Karlsruher Institut für Technologie

Prof. Dr.-Ing. Anne Koziolek
Institute of Information Security and Dependability (KASTEL)
https://sdq.kastel.kit.edu/wiki/Programmieren/
programmieren-vorlesung@cs.kit.edu

# Programmieren – Wintersemester 2024/25

**Abschlussaufgabe 2**
Version 1.0

20 Punkte

| | |
|---|---|
| Ausgabe: | 26.02.2025, ca. 12:00 Uhr |
| Abgabestart: | 12.03.2024, 12:00 Uhr |
| Abgabefrist: | 27.03.2024, 06:00 Uhr |

## Plagiarismus

Es werden nur selbstständig angefertigte Lösungen akzeptiert. Das Einreichen fremder Lösungen, seien es auch nur teilweise Lösungen von Dritten, aus Büchern, dem Internet oder anderen Quellen, ist ein Täuschungsversuch und führt jederzeit (auch nachträglich) zur Bewertung „nicht bestanden“. Ausdrücklich ausgenommen hiervon sind Quelltextschnipsel von den Vorlesungsfolien und aus den Lösungsvorschlägen des Übungsbetriebes in diesem Semester. Alle benutzten Hilfsmittel müssen vollständig und genau angegeben werden. Alles, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, muss deutlich kenntlich gemacht werden. Beachten Sie darüber hinaus die Richtlinien der Fakultät zum Verwenden von Generativer KI [1].

Studierende, die den ordnungsgemäßen Ablauf einer Erfolgskontrolle stören, können von der Erbringung der Erfolgskontrolle ausgeschlossen werden. Ebenso stellt unter anderem die Weitergabe von Teilen von Testfällen oder Lösungen bereits eine Störung des ordnungsgemäßen Ablaufs dar. Auch diese Art von Störungen können ausdrücklich jederzeit zum Ausschluss der Erfolgskontrolle führen. Dies bedeutet ausdrücklich, dass auch nachträglich die Punktzahl reduziert werden kann.

## Kommunikation und aktuelle Informationen

In unseren *FAQs*[2] finden Sie einen Überblick über häufig gestellte Fragen und die entsprechenden Antworten zum Modul „Programmieren“. Bitte lesen Sie diese sorgfältig durch, noch bevor Sie Fragen stellen, und überprüfen Sie diese regelmäßig und eigenverantwortlich auf Änderungen. Beachten Sie zudem die Hinweise im Wiki[3].

In den *ILIAS-Foren* oder auf *Artemis* veröffentlichen wir gelegentlich wichtige Neuigkeiten. Eventuelle Korrekturen von Aufgabenstellungen werden ebenso auf diesem Weg bekannt gemacht. Das aktive Beobachten der Foren wird daher vorausgesetzt.

[1] https://www.informatik.kit.edu/faq-wiki/doku.php?id=generative_ki
[2] https://sdq.kastel.kit.edu/wiki/Programmieren/FAQ
[3] https://sdq.kastel.kit.edu/programmieren/

Überprüfen Sie das Postfach Ihrer *KIT-Mailadresse* regelmäßig auf neue E-Mails. Sie erhalten unter anderem eine Zusammenfassung der Korrektur per E-Mail an diese Adresse. Alle Anmerkungen können Sie anschließend im Online-Einreichungssystem[4] einsehen.

## Bearbeitungshinweise

Bitte beachten Sie, dass das erfolgreiche Bestehen der verpflichtenden Tests für eine erfolgreiche Abgabe von Abschlussaufgabe 2 notwendig ist. Ihre Abgabe wird automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Sie müssen zuerst die verpflichtenden Tests bestehen, bevor die anderen Tests ausgewertet werden können. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

- Achten Sie auf fehlerfrei kompilierenden Programmcode.
- Verwenden Sie ausschließlich *Java SE 17*.
- Sofern in einer Aufgabe nicht ausdrücklich anders angegeben, verwenden Sie keine Elemente der Java-Bibliotheken. Ausgenommen ist die Klasse `java.util.Scanner` und alle Elemente aus den folgenden Paketen: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.nio.file`, `java.nio.charset`.
- Achten Sie darauf, nicht zu lange Zeilen, Methoden und Dateien zu erstellen. Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 140 Zeichen einhalten.
- Halten Sie alle Whitespace-Regeln ein.
- Halten Sie alle Regeln zu Variablen-, Methoden- und Paketbenennung ein.
- Wählen Sie geeignete Sichtbarkeiten für Ihre Klassen, Methoden und Attribute.
- Nutzen Sie nicht das `default`-Package.
- `System.exit()`, `Runtime.exit()` oder ähnliches dürfen nicht verwendet werden.
- Halten Sie die Regeln zur Javadoc-Dokumentation ein.
- Halten Sie auch alle anderen Checkstyle-Regeln ein.

Diese folgenden Bearbeitungshinweise sind relevant für die Bewertung Ihrer Abgabe. Dennoch wird Ihre Abgabe durch das Abgabesystem *nicht* automatisch mit null Punkten bewertet, falls eine der nachfolgenden Regeln verletzt ist. Orientieren Sie sich zudem an den Bewertungskriterien im Wiki.

- Fügen Sie außer Ihrem u-Kürzel keine weiteren persönlichen Daten zu Ihren Abgaben hinzu.
- Beachten Sie, dass Ihre Abgaben sowohl in Bezug auf objektorientierte Modellierung als auch Funktionalität bewertet werden. Halten Sie die Hinweise zur Modellierung im Wiki ein.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.

[4] `https://artemis.praktomat.cs.kit.edu/`

- Die Kommentare sollen einheitlich in englischer oder deutscher Sprache verfasst werden.
- Geben Sie im Javadoc-Autoren-Tag nur Ihr u-Kürzel an.
- Wählen Sie aussagekräftige Namen für alle Ihre Bezeichner.

## Checkstyle

Das Online-Einreichungssystem überprüft Ihre Quelltexte während der Abgabe automatisiert auf die Einhaltung der Checkstyle-Regeln. Es gibt speziell markierte Regeln, bei denen das Online-Einreichungssystem die Abgabe mit null Punkten bewertet, da diese Regeln verpflichtend einzuhalten sind. Andere Regelverletzungen können zu Punktabzug führen. Sie können und sollten Ihre Quelltexte bereits während der Entwicklung auf die Regeleinhaltung überprüfen. Das Programmieren-Wiki beschreibt, wie Checkstyle verwendet werden kann.

## Abgabehinweise

Die Abgabe im Online-Einreichungssystem wird am 12.03.2024, 12:00 Uhr, freigeschaltet. Achten Sie unbedingt darauf, Ihre Dateien im Einreichungssystem bei der richtigen Aufgabe vor Ablauf der Abgabefrist am 27.03.2024, 06:00 Uhr, hochzuladen. Beginnen Sie frühzeitig mit dem Einreichen, um Ihre Lösung dahingehend zu testen, und verwenden Sie das Forum, um eventuelle Unklarheiten zu klären. Falls Sie mit Git abgeben, *muss immer* auf den `main`-Branch gepusht werden.

- Geben Sie online Ihre `*.java`-Dateien zur Aufgabe A in Einzelarbeit mit der entsprechenden Ordnerstruktur im zugehörigen Verzeichnis ab.

## Wiederverwendung von Lösungen

Falls Sie für die Bearbeitung der Abschlussaufgaben oder Übungsblätter Beispiellösungen aus diesem Semester wiederverwenden, *müssen* Sie in die entsprechenden Klassen "Programmieren-Team" ins Autor-Tag eintragen. Dies ist nötig, um die Checkstyle-Kriterien zu erfüllen.

## Prüfungsmodus in Artemis

Wenn Sie mit einer Abschlussaufgabe fertig sind, können Sie diese frühzeitig abgeben. Dazu dient Schaltfläche „Vorzeitig abgeben“. Nach der frühzeitigen Abgabe einer Abschlussaufgabe können Sie keine Änderungen an Ihrer Abgabe mehr vornehmen.

# Aufgabe A: Empfehlungssystem

## A.1 Einführung

In dieser Aufgabe programmieren Sie ein *Produkt-Empfehlungssystem*, das in keinem größeren Online-Shop fehlen darf. Das Empfehlungssystem erlaubt seinen Benutzern ein (für sie interessantes) *Referenzprodukt* zu benennen, wovon ausgehend verwandte Produkte ermittelt werden. Diese werden dem Benutzer als *Produktempfehlungen* ausgegeben.

Die Abbildung von einem Referenzprodukt auf passende Produktempfehlungen wird durch *Empfehlungsstrategien* bestimmt. Der Benutzer kann aus verschiedenen Empfehlungsstrategien auswählen, oder diese miteinander kombinieren, um das Empfehlungssystem an seine Bedürfnisse anzupassen.

Ein besonderes Merkmal des Empfehlungssystem ist die Verwaltung des Datenbestandes in Form eines *Graphen*. Der Datenbestand beinhaltet *Produkte* und *Kategorien*, zwischen denen bestimmte *Beziehungen* aufgebaut werden können. Eine Beziehung zwischen zwei Produkten *p1* und *p2* könnte beispielsweise ausdrücken, dass *p2* Nachfolger von *p1* ist. Interessiert sich der Benutzer für *p1*, erscheint es lohnenswert, *p2* als Produktempfehlung auszugeben.

## A.2 Datenbestand

Der Datenbestand ist als *gerichteter* Graph organisiert. Produkte und Kategorien werden durch **Knoten** modelliert. Beziehungen zwischen Produkten und Kategorien werden als **gerichtete Kanten** modelliert.

### A.2.1 Produkte und Kategorien

Knoten im Graphen repräsentieren Produkte und Kategorien. Produkte und Kategorien besitzen jeweils einen *Namen*. Produkte tragen außerdem eine eindeutige *Identifikationsnummer*. Gültige Produkt- bzw. Kategorienamen beschreibt der reguläre Ausdruck `[a-zA-Z0-9]+`. Gültige Identifikationsnummern sind positive Integer-Zahlen inklusive 0.

Weiterhin sollen im Rahmen der Aufgabe auch Produkt- und Kategoriennamen eindeutig sein. Das bedeutet, wenn zwei Knoten $n1$ und $n2$ den gleichen Namen tragen, muss gelten: $n1 = n2$.

Knotennamen, die sich nur durch Groß-/Kleinschreibung unterscheiden, werden als identisch betrachtet. Bei zwei Knoten mit den Namen *libreoffice* und *libreOffice* handelt es sich also um den selben Knoten.

### A.2.2 Beziehungen

Kanten im Graphen setzen Produkte und Kategorien zueinander in Beziehung. Dafür existieren folgende *Beziehungstypen*.

Beachten Sie im Folgenden, dass jede Art von Beziehung genau eine *Umkehrbeziehung* besitzt.

**contains:** $n1$ *contains* $n2$ drückt aus, dass Kategorie $n1$ das Produkt oder die Kategorie $n2$ enthält. Beispiel: `software contains operatingsystem`. Dabei ist $n1$ stets eine Kategorie und $n2$ entweder ein Produkt oder eine Kategorie. Sind beide Knoten vom Typ Kategorie, so drückt diese Beziehung eine Spezialisierung der Oberkategorie $n1$ durch die Unterkategorie $n2$ aus. (Umkehrbeziehung: *contained-in*)

**contained-in:** $n1$ *contained-in* $n2$ drückt aus, dass Produkt oder Kategorie $n1$ in Kategorie $n2$ enthalten ist. Beispiel: `operatingsystem contained-in software`. Dabei ist $n1$ ein Produkt oder eine Kategorie und $n2$ stets eine Kategorie. Sind beide Knoten vom Typ Kategorie, so drückt diese Beziehung eine Spezialisierung der Oberkategorie $n2$ durch die Unterkategorie $n1$ aus. (Umkehrbeziehung: *contains*)

**part-of:** $n1$ *part-of* $n2$ drückt aus, dass Produkt $n1$ ein Teil von Produkt $n2$ ist, wobei $n2$ eine Sammlung von Produkten repräsentiert. Beispiel: `writer part-of libreoffice`. Dabei sind $n1$ und $n2$ stets Produkte. (Umkehrbeziehung: *has-part*)

**has-part:** $n1$ *has-part* $n2$ drückt aus, dass Produkt $n1$ eine Sammlung von Produkten repräsentiert, die Produkt $n2$ beinhaltet. Beispiel: `libreoffice has-part writer`. Dabei sind $n1$ und $n2$ stets Produkte. (Umkehrbeziehung: *part-of*)

**successor-of:** $n1$ *successor-of* $n2$ drückt aus, dass Produkt $n1$ das (direkte) Nachfolgeprodukt von $n2$ ist. Beispiel: `centos7 successor-of centos6`. Dabei sind $n1$ und $n2$ stets Produkte. (Umkehrbeziehung: *predecessor-of*)

**predecessor-of:** $n1$ *predecessor-of* $n2$ drückt aus, dass Produkt $n1$ das (direkte) Vorgängerprodukt von $n2$ ist. Beispiel: `centos6 predecessor-of centos7`. Dabei sind $n1$ und $n2$ stets Produkte. (Umkehrbeziehung: *successor-of*)

Wird dem Graphen mittels einer Kante $e = (n1, n2, b)$ die Beziehung $b$ von Knoten $n1$ zu Knoten $n2$ hinzugefügt, soll zusätzlich die Umkehrbeziehung $\bar{b}$ mittels der Kante $e = (n2, n1, \bar{b})$ hergestellt werden, sofern diese nicht schon vorhanden sind.

Denn, redundante Kanten sind zu vermeiden: Für zwei Kanten $e1 = (n11, n12, b1)$ und $e2 = (n21, n22, b2)$ muss $e1 = e2$ gelten, genau dann wenn $n11 = n21$ und $n12 = n22$ und $b1 = b2$.

Weiterhin kann ein Knoten nicht zu sich selbst in Beziehung gesetzt werden: Für jede Kante $e = (n1, n2, b)$ gilt $n1 \neq n2$.

Beachten sie, dass es beliebig viele Beziehungen zwischen Knoten geben kann, so kann zum Beispiel ein Produkt der Nachfolger von mehreren Produkten sein, oder ein Produkt aus mehreren Produkten bestehen.

In Abbildung A.1 finden sie den Graphen eines Beispiel-Datenbestandes.

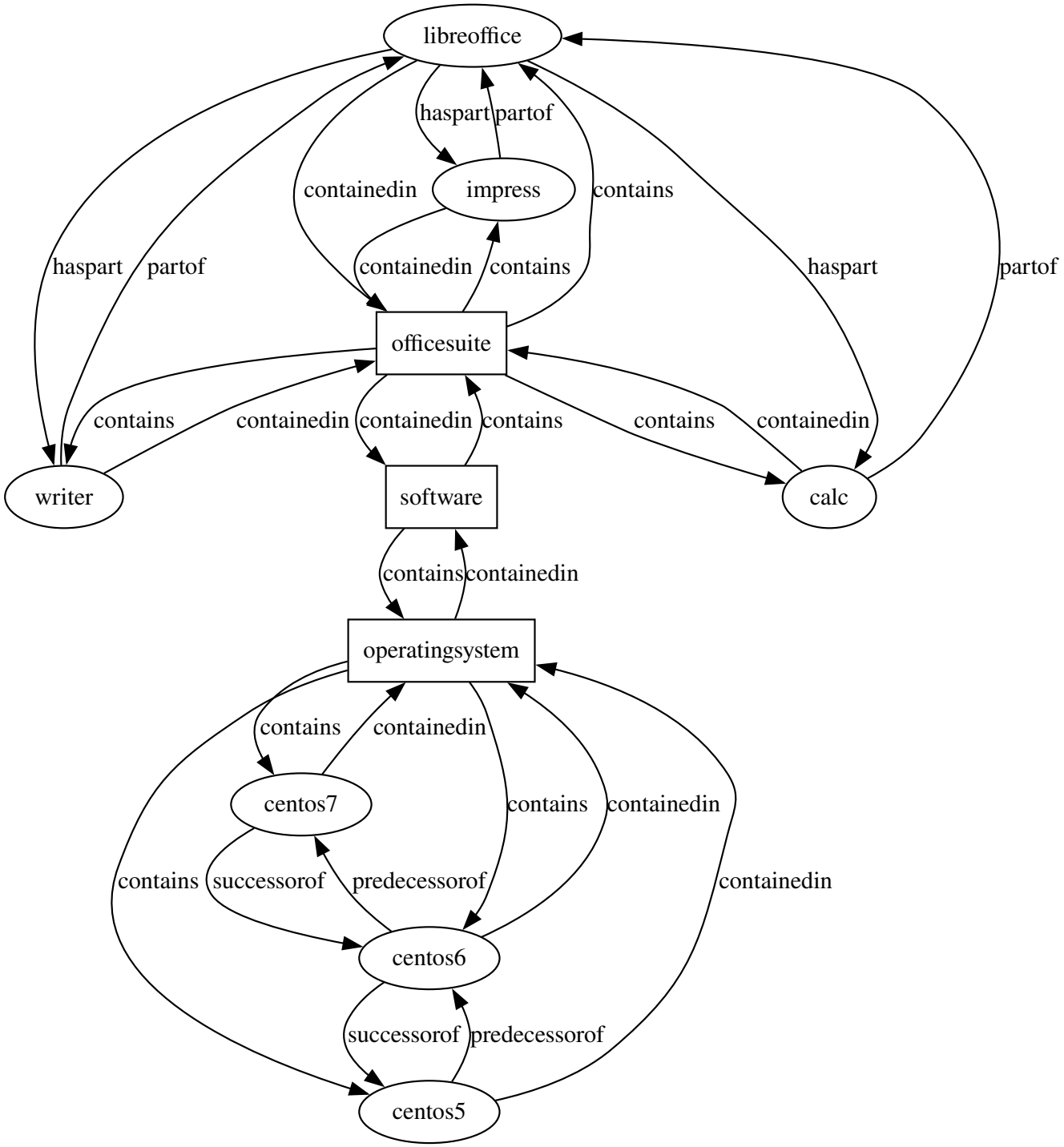



Abbildung A.1: Graph eines Beispiel-Datenbestandes

## A.3 Empfehlungsstrategien

Eine Empfehlungsstrategie berechnet ausgehend von einem Referenzprodukt eine Menge von verwandten Produkten. Dazu operiert jede Empfehlungsstrategie auf der Datenbasis und macht sich insbesondere die durch Beziehungen gegebenen semantischen Informationen zunutze.

Im Folgenden werden zunächst drei *einfache Empfehlungsstrategien* spezifiziert, bevor beschrieben wird, wie sich diese zu *zusammengesetzten Empfehlungsstrategien* kombinieren lassen.

### A.3.1 Einfache Empfehlungsstrategien

Für alle im Folgenden eingeführten Empfehlungsstrategien gilt, dass das Referenzprodukt niemals Teil der empfohlenen Produktmenge ist und gegebenenfalls entfernt werden muss bevor die Empfehlungen zurückgeliefert werden. Weiterhin sollen ausschließlich Produkte empfohlen werden, niemals Kategorien.

**Geschwisterprodukte (S1)** liefert für ein bestimmtes Referenzprodukt $p_r$ alle Produkte, die zusammen mit $p_r$ in einer **direkten** *contained-in*-Beziehung zu einer gemeinsamen Oberkategorie stehen. Nehmen wir am Beispiel der Abbildung A.1 an, dass `centos6` das Referenzprodukt ist. Diese Strategie empfehlt dann die Produkte `centos5` und `centos7`.

**Nachfolgeprodukte (S2)** liefert für ein bestimmtes Referenzprodukt $p_r$ alle **direkten und indirekten** Nachfolgeprodukte von $p_r$ durch Verfolgung der *predecessor-of*-Beziehung beginnend mit $p_r$. Nehmen wir am Beispiel der Abbildung A.1 an, dass `centos5` das Referenzprodukt ist. Diese Strategie empfiehlt dann die Produkte `centos6` und `centos7`. Beachten Sie, dass ein Produkt mehrere direkte Nachfolger haben kann. Ein Beispiel dafür ist die Office-Suite Open-Office, aus der die Produkte Apache OpenOffice sowie LibreOffice als zwei unabhängige Nachfolger hervorgingen (nicht in Abbildung A.1 dargestellt).

**Vorgängerprodukte (S3)** liefert für ein bestimmtes Referenzprodukt $p_r$ alle **direkten und indirekten** Vorgängerprodukte von $p_r$ durch Verfolgung der *successor-of*-Beziehung beginnend mit $p_r$. Nehmen wir am Beispiel der Abbildung A.1 an, dass `centos7` das Referenzprodukt ist. Diese Strategie empfiehlt dann die Produkte `centos6` und `centos5`. Beachten Sie, dass ein Produkt mehrere direkte Vorgänger haben kann. Um beim Beispiel Open-Office zu bleiben, etwa wenn der (unwahrscheinliche) Fall eintreten würde, dass die Projekte Apache OpenOffice und LibreOffice ihre Kräfte wieder in einem gemeinsam Produkt bündeln (nicht in Abbildung A.1 dargestellt).

### A.3.2 Zusammengesetze Empfehlungsstrategien

Eine zusammengesetzte Empfehlungsstrategie kombiniert die Empfehlungen zweier anderer Empfehlungsstrategien. Seien $R1$ und $R2$ Produktmengen, die etwa durch zwei einfache Empfehlungsstrategien gewonnen wurden. Die Kombination von $R1$ und $R2$ kann dann auf zweierlei Weise erfolgen:

**Schnittmengenbildung:** $R1 \cap R2 = \{x | x \in R1 \wedge x \in R2\}$

**Vereinigungsmengenbildung:** $R1 \cup R2 = \{x | x \in R1 \vee x \in R2\}$

Dabei können die Produktmengen $R1$ und $R2$ selbst wiederum aus einer zusammengesetzten Empfehlungsstrategie hervorgegangen sein. Nähere Informationen hierzu liefert A.7.1.6

## A.4 Datenbestand-Datei

Ihr Programm liest eine Textdatei ein, die die im Datenbestand zu speichernden Produkte, Kategorien sowie deren Beziehungen beinhaltet.

### A.4.1 Grammatik

Die Datenbestand-Datei besteht aus einer oder mehreren Zeilen. Jede Zeile stellt durch ein Prädikat eine Beziehung zwischen einem Subjekt und einem Objekt her. Das Format einer solchen Zeile ist durch folgende BNF-Grammatik [5] gegeben. Terminalsymbole sind im `Typewriter`-Stil geschrieben. Abweichend von der BNF sind die Nichtterminalsymbole *productid*, *productname* sowie *categoryname* jeweils durch einen regulären Ausdruck spezifiziert, der die Menge der gültigen Terminalsymbole ausdrückt.

**BNF-Grammatik einer Zeile der Datenbestand-Datei**

*line* ::= *subject predicate object*
*subject* ::= *product* | *categoryname*
*object* ::= *product* | *categoryname*
*predicate* ::= `contains` | `contained-in` | `part-of` | `has-part` | `successor-of` | `predecessor-of`
*product* ::= *productname* (`id=` *productid*)
*productid* ::= `[0-9]+`
*productname* ::= `[a-zA-Z0-9]+`
*categoryname* ::= `[a-zA-Z0-9]+`

Ergänzend zu obiger Grammatik sollen unnötige Leerzeichen ignoriert werden anstatt eine Fehlermeldung auslösen und zwar nach folgenden Regeln:

Innerhalb der Ableitungsregel *line* sind beide Leerzeichen erforderlich; zusätzliche Leerzeichen dürfen vor/nach *subject*, *predicate* und *object* stehen. Innerhalb der Ableitungsregel *product* ist keines dargestellten Leerzeichen erforderlich; zusätzliche Leerzeichen dürfen vor/nach *productname*, (, `id`, =, *productid*, ) stehen. Alle anderen Ableitungsregeln folgen strikt der entsprechenden BNF-Spezifikation.

So soll beispielsweise eine Zeile der folgenden Form akzeptiert werden:

```
centos7 ( id = 107 ) contained-in operatingsystem
```

Folgende Bestandteile dürfen jedoch absolut keine Leerzeichen enthalten: *predicate*, *productid*, *productname*, *categoryname*.

Denken Sie beim Aufbau des Graphen daran, Umkehrbeziehungen herzustellen (vgl. Unterunterabschnitt A.2.2), falls diese nicht bereits in der Datenbestand-Datei beschrieben sind.

Tritt beim Einlesen der Datei ein Fehler auf, sei es ein Ein-/Ausgabefehler beim Lesen der Datei, ein Syntaxfehler, oder ein semantischer Fehler, so wird eine Fehlermeldung ausgegeben. Ein Syntaxfehler besteht, wenn die Datei nicht gemäß obenstehender Spezifikation geformt ist. Ein semantischer

[5] Backus-Naur-Form, siehe z.B. `http://de.wikipedia.org/wiki/Backus-Naur-Form`

Fehler besteht, wenn beispielsweise eine ungültige Beziehung aufgebaut wird, etwa zwei Kategorien, die mittels der *successor-of*-Beziehung verbunden werden sollen.

### A.4.2 Beispiel

Folgendes Beispiel präsentiert eine wohlgeformte Datenbestand-Datei zum Aufbau des in Abbildung A.1 dargestellten Graphen:

**Beispiel**

```
1  CentOS5 ( id= 105) contained-in operatingSystem
2  centOS6 ( id = 106) contained-in OperatingSystem
3  operatingSystem contains centos7 ( id = 107 )
4  operatingsystem contained-in Software
5  CentOS7 (id=107) successor-of centos6(id=106)
6  CentOS5 (id=105) predecessor-of centos6(id=106)
7  writer (id=201) contained-in officesuite
8  calc (id=202) contained-in officesuite
9  impress (id=203) contained-in officesuite
10 officesuite contained-in software
11 LibreOffice (id=200) contained-in officesuite
12 writer (id=201) part-of LibreOffice (id=200)
13 calc (id=202) part-of libreoffice (id=200)
14 libreoffice (id=200) has-part impress (id=203)
```

Der in Abbildung A.1 abgebildete Graph kann mit unterschiedlichen Eingaben erzeugt werden. Die obige Eingabe ist somit nur eine Beispieleingabe für den Aufbau dieses Graphen.

## A.5 DOT-Notation

Die DOT-Notation wird zur Beschreibung von Graphen verwendet. Ein besonderes Merkmal dieser Notation ist, dass sie sowohl von Menschen, als auch von Maschinen lesbar ist[6].

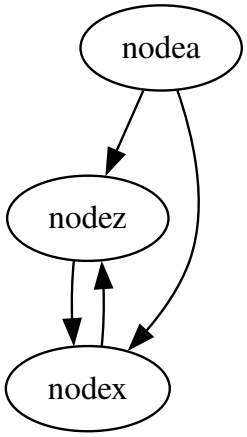
### A.5.1 Gerichteter Graph

Die Notation eines gerichteten Graphen beginnt mit dem Schlüsselwort `digraph`. Die Beschreibung von Knoten und Kanten erfolgt innerhalb von zwei geschweiften Klammern. Die Namen der Knoten sind beliebig wählbar. Eine gerichtete Kante wird durch `->` definiert. Im Folgenden ist ein Graph in DOT-Notation, sowie seine entsprechende visuelle Darstellung abgebildet.

[6]Siehe hierzu `http://www.graphviz.org/content/dot-language`, sowie `http://en.wikipedia.org/wiki/DOT_(graph_description_language)`

**Beispiel**

```
1 digraph {
2 nodea -> nodez
3 nodea -> nodex
4 nodez -> nodex
5 nodex -> nodez
6 }
```




### A.5.2 Weitere Merkmale der DOT-Notation

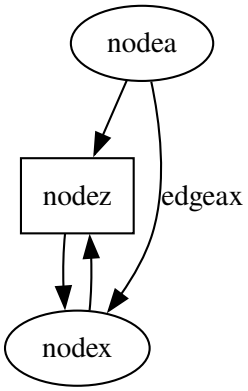
Um ein Knoten als ein Rechteck darstellen zu können, benutzen Sie den Namen des Knotens und das Schlüsselwort `[shape=box]`.

Um Kanten mit einer Beschriftung zu versehen, nutzen Sie `[label=edgename]`.

Folgendes Beispiel illustriert diese Fälle:

**Beispiel**

```
1 digraph {
2 nodez [shape=box]
3 nodea -> nodez
4 nodea -> nodex [label=edgeax]
5 nodez -> nodex
6 nodex -> nodez
7 }
```




## A.6 Hinweise zur Implementierung

Nach dem Start nimmt Ihr Programm über die Kommandozeile mittels der Standardeingabe Eingaben entgegen, die im Folgenden näher spezifiziert werden. Alle Befehle werden auf dem aktuellen Zustand des Programms ausgeführt. Nach Abarbeitung einer Eingabe wartet Ihr Programm auf weitere Eingaben, bis das Programm irgendwann durch die Eingabe der Zeichenfolge `quit` beendet wird.

Im Folgenden werden Eingabezeilen mit einer schließenden spitzen Klammer (`>`) gefolgt von einem Leerzeichen eingeleitet. Diese beiden Zeichen sind kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung von Ein- und Ausgabezeilen.

Achten Sie darauf, dass durch die Ausführung der folgenden Befehle die gegebenen semantischen und syntaktischen Spezifikationen nicht verletzt werden und geben Sie bei Verletzung der Spezifikationen immer eine aussagekräftige Fehlermeldung aus. Wenn die Benutzereingabe nicht dem vorgegebenen

Format entspricht, ist auch eine Fehlermeldung auszugeben. Nach der Ausgabe einer Fehlermeldung soll das Programm wie erwartet fortfahren und wieder auf die nächste Eingabe warten. Die Fehlermeldung sollte so geformt sein, dass für den Benutzer erkenntlich ist, warum eine Eingabe abgelehnt wurde.

Da wir automatische Tests Ihrer interaktiven Benutzerschnittstelle durchführen, müssen die Ausgaben exakt den Vorgaben entsprechen. Insbesondere sollen sowohl Klein- und Großbuchstaben als auch die Leerzeichen und Zeilenumbrüche genau übereinstimmen. Setzen Sie nur die in der Aufgabenstellung angegebenen Informationen um. Geben Sie auch keine zusätzlichen Informationen aus. Bei Fehlermeldungen dürfen Sie den englischsprachigen Text frei wählen, er sollte jedoch sinnvoll sein. Jede Fehlermeldung muss aber mit `Error,` beginnen und darf keine Sonderzeichen, wie beispielsweise Zeilenumbrüche oder Umlaute, enthalten.

Wenn nicht anders angegeben, ist für Eingabe immer die Standardeingabe `System.in` zu verwenden. Wenn nicht anders angegeben, ist für Ausgaben immer die Standardausgabe `System.out` zu verwenden. Für Fehlermeldungen kann anstelle der Standardausgabe optional die Standardfehlerausgabe `System.err` verwendet werden. Weisen Sie diese Standardeingabe und -ausgabe niemals neu zu.

Beachten Sie, dass bei der Beschreibung der Eingabe- und Ausgabeformate die Wörter zwischen spitzen Klammen für Platzhalter stehen, die bei der konkreten Ein- und Ausgabe durch Werte ersetzt werden. Diese eigentlichen Werte enthalten bei der Ein- und Ausgabe keine spitzen Klammern. Vergleichen Sie hierzu auch den Beispielablauf.

## A.7 Funktionalität

Im Folgenden wird die erforderliche Funktionalität des Systems beschrieben:

### A.7.1 Befehle

#### A.7.1.1 Der `load database`-Befehl

Lädt eine spezifizierte Datenbestands-Datei und parsed deren Inhalt. Zu Testzwecken soll der Befehl den Inhalt der Datenbestands-Datei Verbatim ausgeben. Wird der Befehl mehrfach ausgeführt, so wird, falls der neue Datenbestand gültig ist, der alte überschrieben.

**Eingabe** `load␣database␣path`

**Ausgabe** Verbatim Inhalt der Datei

**Beispielinteraktion**

```
> load database database.txt
CentOS5 ( id= 105) contained-in operatingSystem
centOS6 ( id = 106) contained-in OperatingSystem
operatingSystem contains centos7 ( id = 107 )
operatingsystem contained-in Software
CentOS7 (id=107) successor-of centos6(id=106)
CentOS5 (id=105) predecessor-of centos6(id=106)
```

#### A.7.1.2 Der `quit`-Befehl

Der parameterlose Befehl ermöglicht es das Programm vollständig zu beenden. Beachten Sie, dass hierfür keine Methoden wie `System.exit()` oder `Runtime.exit()` verwendet werden dürfen.

**Eingabe** `quit`

**Beispielinteraktion**

```
1 > quit
```

#### A.7.1.3 Der `add`-Befehl

Der `add` Befehl fügt eine Beziehung dem Graphen hinzu. Das Eingabeformat ist äquivalent zu dem der Datenbestands-Datei. Achten sie darauf, dass jede Beziehung eine Umkehrbeziehung hat und eine Beziehung nicht doppelt hinzugefügt werden darf.

**Eingabe** `add␣subject␣predicate␣object`

**Beispielinteraktion**

```
1 > add operatingSystem contains centos7 ( id = 107 )
```

#### A.7.1.4 Der `remove`-Befehl

Der `remove` Befehl entfernt eine Beziehung zwischen zwei Knoten, falls diese davor Bestand. Das Eingabeformat ist äquivalent zu dem der Datenbestands-Datei. Achten sie darauf, dass die Umkehrbeziehung auch entfernt wird. Sollte nach Entfernen der Beziehung ein Knoten keine Beziehung mehr zu einem anderen Knoten haben, so soll dieser auch entfernt werden.

**Eingabe** `remove␣subject␣predicate␣object`

**Beispielinteraktion**

```
1 > remove operatingSystem contains centos7 ( id = 107 )
```

#### A.7.1.5 Der `nodes`-Befehl

Der `nodes`-Befehl gibt eine Liste aller Knoten aus.

Die Knoten werden Leerzeichen-separiert in einer einzigen Zeile ausgegeben. Ein *Produktknoten* wird dargestellt durch dessen Namen in Kleinschreibung, gefolgt von einem Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Ein *Kategorieknoten* wird dargestellt durch dessen Namen in Kleinschreibung.

Die Knoten werden aufsteigend sortiert nach Produkt- bzw. Kategorienamen ausgegeben.

**Eingabe** `nodes`

**Ausgabe** `productname:productid␣categoryname`

**Beispielinteraktion**

```
1 > nodes
2 calc:202 centos5:105 libreoffice:200 officesuite operatingsystem
```

#### A.7.1.6 Der `edges`-Befehl

Der `edges`-Befehl gibt eine Liste aller Kanten aus.

Die Kanten werden zeilenweise ausgegeben, wobei jede Kante in einer separaten Zeile steht. Jede dieser gerichteten Kanten besitzt im Rahmen dieser Aufgabe drei Bestandteile: Quellknoten, Beziehungstyp, Zielknoten. Quell- und Zielknoten werden jeweils dargestellt durch deren Namen in Kleinschreibung. Falls es sich um ein Produkt handelt folgt ein Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Der Beziehungstyp wird eingeleitet durch die Zeichen `-[`, gefolgt von dem Namen des Beziehungstyps in Kleinschreibung, gefolgt von den Zeichen `]->`.

Die Kanten werden aufsteigend sortiert, nach Quellnamen, dann Zielnamen und schließlich Prädikat, ausgegeben. Bei Prädikaten zählt die Reihenfolge aus Unterunterabschnitt A.2.2.

**Eingabe** `edges`

**Ausgabe** `subject-[predicate]->object`

**Beispielinteraktion**

```
1 > edges
2 calc:202-[part-of]->libreoffice:200
3 calc:202-[contained-in]->officesuite
```

#### A.7.1.7 Der `recommend`-Befehl

Mithilfe des `recommend`-Befehls kann sich der Benutzer Produkte empfehlen lassen. Dabei wird das jeweilige Referenzprodukt durch dessen Produktidentifikationsnummer repräsentiert.

Ein gültiger `recommend`-Befehl ist wie folgt aufgebaut. Wie bereits zuvor werden reguläre Ausdrücke als BNF-Erweiterung eingesetzt und Terminalsymbole sind im `Typewriter`-Stil gesetzt.

**BNF-Grammatik des recommend-Befehls**

*command* ::= recommend *term*
*term* ::= *final* | `INTERSECTION`(*term*, *term*) | `UNION`(*term*, *term*)
*final* ::= *strategy productid*
*strategy* ::= `S1` | `S2` | `S3`
*productid* ::= `[0-9]+`

Ergänzend zu obiger Grammatik sollen unnötige Leerzeichen (nur) innerhalb der Nichtterminale *term* sowie *final* ignoriert werden anstatt eine Fehlermeldung auszulösen. So soll beispielsweise eine Eingabe der folgenden Form akzeptiert werden:

```
recommend UNION( S1 105 , S3 107)
```

Implementieren Sie für die Verarbeitung des `recommend`-Befehls einen **Recursive Descent Parser** (dt.: *Rekursiver-Abstieg-Zerteiler*) wie in der Vorlesung eingeführt. Überlegen Sie sich geeignete Datenstrukturen für die geparsten Terme (Final-Term, Intersection-Term, Union-Term).

Die empfohlenden Produkte werden Leerzeichen-separiert in einer einzigen Zeile ausgegeben. Jedes Produkt wird dargestellt durch dessen Namen in Kleinschreibung, gefolgt von einem Doppelpunkt, gefolgt von der Produktidentifikationsnummer. Die Produkte werden aufsteigend sortiert nach Produktnamen ausgegeben. Ist die Menge der empfohlenen Produkte leer, so wird eine leere Zeile ausgegeben.

**Eingabe** Siehe BNF-Grammatik des recommend-Befehls

**Ausgabe** `productname:productid␣productname:productid`

**Beispielinteraktion**

```
1 > recommend S1 105
2 centos6:106 centos7:107
3 > recommend UNION(S1 105,S3 107)
4 centos5:105 centos6:106 centos7:107
```

#### A.7.1.8 Der `export`-Befehl

Der `export`-Befehl gibt den Graphen des eingelesenen Datenbestands in DOT-Notation auf die Konsole aus.

Details zur DOT-Notation finden Sie in Unterabschnitt A.5. Nutzen Sie für den `export`-Befehl bitte ausschließlich die Sprachkonstrukte aus Unterabschnitt A.5. Andernfalls könnten die automatischen Tests fehlschlagen.

Die Ausgabe beginnt mit der Zeile `digraph {` und endet mit der Zeile `}`.

Dazwischen enthält jede Zeile entweder eine Kante in DOT-Notation oder die Definition eines Kategorieknotens. Das Label einer Kante entspricht ihrem Beziehungstyp, wobei Bindestriche entfernt werden. Gültige Kantenlabel sind beispielsweise `containedin` oder `successorof`.

Produktknoten mit eingehenden und/oder ausgehenden Kanten müssen nicht in einer separaten Zeile beschrieben werden, da sie indirekt durch die Kanten dargestellt werden. Kategorieknoten hingegen sollen als Rechtecke dargestellt werden. Dazu muss jeder Kategorieknoten in einer separaten Zeile ausgegeben werden.

Die Ausgabe erfolgt in zwei Schritten:

1. Zunächst kommen die Zeilen mit Kanten aufsteigend sortiert, zuerst nach Quellnamen, dann nach Zielnamen und schließlich nach dem Prädikat. Bei Prädikaten zählt die Reihenfolge aus Unterunterabschnitt A.2.2.
2. Anschließend folgen die Zeilen der Kategorieknoten, die ebenfalls aufsteigend sortiert werden.

Sowohl Produktknoten als auch Kategorieknoten werden durch ihren Namen dargestellt.

**Eingabe** `export`

**Ausgabe** Graph in DOT-Notation

**Beispielinteraktion**

```
1  > export
2  digraph {
3  calc -> officesuite [label=containedin]
4  centos5 -> operatingsystem [label=containedin]
5  centos6 -> operatingsystem [label=containedin]
6  centos7 -> operatingsystem [label=containedin]
7  libreoffice -> impress [label=haspart]
8  writer -> libreoffice [label=partof]
9  writer -> officesuite [label=containedin]
10 officesuite [shape=box]
11 operatingsystem [shape=box]
12 software [shape=box]
13 }
```

## A.8 Beispielinteraktion

Die Zeilennummern und die Trennlinie sind kein Bestandteil der Benutzerschnittstelle, sie dienen lediglich zur Orientierung für die gegebene Beispielinteraktion. Die Eingabezeilen werden mit einer schließenden spitzen Klammer (`>`) gefolgt von einem Leerzeichen eingeleitet, diese beiden Zeichen sind ebenfalls kein Bestandteil des eingegebenen Befehls, sondern dienen der Unterscheidung zwischen Ein- und Ausgabezeilen.

**Beispielinteraktion**

```
%> java -jar Recommender.jar
> load database database.txt
CentOS5 ( id= 105) contained-in operatingSystem
centOS6 ( id = 106) contained-in OperatingSystem
operatingSystem contains centos7 ( id = 107 )
operatingsystem contained-in Software
CentOS7 (id=107) successor-of centos6(id=106)
CentOS5 (id=105) predecessor-of centos6(id=106)
writer (id=201) contained-in officesuite
calc (id=202) contained-in officesuite
impress (id=203) contained-in officesuite
officesuite contained-in software
LibreOffice (id=200) contained-in officesuite
writer (id=201) part-of LibreOffice (id=200)
calc (id=202) part-of libreoffice (id=200)
libreoffice (id=200) has-part impress (id=203)
> export
digraph {
calc -> libreoffice [label=partof]
calc -> officesuite [label=containedin]
centos5 -> centos6 [label=predecessorof]
centos5 -> operatingsystem [label=containedin]
centos6 -> centos5 [label=successorof]
centos6 -> centos7 [label=predecessorof]
centos6 -> operatingsystem [label=containedin]
centos7 -> centos6 [label=successorof]
centos7 -> operatingsystem [label=containedin]
impress -> libreoffice [label=partof]
impress -> officesuite [label=containedin]
libreoffice -> calc [label=haspart]
libreoffice -> impress [label=haspart]
libreoffice -> officesuite [label=containedin]
libreoffice -> writer [label=haspart]
officesuite -> calc [label=contains]
officesuite -> impress [label=contains]
officesuite -> libreoffice [label=contains]
officesuite -> software [label=containedin]
officesuite -> writer [label=contains]
operatingsystem -> centos5 [label=contains]
operatingsystem -> centos6 [label=contains]
operatingsystem -> centos7 [label=contains]
operatingsystem -> software [label=containedin]
software -> officesuite [label=contains]
software -> operatingsystem [label=contains]
writer -> libreoffice [label=partof]
writer -> officesuite [label=containedin]
officesuite [shape=box]
operatingsystem [shape=box]
software [shape=box]
```

**Beispielinteraktion**

```
}
> add calc (id=202) contained-in operatingsystem
> remove writer (id=201) part-of LibreOffice (id=200)
> remove writer (id=201) contained-in officesuite
> nodes
calc:202 centos5:105 centos6:106 centos7:107 impress:203 libreoffice:200
    officesuite operatingsystem software
> edges
calc:202-[part-of]->libreoffice:200
calc:202-[contained-in]->officesuite
calc:202-[contained-in]->operatingsystem
centos5:105-[predecessor-of]->centos6:106
centos5:105-[contained-in]->operatingsystem
centos6:106-[successor-of]->centos5:105
centos6:106-[predecessor-of]->centos7:107
centos6:106-[contained-in]->operatingsystem
centos7:107-[successor-of]->centos6:106
centos7:107-[contained-in]->operatingsystem
impress:203-[part-of]->libreoffice:200
impress:203-[contained-in]->officesuite
libreoffice:200-[has-part]->calc:202
libreoffice:200-[has-part]->impress:203
libreoffice:200-[contained-in]->officesuite
officesuite-[contains]->calc:202
officesuite-[contains]->impress:203
officesuite-[contains]->libreoffice:200
officesuite-[contained-in]->software
operatingsystem-[contains]->calc:202
operatingsystem-[contains]->centos5:105
operatingsystem-[contains]->centos6:106
operatingsystem-[contains]->centos7:107
operatingsystem-[contained-in]->software
software-[contains]->officesuite
software-[contains]->operatingsystem
> recommend S1 105
calc:202 centos6:106 centos7:107
> recommend S3 107
centos5:105 centos6:106
> recommend UNION(S1 105,S3 107)
calc:202 centos5:105 centos6:106 centos7:107
> recommend S1 202
centos5:105 centos6:106 centos7:107 impress:203 libreoffice:200
> recommend INTERSECTION(S1 202,UNION(S1 105,S3 107))
centos5:105 centos6:106 centos7:107
> quit
```