Abnahme: Prüfung der Funktionalität der Implementierung außerhalb und innerhalb der realen Gegebenheiten.
Wartung: Arbeiten, die nach Abschluss der Entwicklung am System vorgenommen werden (Korrektur von Fehlern, Portierung, Weiterentwicklung, Schulung, ...)

Aufgabe 2 - Graphen

2 + 3.5 + 3.5 + 3 = 12 Punkte

Betrachten Sie folgenden gerichteten, gewichteten Graphen G:



a) Erklären Sie den Begriff *Weg* im Kontext von Graphen. Geben Sie ein Beispiel aus dem Graph G an.

Lösung

Ein Weg im Graphen G = (V, E) ist eine endliche Folge $v_0, v_1, ...v_S$ von Knoten mit $\{v_{i-1}, v_i\} \varepsilon E$ für alle i mit $0 \leq i \leq S - 1$.
Beispiel: CED

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

Lösung

Schritt	**Menge** S	**Menge** $V\S$
0	∅	(C,0)
1	(C,0)	(B,1),(A,4),(E,5)
2	(C,0),(B,1)	(A,**3**),(D,4),(E,5)
3	(C,0),(B,1),(A,3)	(D,4),(E,5),(F,6)
4	(C,0),(B,1),(A,3),(D,4)	(E,5),(F,6)
5	(C,0),(B,1),(A,3),(D,4),(E,5)	(F,6)
6	(C,0),(B,1),(A,3),(D,4),(E,5),(F,6)	∅

c) Geben Sie die Adjazenzmatrix für G an.

Lösung

	A	B	C	D	E	F
A	0	0	0	9	0	3
B	2	0	0	3	0	0
C	4	1	0	0	5	0
D	0	0	0	0	0	0
E	0	0	0	1	0	0
F	0	0	0	1	0	0

d) Ist G topologisch sortierbar? Begründen Sie Ihre Antwort. Im Falle, dass G sortierbar ist, geben Sie eine valide topologische Sortierung an.

Lösung

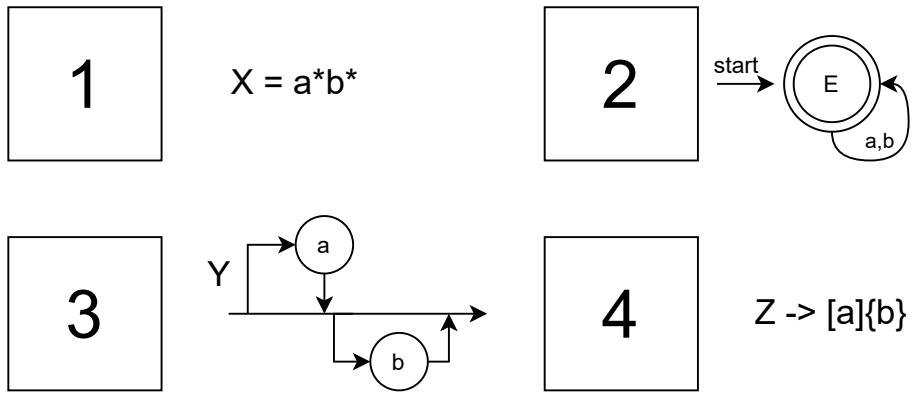
z.B.: C,E,B,A,F,D

Aufgabe 3 - Formale Sprachen

4 + 4.5 + 3.5 = 12 Punkte

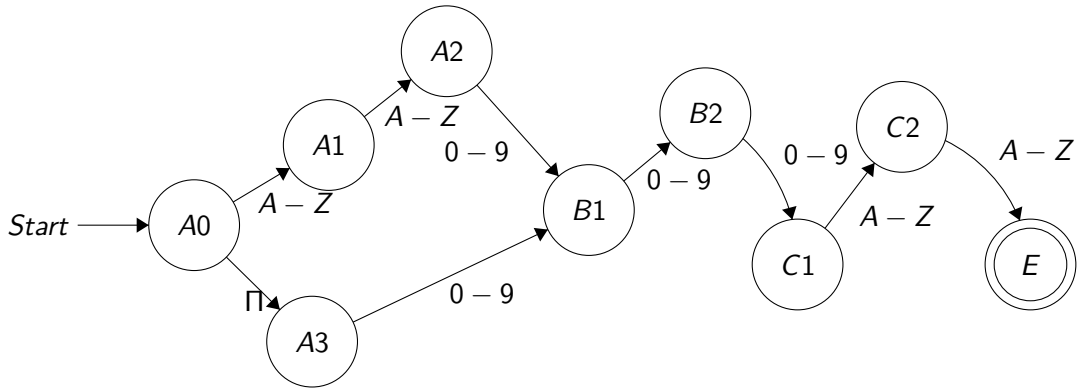
a) Im Folgenden sehen Sie einen regulären Ausdruck, einen endlichen Automaten, ein Syntaxdiagramm und eine EBNF. Markieren Sie, welche davon jeweils die gleiche Sprache erzeugen bzw. beschreiben, indem Sie in die Box eine Zahl eintragen. *Wären Sie zum Beispiel der Meinung, dass alle unterschiedliche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

Lösung



b) Serbische KFZ-Kennzeichen unterscheiden sich maßgeblich von deutschen. Der "Landkreis" in dem das Fahrzeug zugelassen wurde wird immer über 2 Buchstaben identifiziert. Darauf folgt eine immer 3-stellige Nummer auf welche dann wieder 2 Buchstaben folgen. Die einzige Ausnahme für diese Art der Kennzeichen bilden die Nummernschilder der Polizei. Diese haben statt des Zulassungskreises den kyrillischen Buchstaben П für Polizei.
Geben Sie einen **deterministischen** endlichen Automaten an, der die beschriebene Menge an serbischen Kennzeichen erkennt.

Lösung



c) Ein Gebirge sei wie folgt kodiert:

- / stellt einen Aufstieg dar
- \ stellt einen Abstieg dar
- _ repräsentiert ebenes Gelände

Ferner gilt, dass es zu jedem Aufstieg einen Abstieg geben muss (und umgekehrt) und dass ein Gebirge stets mit ebenem Gelände beginnt und endet.
Geben Sie eine Grammatik in BNF-Notation an, welche derartige Gebirge als Wörter zulässt.
Beispiel: Folgendes Gebirge ist ein valides Gebirge: _//_//\\\\/_

Lösung

V = {A,B}
Σ = {/, \, _}
S = {A}
P = {A → _ B _
B → / B \
B → \ B /
B → _ B
B → ε
}

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

Betrachten Sie folgende Klassen:

```java
public class Library {
    //Maps Genretype to Collection of books of that genre
    private Map<String, Collection<Book>> books;
}
```

```java
public class Book {
    public String title;
    protected Stack<Page> pages;
}
```

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

a) Implementieren Sie die Methode `findBookByTitle(...)` der Klasse `Library`, welche alle Bücher der Bibliothek durchsucht und das Buch mit dem übergebenen Titel als `Optional` zurückgibt.
Hinweis: Sie können davon ausgehen, dass Büchernamen eindeutig sind.
Hinweis: `Optionals` von einem Objekt, das möglicherweise `null` ist, können mittels `Optional.ofNullable(object)` erzeugt werden.

— **Lösung**

```
1  public Optional<Book> findBookByTitle(String title) {
2      return books.values()
3        .stream()
4        .flatMap(bookCollection -> bookCollection
5                .stream().filter($ -> $.title.equals(title)))
6        .findFirst();
7  }
```

alternativ:

— **Lösung**

```
1  public Optional<Book> findBookByTitle(String title){
2      for(Collection<Book> c : books.values()){
3          for(Book b : c){
4              if(b.title.equals(title)){
5                  return Optional.ofNullable(b);
6              }
7          }
8      }

9      return Optional.ofNullable(null);
10 }
```

b) Implementieren Sie die Methode `insertBook(...)` der Klasse `Library`, welche ein Buch mit übergebenem Genre in die Menge der Bücher zu diesem Genre hinzufügt. Achten Sie dabei darauf, dass Buchtitel pro Genre eindeutig sein müssen. Sollte ein Buch mit selbem Namen bereits existieren soll dieses mit dem übergebenen Buch ersetzt werden.

Lösung

```
1 public void insertBook(Book book, String type) {
2     String bookTitle = book.title;
3     Collection<Book> filteredBooks = books.get(type)
4       .stream()
5       .filter(aBook -> !aBook.title.equals(bookTitle))
6       .collect(Collectors.toList());
7     filteredBooks.add(book);
8     books.put(type, filteredBooks);
9 }
```

alternativ:

Lösung

```
1  public void insertBook(Book book, String genre){
2          Collection<Book> c = books.get(genre);
3
3          // Entfernen falls vorhanden
4      Book replaceThis = null;
5      for(Book b : c){
6          if(b.title.equals(book.title)){
7              replaceThis = b;
8          }
9      }
10     if(replaceThis != null){
11         c.remove(replaceThis);
12     }
13     // Hinzufuegen
14     c.add(book);
15 }
```

c) Die Implementierung von Büchern mittels eines `Stacks` von `Pages` hat einen entscheidenden Nachteil: um auf Folgeseiten zugreifen zu können, müssen vorherige Seiten verworfen werden. Lösen Sie dieses Problem mit Hilfe eines weiteren Stack-Attributs der Klasse `Book`. Geben Sie hierfür die **Signatur** (Sichtbarkeit, Typ, Name) des Attributs an und **implementieren** Sie die Methode `turnPage(...)`, welche im Buch vor- bzw. zurückblättern kann abhängig davon ob `true` oder `false` übergeben wird. Im Falle, dass nicht mehr geblättert werden kann, soll keine Aktion durchgeführt werden.

— **Lösung**

```
1  // Fügen ein protected Stack<Page> leftPages Attribut hinzu
2  public void turnPage(boolean forward) {
3      Stack<Page> from;
4      Stack<Page> to;
5      if (forward) {
6          from = pages;
7          to = leftPages;
8      } else {
9          from = leftPages;
10         to = pages;
11     }
12     if (!from.empty()) {
13         Page p = from.pop();
14         to.push(p);
15     }
16 }
```

Aufgabe 5 - Java - Graphen **6 + 6 = 12 Punkte**

Betrachten Sie folgende Klasse, die einen Knoten eines gerichteten Graphen implementiert:

```
1 public class Node {
2     int value;
3     HashSet<Node> connected = new HashSet<>();
4 }
```

a) Implementieren Sie die Methode `insertValue(...)` der Klasse `Node`, welche einen neuen `Node`, mit dem `value` `toInsert`, in die `connected` Menge des Knotens einfügt, welcher den Wert `at` in seinem `value` stehen hat. Die Methode soll `true` zurückliefern wenn der Wert eingefügt werden konnte und `false` wenn nicht.
Hinweis: Sie können davon ausgehen, dass der Graph zusammenhängend ist und keine Zyklen enthält.
Hinweis: Rekursion erleichtert Ihnen hier die Implementierung.

— **Lösung**

```
1  public boolean insertValue(int at, int toInsert) {
2      if (this.value == at) {
3          Node newNode = new Node(toInsert);
4          this.connected.add(newNode);
5          return true;
6      }
7      for (Node node : connected) {
8          if (node.insertValue(at, toInsert)) {
9              return true;
10         }
11     }
12     return false;
13 }
```

b) Implementieren Sie die Methode `uniqueValues()` der Klasse `Node` welche überprüft, ob in den vom aktuellen Knoten aus erreichbaren Knoten kein `value` mehrfach vorkommt. Falls kein Wert mehrfach vorkommt, soll `true` zurückgegeben werden, ansonsten `false`.
Hinweis: Sie können davon ausgehen, dass der Graph keine Zyklen enthält.
Hinweis: Eine Hilfsmethode kann hier nützlich sein!

Lösung

```java
public boolean uniqueValues() {
    return uniqueValues(new HashSet<>());
}

public boolean uniqueValues(HashSet<Integer> values) {
    if (values.contains(value)) {
        return false;
    } else {
        // this manipulates the passed hashSet
        // as such each recursive call will contain all checked
        // values afterwards
        values.add(value);
        for (Node node : connected) {
            if (!node.uniqueValues(values)) {
                return false;
            }
        }
        return true;
    }
}
```