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

Programming - Winter semester 2024/25

Final task 2 Version 1.0	20 points	Issue: 26.02.2025, approx. 12:00 noon Submission start: 12.03.2024, 12:00 noon Deadline: 27.03.2024, 06:00 a.m.

Plagiarism

Only independently prepared solutions will be accepted. Submitting external solutions, even partial solutions from third parties, books, the Internet or other sources, is an attempt to cheat and will result in a "failgrade at any time (even retrospectively). Expressly excluded from this are source text snippets from the lecture slides and from the proposed solutions from the exercises in this semester. All aids used must be listed completely and precisely. Everything that has been taken from other people's work unchanged or with modifications must be clearly indicated. Please also observe the faculty's guidelines on the use of Generative AI [1].

Students who disrupt the proper course of an assessment can be excluded from the assessment. The passing on of parts of test cases or solutions, among other things, also constitutes a disruption of the orderly process. These types of disruptions can also expressly lead to the exclusion of the performance review at any time. This expressly means that the score can also be reduced retrospectively.

Communication and up-to-date information

In our *FAQs*[2] you will find an overview of frequently asked questions and the corresponding answers to the "Programming" module. Please read them carefully before asking questions and check them regularly and independently for changes. Please also note the information in the Wiki[3].

We occasionally publish important news in the *ILIAS forums* or on *Artemis*. Any corrections to assignments will also be announced in this way. Active monitoring of the forums is therefore a prerequisite.

[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/

Check the mailbox of your *KIT e-mail address* regularly for new e-mails. Among other things, you will receive a summary of the correction by e-mail to this address. You can then view all comments in the online submission system[4].

Editing instructions

Please note that successful completion of the mandatory tests is required for successful submission of final assignment 2. Your submission will automatically be assessed with zero points if one of the following rules is violated. You must first pass the mandatory tests before the other tests can be evaluated. Allow sufficient time for your first submission attempt.

- Make sure that the program code compiles without errors.
- Only use *Java SE 17*.
- Unless explicitly stated otherwise in a task, do not use any elements of the Java libraries. Exceptions are the class `java.util.Scanner` and all elements from the following packages: `java.lang`, `java.io`, `java.util`, `java.util.regex`, `java.nio.file`, `java.nio.charset`.
- Take care not to create lines, methods and files that are too long. You must adhere to a maximum line width of 140 characters for your solutions.
- Comply with all whitespace rules.
- all rules for variable, method and package naming.
- Select suitable visibilities for your classes, methods and attributes.
- Do not use the `default` package.
- `System.exit()`, `Runtime.exit()` or similar must not used.
- Observe the rules for Javadoc documentation.
- Comply with all other Checkstyle rules.

The following processing instructions are relevant for the assessment of your submission. However, the submission system will *not* automatically your submission zero points if one of the following rules is violated. Please also refer to the assessment criteria in the wiki.

- Do not add any personal data other than your u abbreviation to your submissions.
- Please note that your submissions will be assessed in terms of both object-oriented modeling and functionality. Follow the modeling instructions in the wiki.
- Program code must written in English.
- Comment your code appropriately: as much as necessary, as little as possible.

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

- The comments should be written uniformly in English or German.
- Enter only your u abbreviation in the Javadoc author tag.
- Choose meaningful names for all your identifiers.

Checkstyle

The online submission system automatically checks your source texts for compliance with the checkstyle rules during submission. There are specially marked rules for which the online submission system rates the submission with zero points, as compliance with these rules is mandatory. Other rule violations can lead to points being deducted. You can and should check your source code for compliance with the rules during development. The programming wiki describes how Checkstyle can be used.

Delivery instructions

The submission in the online submission system will be activated on 12.03.2024, 12:00 . Please sure to upload your files to the correct task in the submission system before the submission deadline on 27.03.2024, 06:00 a.m. Start submitting early,
to test your solution and use the forum to clarify any ambiguities. If you are submitting with Git, *you must always* push to the `main branch`.

- Submit your `*. java files` for task A online in individual work with the corresponding folder structure in the corresponding directory.

Reuse of solutions

If you reuse sample solutions from this semester for the final assignments or exercise sheets, you *must* enter "Programming team" in the author tag in the corresponding classes. This is necessary to fulfill the checkstyle criteria.

Exam mode in Artemis

When you have finished a final assignment, you can hand it in early. The "Submit early" button is used for this purpose. Once you have submitted a final assignment early, you can no longer any changes to your submission.

Task A: Recommendation system

A.1 Introduction

In this task, you will program a *product recommendation system* that no large online store should be without. The recommendation system allows its users to name a *reference product* (of interest to them), from which related products are determined. These are displayed to the user as *product recommendations.*

The mapping from a reference product to suitable product recommendations is determined by *recommendation strategies.* The user can from various recommendation strategies or combine them with each other to adapt the recommendation system to their needs.

A special feature of the recommendation system is the management of the database in the form of a *graph*. The database contains *products* and *categories* between which certain *relationships* can be established. For example, a relationship between two products *p1* and *p2* could express that *p2* is the successor of *p1*. If the user is interested in *p1*, it seems worthwhile to display *p2* as a product recommendation.

A.2 Database

The database is organized as a *directed* graph. Products and categories are modeled by **nodes**. Relationships between products and categories are modeled as **directed edges**.

A.2.1 Products and categories

Nodes in the graph represent products and categories. Products and categories have
each have a *name*. Products also have a unique *identification number*. Valid product or category names are described by the regular expression `[a-zA -Z0 -9]+`. Valid identification numbers are positive integer numbers including 0.

Furthermore, product and category names should also be unique within the scope of the task. This means that if two nodes*n* 1 and*n* 2 have the same name, the following must apply: 1*n*=*n* 2.

Node names that differ only by upper/lower case are considered identical. Two nodes with the names *libreoffice* and *libreOffice* are therefore the same node.

A.2.2 Relationships

Edges in the graph relate products and categories to each other. The following *relationship types* exist for this purpose.

Note below that each type of relationship has exactly one *inverse relationship.*

contains: *n 1 containsn* 2 expresses that category*n* 1 contains the product or category*n* 2. Example: software contains operatingsystem. Here,*n* 1 is always a category and*n* 2 is either a product or a category. If both nodes are of the category type, then this relationship expresses a specialization of the supercategory*n* 1 by the subcategory*n* 2. (inverse relationship: *contained-in*)

contained-in: *n 1 contained-inn* 2 indicates that product or category*n* 1 is contained in category*n* 2. Example: operating system contained - in software. Here,*n* 1 is a product or a category and*n* 2 is always a category. If both nodes are of the category type, this relationship expresses a specialization of the supercategory*n* 2 by the Subcategory*n* 1 from. (inverse relationship: *contains*)

part-of: *n 1 part-ofn* 2 expresses that product*n* 1 is a part of product*n* 2, whereby*n* 2 represents a collection of products. Example: writer part - of libreoffice. These are *n*1 and*n* 2 are always products. (inverse relationship: *has-part*)

has-part: *n 1 has-partn* 2 expresses that product*n* 1 represents a collection of products that contains product*n* 2. Example: libreoffice has - part writer. In this case,*n* 1 and*n* 2 are always products. (inverse relationship: *part-of*)

successor-of: *n 1 successor-ofn* 2 expresses that product*n* 1 is the (direct) successor product of*n* 2. Example: centos7 successor - of centos6 . In this case,*n* 1 and*n* 2 are always products. (Reverse relationship: *predecessor-of*)

predecessor-of: *n 1 predecessor-ofn* 2 expresses that product*n* 1 is the (direct) predecessor product of*n* 2. Example: centos6 predecessor - of centos7 . In this case,*n* 1 and*n* 2 are always products. (Reverse relationship: *successor-of*)

If an edge *e*=(*n* 1,*n* 2,*b*) is added to the graph using the relationship*b* from node*n* 1 to node *n*2 is added, the inverse relationship$\bar{b}$ should also be created using the edge (*e* = *n* 2,*n* 1,$\bar{b}$) if these are not already present.

This is because redundant edges should be avoided: For two edges 1*e*= (*n* 11,*n* 12,*b* 1) and*e*= (*n* 21,*n* 22,*b* 2) 1*e*=*e* 2 must apply if*n* 11 =*n* 21 and*n* 12 =*n* 22 and 1*b*=*b* 2.

Furthermore, a node cannot related to itself: For each edge *e*= (*n* 1,*n* 2,) 1*bn*≠ *n* 2 applies.

Note that there can be any number of relationships between nodes, for example, a product can be the successor of several products, or a product can consist of several products.

Figure A.1 shows the graph of an example dataset.

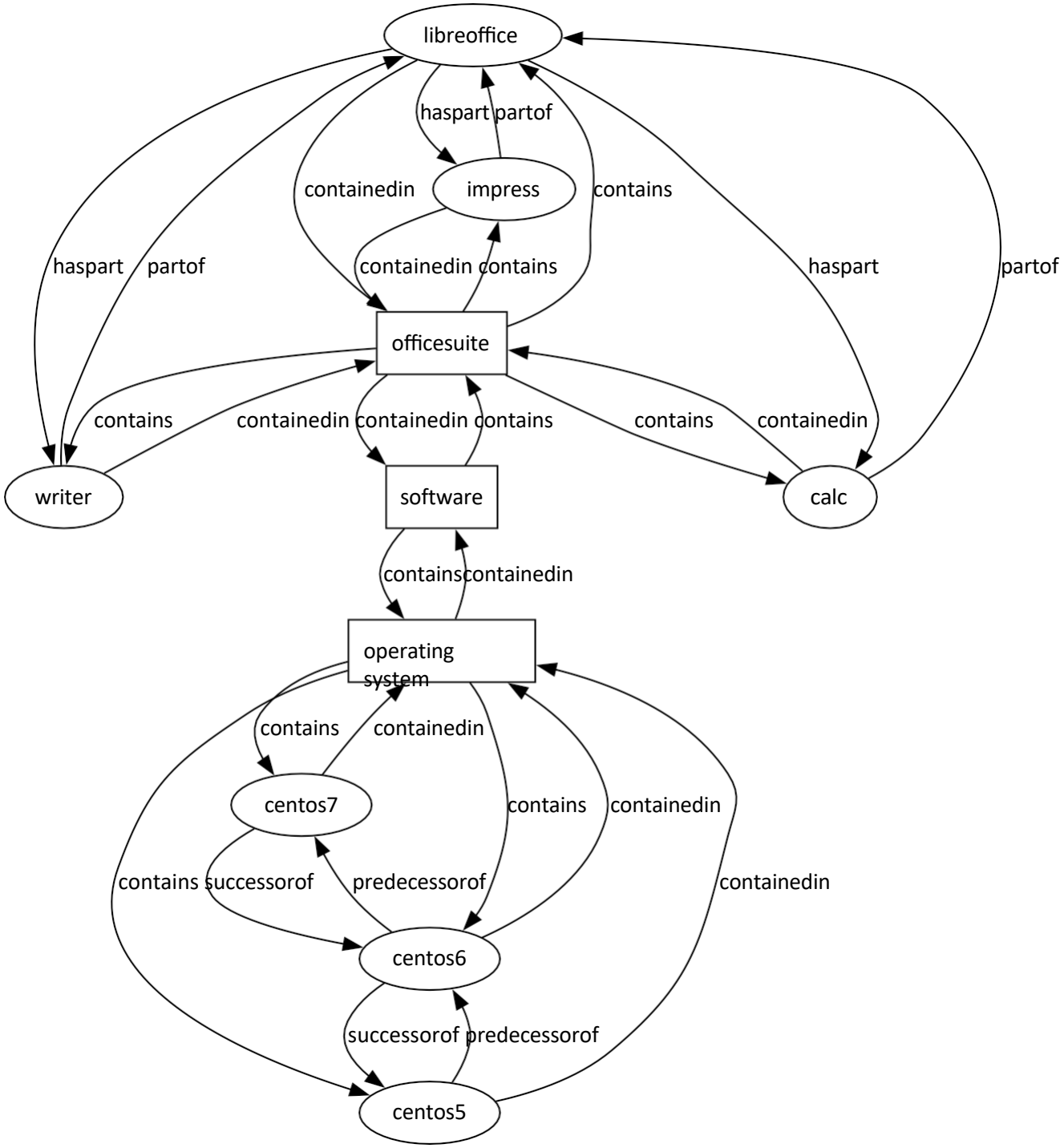


Figure A.1: Graph of a sample dataset

A.3 Recommendation strategies

A recommendation strategy calculates a set of related products based on a reference product. To do this, each recommendation strategy operates on the database and makes particular use of the semantic information provided by relationships.

In the following, three *simple recommendation strategies* are first specified before describing how these can be combined to form *composite recommendation strategies*.

A.3.1 Simple recommendation strategies

For all recommendation strategies introduced in the following, the reference product is never part of the recommended product quantity and may have to be removed before the recommendations returned. Furthermore, only products should recommended, never categories.

Sibling products (S1) provides all products for a specific reference product p_r that *have* a **direct** *contained-in relationship with* p_r to a common supercategory. Using the example of Figure A.1, let us assume that `centos6` is the reference product

is. This strategy then recommends the products `centos5` and `centos7` .

Successor products (S2) returns all **direct and indirect** successor products of p_r for a specific reference product p_r by tracing the *predecessor-of* relationship starting with p_r . Using Figure A.1 as an example, let us assume that `centos5` is the reference product

is. This strategy then recommends the products `centos6` and `centos7` . Please note that a product can have several direct successors. One example of this is the Office suite Open-Office, from which the products Apache OpenOffice and LibreOffice emerged as two independent successors (not shown in Figure A.1).

Predecessor products (S3) returns all **direct and indirect** predecessor products of p_r for a specific reference product p_r by tracing the *successor-of* relationship starting with p_r . Using Figure A.1 as an example, let us assume that `centos7` is the reference product

is. This strategy then recommends the products `centos6` and `centos5` . Please note, that a product can have several direct predecessors. To continue with the Open Office example The Apache OpenOffice and LibreOffice projects could continue to work together, for example in the (unlikely) event that the Apache OpenOffice and LibreOffice projects join forces again in a joint product (not shown in Figure A.1).

A.3.2 Composite recommendation strategies

A composite recommendation strategy combines the recommendations of two other recommendation strategies. Let R 1 and R 2 be product sets that were obtained using two simple recommendation strategies. The combination of R 1 and R 2 can then take place in two different ways:

Intersection: $1 R \cap R\ 2 = \{ x | x \in R\ 1 \wedge x \in R\ 2 \}$

Formation of union: $1 R \cup R\ 2 = \{ x | x \in R\ 1 \vee x \in R\ 2 \}$

The product quantities R 1 and R 2 may themselves have originated from a composite recommendation strategy. Further information on this is provided in A.7.1.6

A.4 Database file

Your program reads in a text file containing the products, categories and their relationships to be stored in the database.

A.4.1 Grammar

The dataset file consists of one or more lines. Each line uses a predicate to establish a relationship between a subject and an object. The format of such a line is
is given by the following BNF grammar [5]. Terminal symbols are written `in typewriter style`.
In contrast to the BNF, the non-terminal symbols *productid*, *productname* and *categoryname* are each specified by a regular expression that expresses the set of valid terminal symbols.

BNF grammar of a line of the database file

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

In addition to the above grammar, unnecessary spaces should be ignored instead of triggering an error message according to the following rules:

Both spaces are required within the derivation rule *line*; additional spaces may appear before/after *subject*, *predicate* and *object*. Within the derivation rule *product*

no spaces required; additional spaces may before/after *productname*, (, `id`, =, *productid*,). All other derivation rules strictly follow the corresponding BNF specification.

For example, a line of the following form should accepted:

`centos7 ( id= 107 ) contained - in operating system`

However, the following components must not contain any spaces: *predicate*, *productid*, *productname*, *categoryname*.

When constructing the graph, remember to create inverse relationships (see subsection A.2.2) if these are not already described in the dataset file.

If an error occurs when reading the file, be it an input/output error when reading the file, a syntax error or a semantic error, an error message is issued. A syntax error occurs if the file is not formed according to the above specification. A semantic

[5] Backus-Naur form, see e.g. http://de.`wikipedia.org/wiki/Backus-Naur-Form`

An error occurs, for example, if an invalid relationship is created, such as two categories that are to be linked using the *successor-of* relationship.

A.4.2 Example

The following example presents a well-formed dataset file to build the graph shown in A.1:

Example

```
1  CentOS5 ( id= 105) contained-in operatingSystem
2  centOS6 ( id= 106) contained-in OperatingSystem
3  operatingSystem contains centos7 ( id= 107 )
4  operating system 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)
```

The graph shown in Figure A.1 can generated with different inputs. The above input is therefore only an example input for the structure of this graph.

A.5 DOT notation

The DOT notation is used to describe graphs. A special feature of this notation is that it can be read by both humans and machines[6].

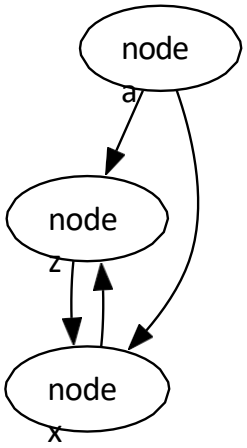
A.5.1 Directed graph

The notation of a directed graph begins with the keyword `digraph`. Nodes and edges are described within two curly brackets. The names of the nodes are arbitrary. A directed edge is defined by `->`. A graph in DOT notation and its corresponding visual representation are shown below.

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

Example

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



A.5.2 Further features of the DOT notation

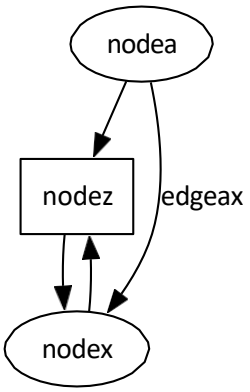
To display a node as a rectangle, use the name of the node and the keyword [shape= box].

To provide edges with a label, use [**label**= edgename]. The following example

illustrates these cases:

Example

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



A.6 Notes on implementation

After starting, your program accepts input via the command line using the standard input, which is specified in more detail below. All commands are executed in the current state of the program. After processing an input, your program waits
for further input until the program is terminated at some point by entering the character string quit
is terminated.

In the following, input lines are introduced with a closing angle bracket (>) followed by a space. These two characters are not part of the entered command, but
are used to differentiate between input and output lines.

Make sure that the semantic and syntactic specifications are not violated when executing the following commands and always a meaningful error message if the specifications are violated. If the user input does not correspond to the specified

format, an error message must also be output. After an error message has been output, the program should continue as expected and wait for the next input. The error message should be formed in such a way that the user can recognize why an input was rejected.

As we carry out automatic tests of your interactive user interface, the output must correspond exactly to the specifications. In particular, lower and upper case letters as well as spaces and line breaks should match exactly. Only use the information specified in the task. Do not enter any additional information. For error messages, you are free to the English-language text, but it should be

can be useful. However, every error message must begin with `Error`, and must not contain any special characters,
such as line breaks or umlauts.

Unless otherwise specified, the standard input `System.in` should always be used for input. Unless otherwise specified, the standard output `System.out` should always be used for output.

use. For error messages, the standard error output `System.err` can optionally be used instead of the standard output. Never reassign this standard input and output.

Please note that in the description of the input and output formats, the words between angle brackets placeholders that are replaced by values in the actual input and output. These actual values do not contain angle brackets during input and output. See also the example sequence.

A.7 Functionality

The required functionality of the system is described below:

A.7.1 Commands

A.7.1.1 The `load database` command

Loads a specified database file and parses its content. For test purposes, the command should output the content of the Verbatim database file. If the command is executed multiple times, the old dataset is overwritten if the new dataset is valid.

Input `load␣database␣path`

Output Verbatim Contents of the file

Example interaction

```
1 > load database database.txt
2 CentOS5 ( id= 105) contained-in operatingSystem
3 centOS6 ( id= 106) contained-in OperatingSystem
4 operatingSystem contains centos7 ( id= 107 )
5 operating system contained-in software
6 CentOS7 (id=107) successor-of centos6(id=106)
7 CentOS5 (id=105) predecessor-of centos6(id=106)
```

A.7.1.2 The quit command

The parameterless command allows you to exit the program completely. Please note that methods such as `System.exit()` or `Runtime.exit()` must not be used for this.

Enter `quit`

Example interaction

```
1 > quit
```

A.7.1.3 The add command

The `add` command a relationship to the graph. The input format is equivalent to that of the dataset file. Make sure that each relationship has an inverse relationship and
a relationship may not be added twice.

Input `add␣subject␣predicate␣object`

Example interaction

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

A.7.1.4 The remove command

The `remove` command removes a relationship between two nodes if they exist before it. The input format is equivalent to that of the data inventory file. Make sure that the
inverse relationship is also removed. If a node no longer has a relationship to another node after the relationship has been removed, this node should also be removed.

Input `remove␣subject␣predicate␣object`

Example interaction

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

A.7.1.5 The nodes command

The `nodes` command outputs a list of all nodes.

The nodes are output separated by spaces in a single line. A *product node* is represented by its name in lower case, followed by a colon, followed by the product identification number. A *category node* is represented by its name in lower case.

The nodes are displayed in ascending order by product or category name.

Input nodes

output productname : productid␣ categoryname

Example interaction

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

A.7.1.6 The edges command

The edges command outputs a list of all edges.

The edges are output line by line, with each edge in a separate line. Each of these directed edges has three components in the context of this task: Source node, relationship type, target node. Source and target nodes are each represented by their names in lower case. If it is a product, a colon follows, followed by the
Product identification number. The relationship type is introduced by the characters -[, followed by the name of the relationship type in lower case, followed by the characters]- .>

The edges are output in ascending order, sorted by source name, then target name and finally predicate. For predicates, the order from subsection A.2.2 applies.

Input edges

Output subject -[predicate]-> object

Example interaction

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

A.7.1.7 The recommend command

With the help of the recommend command, the user can have products recommended. The respective reference product is represented by its product identification number.

A valid recommend command is structured as follows. As before, regular expressions are used as BNF extensions and terminal symbols are set in typewriter style.

BNF grammar of the recommend command

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

In addition to the above grammar, unnecessary spaces (only) within the non-terminals *term* and *final* should be ignored instead of triggering an error message. For example, an input of the following form should be accepted:

recommend UNION (S1 105 , S3 107)

Implement a **recursive** *descent* **parser** for processing the recommend command as introduced in the lecture. Think about
suitable data structures for the parsed terms (final term, intersection term, union term).

The recommended products are displayed in a single line, separated by spaces. Each product is represented by its name in lower case, followed by a colon, followed by the product identification number. The products are displayed in ascending order by product name. If the quantity of recommended products is empty, an empty line is displayed.

Input See BNF grammar of the recommend command

Output productname : productid␣ productname : productid

Example interaction

```
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 The `export` command

The export command outputs the graph of the imported dataset to the console in DOT notation.

Details on the DOT notation can be found in subsection A.5. Please use only the language constructs from subsection A.5 .for the export command

Tests fail.

The output begins with the line digraph { and ends with the line }.

In between, each line contains either an edge in DOT notation or the definition of a category node. The label of an edge corresponds to its relationship type, whereby hyphens

can be removed. Valid edge labels are, for example, containedin or successorof.

Product nodes with incoming and/or outgoing edges do not need to be described in a separate line, as they are represented indirectly by the edges. Category nodes, on the other hand, should be displayed as rectangles. For this purpose, each category node must be output in a separate line.

The output takes place in two steps:

1. First, the lines with edges are sorted in ascending order, first by source name, then by target name and finally by predicate. For predicates, the order from subsection A.2.2 applies.
2. This is followed by the rows of the category nodes, which are also sorted in ascending order.

Both product nodes and category nodes are represented by their names.

input `export`

output Graph in DOT notation

Example interaction

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

A.8 Example interaction

The line numbers and the separator line are not part of the user interface, they only serve as orientation for the given example interaction. The input lines are marked with
is introduced by a closing angle bracket (>) followed by a space, these two characters are also not part of the command entered, but are used to differentiate between input and output lines.

Example interaction

```
%> 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 )
operating system 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 -> operating system [label=containedin]
centos6 -> centos5 [label=successorof]
centos6 -> centos7 [label=predecessorof]
centos6 -> operating system [label=containedin]
centos7 -> centos6 [label=successorof]
centos7 -> operating system [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]
operating system -> centos5 [label=contains]
operating system -> centos6 [label=contains]
operating system -> centos7 [label=contains]
operating system -> software [label=containedin]
software -> officesuite [label=contains]
software -> operating system [label=contains]
writer -> libreoffice [label=partof]
writer -> officesuite [label=containedin]
officesuite [shape=box]
operating system [shape=box]
software [shape=box]
```

Example interaction

```
}
> add calc (id=202) contained-in operating system
> 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 operating system software
> edges
calc:202-[part-of]->libreoffice:200
calc:202-[contained-in]->officesuite
calc:202-[contained-in]->operating system
centos5:105-[predecessor-of]->centos6:106
centos5:105-[contained-in]->operating system
centos6:106-[successor-of]->centos5:105
centos6:106-[predecessor-of]->centos7:107
centos6:106-[contained-in]->operating system
centos7:107-[successor-of]->centos6:106
centos7:107-[contained-in]->operating system
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
operating-system-[contains]->calc:202
operating-system-[contains]->centos5:105
operating-system-[contains]->centos6:106
operating-system-[contains]->centos7:107
operating system-[contained-in]->software
software-[contains]->officesuite
software-[contains]->operating system
> 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
```