# Information Retrieval project

**First member:** Ilias Varthalitis, AM: 4637 *Department of Computer Science & Engineering*
**Second member:** Aristeidis Panagiotou, AM: 4754 *Department of Computer Science & Engineering*

**GitHub:** Information Retrieval project

## Introduction

The aim of this project is to design and implement an information retrieval system for scientific articles. Using the Lucene library, an open-source toolkit for text search engines, the system will enable users to efficiently search through a collection of academic papers.

## Corpus

To create the corpus, we developed a Python script that extracted all columns from papers.csv except for the source_id. We used the source_id to identify the authors of each paper. Once we had the authors identified, we compiled the data into a final corpus.csv. This final file included a column named ”authors” along with the other fields from papers.csv: ”title”, ”year”, ”abstract”, and ”full_text”.

## Text analysis and index creation

For indexing and text analysis, we created a class named Indexer.java that is responsible for two main tasks. The first method, indexCSV, reads data from corpus.csv and stores it into Document objects. We created our own Document class with fields: authors, year, title, abstract, and full_text. The second method, indexDocument, takes the data from the Document objects and converts them into Lucene documents in a specific format. Once the data is in this format, we use writer.addDocument(luceneDoc) to store the data in the index. In the constructor of the Indexer class, we initialize the necessary components: the IndexWriter, the StandardAnalyzer, and the Directory dir where the index will be stored on disk.

## Search

For searching, we created a class named Searcher.java. Within this class, there is a primary method called search which takes a single argument, String queryString. The queryString is passed to the parser with the following command: Query query = parser.parse(queryString);, and then the search is executed with the command: TopDocs results = searcher.search(query, 200);, where the results are stored in the results variable. In the constructor of the Searcher class, we initialize everything needed, such as the StandardAnalyzer, the fields variable that holds the field names, the IndexSearcher, and the parser.Finally, we have a method named saveSearchToHistory, which saves the history of previous searches.

## Result representation

We also have a final class, Manager.java, which is responsible for creating instances of Indexer and Searcher objects and calling the appropriate methods to properly execute the query and perform the search on the stored index.

Within the Frontend folder, we created gui.java using JavaFX to develop the graphical user interface. In this file, we implemented all the functionalities requested, such as displaying search results in a user-friendly manner with the option to sort them by year, highlighting keywords, and displaying results in batches of ten. Additionally, users can choose between searching by keyword or by field, and for fields like abstract and full_text, where the text is very long, we implemented links that open a new window with the full result when clicked.Finally we have implemented two buttons for the search history. With the first button, the user has the ability to view the history and delete it. With the second button, the user can search based on a previous query found in the history. This is our additional search method as required by the instructions.