**Instructions regarding team formation and programming tasks:**

- Students will be provided with 5 programming tasks (P01 to P05) having a total of 100 (hundred) points. At least **66% of the total programming points must be achieved to successfully qualify for the exam**. P01 to P04 has 15 points each and P05 has 40 points. Hence completing P05 (with a demo) is mandatory to successfully qualify for the exam. Programming tasks are required to be submitted via Moodle. Each programming task is required to be done in groups (details are below).
- Tasks P01 to P04 must be performed using **Apache Lucene 7.4** and **Java 8**. The submitted code will be checked in **Windows 10 or Ubuntu Linux**. **For P01 to P04, you need to create a one-page short document stating how you solved the sub-parts**.
- **Only** for task P05, based on the use case selected by your group, any frontend technology (Java Swing, Spring, JSP-Servlets, Angular, React, Javascript, HTML etc) can be used. However, the backend should be using one of these: **Lucene 7.4, Java** or **PyLucene** or**ElasticSearch** or **Apache Solr** or **LIRE**. The application can be a desktop GUI based (e.g. Java Swing) or running on a web browser and server (like Apache Tomcat etc). Hence, for P05 we need more detailed documentation from you, how to run the code that your group will submit.
- The group should consist of 3 (minimum) to 4 (maximum) students. Email the details of your group to **sayantan.polley@ovgu.de** by 20th of October 2023, 11 AM CET, so that we can create the groups in Moodle.
- At least two people from each team should be available for presentation of P05 in the exercise class, which is supposed to take place between 22-29th of January '24. Note that students attending in-person exercises on different days (e.g. Mondays and Tuesdays) can also form groups for programming tasks. Only for the P05 demo you may have to attend one class together (we will co-ordinate this later in exercises).
- Please note that, besides the programming tasks, there will be 12 homework exercise sheets (about 3 tasks per sheet) with about a total of 36-40 tasks. Students need to prepare them as homework and vote them during the exercises in person. Solutions will be discussed in the exercises. 66% of the total homework exercise tasks need to be voted in the classroom for qualifying for the written exam along with 66% in programming.

**Tasks and Deadlines:**

| Tasks | Deadline |
|---|---|
| If you are unable to find a team for the programming task, email to **sayantan.polley@ovgu.de**. We will create groups out of the applications received by random allocation. Mention the subject of the email as [**IRP-Cannot find group**]. | 20th October 2023, 11:00 CET |
| P01 Submission (15 points) | 27th October 2023, 11:00 CET |
| P02 Submission (15 points) | 10th November 2023, 11:00 CET |
| P03 Submission (15 points) | 17th November 2023, 11:00 CET |
| P04 Submission (15 points) | 1st December 2023, 11:00 CET |
| For Programming task P05, we would like to see the approach which you are going take to solve the task by a simple one page document. In this document (submission via Moodle), the group needs to specify which usecase they will select, API and methods from the API. | 8th December 2023, 11:00 CET |
| P05 Submission (40 points) | 17th January, 2024, 11:00 CET |
| Demo of P05 in Exercise Class | 22-29 January, 2024 |

**Note:** The submissions for the all programming tasks should be done via Moodle and not via email. Any submission done via email will not be considered. Email **sayantan.polley@ovgu.de** for any further questions on the programming tasks.

## Programming Assignment:01 [Points: 15] [Deadline: 27th October, 2023, 11:00 Hours CET]

This assignment will help understand basic preprocessing steps like tokenization, filtering and stemming.

Consider the following text simplicity:

> Today is sunny. She is a sunny girl. To be or not to be. She is in Berlin today. Sunny Berlin! Berlin is always exciting!

Note that For P01 to P04, you need to create a one-page short document (PDF to be uploaded in Moodle per group) stating how you solved the sub-parts. Implement the following sub-parts:

a) Use StandardTokeniser and WhitespaceTokeniser on the above text to print out the tokens. What is the difference?

b) Use StandardTokeniser and StopwordFilter on the above text and print out the tokens. Stop words to use are the following "was", "is", "in", "to", "be"

c) Create an Analyzer with the following configuration

   - Tokenizer: StandardTokenizer
   - Filter: Lowercase Filter
   - Filter: StopwordFilter with stop words to use are the following "was", "is", "in", "to", "be"
   - Filter: PorterStemmerFilter

   With the Analyzer created, analyze the text above and print the tokens.

Hints:

- `https://lucene.apache.org/solr/guide/6_6/understanding-analyzers-tokenizers-and-`
  `html`
- `https://examples.javacodegeeks.com/core-java/apache/lucene/`
  `lucene-indexing-example/`
- `https://lucene.apache.org/core/7_4_0/analyzers-common/org/apache/`
  `lucene/analysis/custom/CustomAnalyzer.html`

## Programming Assignment:02 [Points: 15] [Deadline: 10th November, 2023, 11:00 Hours CET]

This assignment will introduce the idea behind indexing and posting lists.

Implement the following:

a) Implement assignment 4.1

b) Print posting list (without skip pointers) for all terms indexed in above step for terms "sunny" and "to". Print format [tokenname:total_frequency:doc_frequency]->[docid:frequency:[positions]]->[docid:frequency:[positions]]

Hints:

- `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/codecs/lucene70/package-summary.html`
- `https://lucene.apache.org/core/7_4_0/core/index.html` (field section)
- `https://examples.javacodegeeks.com/core-java/apache/lucene/lucene-indexing-example/`
- `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/index/package-summary.html#postings-desc`

## Programming Assignment:03 [Points: 15] [Deadline: 17th November, 2023, 11:00 Hours CET]

This assignment will help understand and create a custom tokenizer and custom filter.

Implement the following:

a) Implement Biword Tokenization and tokenize the following:

> Today is sunny. She is a sunny girl. To be or not to be. She is in Berlin today. Sunny Berlin! Berlin is always exciting!

b) Implement assignment 4.3(a)

Hints:

- Extend TokenFilter class and override the function public boolean incrementToken() throws IOException
- The required attributes needs to be properly filled (CharTermAttribute is enough for this task)
- Extend TokenFilterFactory and override the following functions:
  public TokenStream create(TokenStream input)
  public TokenStream normalize(TokenStream input)

## Programming Assignment:04 [Points: 15] [Deadline: 1st December, 2023, 11:00 Hours CET]

Understand and implement Vector Space Model and probabilistic model to retrieve relevant documents.

Implement the following:

a) Implement assignment 6.1(a)

b) Consider the following text where each sentence should be seen as a document:

> Today is sunny. She is a sunny girl. To be or not to be. She is in Berlin today. Sunny Berlin! Berlin is always exciting!

Use Vector space model and BM25 Model to find relevant documents while querying the following document

> She is a sunny girl.

Print the relevant documents in the format
`<document relevance score> : <document content>`

Hints:

- `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/search/package-summary.html#scoring`
- `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html`
- `https://lucene.apache.org/core/7_4_0/core/org/apache/lucene/search/similarities/package-summary.html`

## Programming Assignment:05 [Points: 40] [Deadline: 17th January, 2024, 11:00 Hours CET]

Design and program your own web search engine.

- Overall, the software should be able to preprocess data (text or images, depending on the use case selected), create a backend index with frontend interface (GUI) that allow users to search. Each group needs to fulfill any one of the six usecases.
- The selected use case for each group should be submitted with a one page high level software architecture plan; **deadline** for this plan is 8th Dec 2023. The plan should provide details of which backend API (Solr, Lucene, LIRE, Pylucene), details of specific methods from the API that will be used. Along with an approximate wire-frame design of the user interface and selection of appropriate frontend technology (e.g. Java Swing, C#, DotNet, Angular/React, Javascript, HTML-CSS etc) that the group is planning to use.
- For Usecase 1, 2 and 3 - consider the corpus as Wikipedia documents in English. Download and unzip the file **Additional resources for P05** from moodle and then follow the steps. Create a custom (sub-set) of Wikipedia (English) dataset with atleast 1000 topics by following steps mentioned in document **Steps to create the dataset**. Parse and index each crawled Wikipedia topic in Lucene/Solr/PyLucene. Wikipedia data set does not apply for usecase 4 and 5, which has relevant data sets for those tasks. Hence for usecase 4 and 5, crawling described above, does not apply. But note that there are specialised pre-processing required for usecase 4 & 5.
- To get the pointers for creating a graphical user interface (GUI) please follow the document which has links to create a graphical user interface (GUI) **Links to create a Graphical User Interface**. These are just a few recommendations, you are free to select any other framework. However the backend search engine for all usecases should be implemented using one of these APIs: Apache Lucene/Solr/Elastic Search/LIRE or PyLucene. The search application can be a desktop GUI (e.g. Java Swing) or running on a web browser and server (like Apache Tomcat etc).

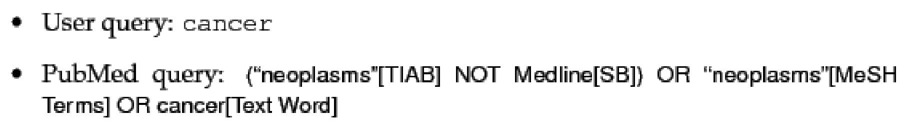
**Usecase 1: Search Result Clustering**
Clustering[1] is an unsupervised machine learning algorithm to partition data points based on similarity. Implement a cluster based visualization of 10 (or more) relevant retrieved results for a given search query. That is, no standard list based result visualization is used here but a visualization that supports exploration by illustrating the relation (e.g. similarity) between the results.You can use any simple clustering algorithms like Kmeans

[1]https://en.wikipedia.org/wiki/K-means_clustering

or HAC. You can also use Solr feature[2] for result clustering. Use any Java/Python based Machine Learning library like WEKA 3.8 or Python Sk-learn.

### Usecase 2: Query expansion with WordNet

Query expansion is the idea where, "users give additional input on query words or phrases, possibly suggesting additional query terms"[3]. Furthermore, introduce the concept of weighting, where the original words are given higher weights than derived words. Finally, highlight the matching keywords. A good example can be the figure below.



- User query: `cancer`
- PubMed query: ("neoplasms"[TIAB] NOT Medline[SB]) OR "neoplasms"[MeSH Terms] OR cancer[Text Word]

Abbildung 1: Query Expansion Example, (Image ref: Introduction to Information Retrieval).

Note the figure, how the term cancer is expanded into neoplasms based on a certain domain. For this task, implement synonym query expansion using WordNet.

Reference:

- Working example of wordnet with synonyms can be found here[4].
- A theoretical reference to the IR text book[5]

[2]https://solr.apache.org/guide/6_6/result-clustering.html
[3]https://nlp.stanford.edu/IR-book/html/htmledition/query-expansion-1.html
[4]https://www.nltk.org/howto/wordnet.html
[5]https://nlp.stanford.edu/IR-book/html/htmledition/relevance-feedback-and-query-expansion-1.html

**Usecase 3: Implement a retrieval based chatbot**
Implement a chatbot based text search engine with GUI. The backend will index the documents in Apache Lucene/Solr. You can use any open source chatbot API implementation. Make simple assumptions that in the front-end user can ask questions like "why" and "how". For other queries (i.e. non-who and non-how queries), we can resort to simple free text search and query Lucene/Solr. You are free to make simple and reasonable assumptions, like restricting the number of results to 3 or 5 or also get images along with text or ask feedback from the user in the GUI! As a final result, we expect a GUI where user can search in a "chat like experience". The chatbot API and tool kits may be in Python etc. but we expect that Apache Lucene and Solr be used for search purposes. Some resources[6] and github code[7] pointers are provided. Hint: This usecase may be difficult for beginners.

**Usecase 4: Personalized Ranking**
In the lecture you have learned about bag-of-words or simple term incidences, as your feature representation. For this task, one is expected to extract handcrafted features, which have more high level connotation, like sentiment or writing style in a fiction book (a book is a long document). Consider a sub-set of books belonging to the 19th Century English Fiction. The data set is created from Project Gutenberg. The data set consists of about 1000 books and roughly 10 genres. Extract features that are relevant to fiction books, which may include ideas like sentiment, setting and so on, using appropriate libraries if possible. Finally weight the features and thus use weighted cosine similarity to present your results. The weights for each feature (or some of the features) can be given by the user from the frontend, and thus by default each feature will have equal weights until otherwise specified.
Hint: This usecase may be difficult for beginners. You need to cover ground with some details of feature engineering used in machine learning (text mining, NLP).

Dataset: A subset of Gutenberg Corpus[8] with about 1000 English fiction books.

Reference:

- Read this paper[9] for more information about handcrafted text features for Gutenberg corpus fiction books.

**Usecase 5: Image Retrieval System**
In the lecture we will see that documents can be represented by vectors and then indexed by extracting features vectors (say tf-idf or incidence vectors). The same is also true for Images, i.e., features vectors or descriptors can also be extracted from images and thus images can also be indexed. You can use just one feature like Colour Layout Descriptor[10]

[6]https://drops.dagstuhl.de/opus/volltexte/2019/10869/pdf/OASIcs-SLATE-2019-2.pdf
[7]https://github.com/JavaChat/OakBot, https://github.com/gunthercox/ChatterBot
[8]https://dke.ovgu.de/findke/en/Research/Data+Sets-p-1140.html
[9]https://github.com/sayantanpolley/fiction/blob/master/SIMFIC_LNCS.pdf
[10]https://en.wikipedia.org/wiki/Color_layout_descriptor

(CLD), one of the simplest feature descriptors based on the MPEG-7 standard or more rich features from neural nets or perhaps combine multiple types of features. Create a system that indexes images and then given a query image, retrieve top n relevant images. This is called the "query by example" approach since we query an actual image. The result page should display the query image, and the relevant result images.

Dataset: Use Pascal 2006 dataset[11] for this task. At least 1000 images, with equal distribution to each class, must be indexed.

Reference:

- LIRE[12]: An image retrieval system based on Lucene.
- Some widely used image feature vectors or descriptors can be found here[13]
- A nice blog post on feature Extraction from Neural Nets (ignore the aspect of large dataset) can be found here[14]

[11]http://host.robots.ox.ac.uk:8080/pascal/VOC/voc2006/index.html
[12]https://github.com/dermotte/LIRE
[13]https://github.com/scferrada/imgpedia
[14]https://www.pyimagesearch.com/2019/05/27/keras-feature-extraction-on-large-datasets-with-deep-learning/

## Evaluation for the Programming tasks

This list gives a short overview about the scoring of programming assignments.

| English | Deutsch |
|---|---|
| correctness/completeness (50% ) | Korrektheit/Vollständigkeit (50%) |
| Overall Approach documentation (1-2 pages) (10%) | Doku. Lösungsweg (1 - 2 Seiten) (10%) |
| Code documentation (10%) | Doku. Quellcode (10%) |
| Code Format** (10%) | Form (10%) |
| Comprehensible (10%) | Nachvollziehbarkeit (10%) |
| punctual submission (10%) | Pünktliche Abgabe (10%) |

**Completeness:** The code should run without any error on Windows or Linux OS.

* how does the program work and how can I use it? (like a mini-user guide). The overall approach documentation is applicable only for P05.
** is the source code structured well into logical sections?

**Note: Plagiarism of any kind is prohibited**