

Information Retrieval

WS 2013 / 2014

Lecture 1, Tuesday October 22nd, 2013

(Demo, Organizational, Inverted Index)

Prof. Dr. Hannah Bast

Chair of Algorithms and Data Structures

Department of Computer Science

University of Freiburg

Overview of this lecture

- Demo
 - One of our prototypes + a glimpse of what lies behind it
- Organizational
 - Style of this course
 - Course Systems: Wiki, Forum, Daphne, SVN, Jenkins, ...
 - Exercise Sheets + Requirements + Exam
- And then let's dive right in
 - Inverted Index (INV) ... we will implement one together
 - How to do efficient full-text search with INV
 - **Exercise Sheet #1:** implement that "how to" !

Demo 1/2

- Let's start with a demo

- Of **Broccoli**, our prototype for semantic full-text search

full-text search = given a set of keywords, return documents containing those keywords

semantic = do not just match the query words literally but (try to) understand what they mean

- At our chair we do everything from **basic research** to actually **building systems** using the whole stuff

That is what makes computer science fascinating for me

And I hope (some of) you will like it, too

Demo 2/2

- Research topics behind the demo you just saw
 - Indexing needed for fast query times
 - List Intersection the basic operation in index lists
 - Ranking most relevant hits should come first
 - Compression lots of data, store it efficiently
 - Error-tolerant search errors in query or document
 - Web app stuff Javascript, AJAX, JSONP, DOM
 - Machine learning solve classification tasks automatically
 - Ontologies how to organize factual knowledge
 - Evaluation argue that one system is better than another

You will learn about all that (and more) in this course

Style of this course

■ What I will do

- Explain the basics, often by example
- Underlying theory, wherever needed
- Give implementation advice + provide code skeletons

■ What you will do

- Understand the basic idea + then implement it

Implementation is great, because it makes you understand all the important details + a working implementation is proof that you understood it

- Interesting experiments with your code
- Some theoretical tasks ... but not too many

Course systems

- All linked from the Wiki page for the course:
 - There is an **SVN** repository for your submissions, in particular for your code
 - There is a **Forum** for asking questions
 - All the **course materials** will be put online: the **slides**, the **lecture recordings**, the **exercise sheets**, as well as any **code we write in the lectures**
 - We also provide **Jenkins**, a **continuous build system** that automatically checks the code you commit to our **SVN**
 - Links to all these resources are provided in **Daphne**, our course management system

Exercises + Exam 1/3

- There will be one exercise sheet per week
 - 80% Implementation, 20% theory on average
 - You can work on the sheets alone or in groups of 2 people
 - Submit the code to our [SVN ... see URL on your Daphne page](#)
 - Follow our [Coding Standards ... see next slide](#)
 - You can get 20 points per exercise sheet
 - The exercises are the most important part of the course

[They are the key to a real understanding](#)

[If you do all the sheets \(yourself + properly\), you are very well prepared for the exam](#)

■ Tutorials

- The tutorials are completely "online", that means:

You can ask questions on our Forum anytime, and you will typically get a quick answer

you find a link to the Forum on the Wiki

You will get the corrections / feedback from your tutor online, too, via the SVN

Exercises + Exam 3/3

- There is a written exam in the end
 - The date is not yet fixed, let's discuss on one in one of the last lectures of this semester
 - **You need 50% of the points from the exercise sheets to be admitted to the exam**
 - In the exam, there will (probably) be six tasks, out of which you can choose five
 - For example tasks, see the exam from a year ago

You find the link on the Wiki page of this course

■ Mandatory

- Write your code in **C++** or in **Java**
- **Document** each class and each non-trivial method
- Your code must conform to our **style checkers**
- Write a **unit test** for every non-trivial function
- Use a standardized **Makefile** / **build.xml** file
- You find a comprehensive example on <https://daphne.informatik.uni-freiburg.de/CodingStandards>
- Check your submissions on our build system **Jenkins**

I will walk you through an example in this lecture

■ Recommended

- Write your code as **simple** as possible
... without compromising efficiency to much
- Write your code as **minimal** (length) as possible
... without compromising simplicity and efficiency too much
- Use long **standard names** for most variables
e.g. `invertedIndex` for an instance of class `InvertedIndex`
- Use short names for indices and variables used frequently in local code blocks

```
for (int i = 0; i < n; i++) { A[i] = B[n - i]; }
```

How much work is it / ECTS points

- ECTS points = working time
 - You get 6 ECTS points for this lecture
 - That is $6 \times 30 = 180$ hours of work for the whole course
 - There will be 13 exercise sheets = 13 weeks with actual work
 - Doing all the exercise sheets and understanding everything behind them is the perfect preparation for the exam
 - Time management options ES = Exercise Sheet, EXAM = exam preparation
 - A.** 9 hours per ES, 60 hours for EXAM **RECOMMENDED**
 - B.** 6 hours per ES, 100 hours for EXAM **MINIMUM**
 - C.** 0 hours per ES, 180 hours for EXAM **NOT POSSIBLE**

The hours per ES are meant all-inclusive, that is: including the time for listening to the lecture, and any follow-up work you might need

Keyword Search

■ Problem definition

- Given a collection of text documents ... e.g. **the web**
- Given a keyword query ... e.g. **uni freiburg**
- Return all documents that contain all keywords

- **Refinements** ... not today, but in some later lectures
 - Rank** the docs so that the most relevant ones come first
 - Also return docs that contain only **some** of the keywords
 - Also return docs that contain **variations** of the keywords
 - Make **suggestions** for related / better queries

Inverted Index 1/2

- Why do we need an index for search?
 - **Naive solution:** given a query, iterate over all the documents, and identify those that match
 - That is, similar to what the un*x `grep` command does
 - Actually not so bad for small text collections:
 - A modern computer can **scan** through **1 GB** of text in about half a second
 - Query times of ≤ 100 milliseconds feel interactive
 - But already for **1 TB** it would be **20** minutes ...
 - Current web: \approx **50 billion** pages / **2500 TB** of text
- Source: www.worldwidewebsize.com ... assuming **50 KB** / page

Inverted Index 2/2

■ Basic idea of an inverted index

- For each word, pre-compute and store the **sorted** list of ids of documents / records containing that word

uni 13, 57, 114, 257, 987, 1345, 2078, ...

freiburg 5, 23, 57, 257, 512, 773, 1345, 3012, ...

- These lists are called **inverted lists**
- Then the list of ids of the matching documents / records is simply the **intersection** of the inverted lists of the keywords from the given query

List Intersection 1/2

■ For two lists

- Provided the lists are sorted, an interleaved left-to-right scan does it in **linear time**

	<i>i</i>					
	↓	↓	↓	↓	↓	
uni	13,	57,	114,	257,	987,	1345, 2078, ...
freiburg	5,	23,	57,	257,	512,	773, 1345, 3012, ...
	↑	↑	↑	↑	↑	
			<i>j</i>			

RESULT 57, 257, ...

For ES1 you can assume that there are no duplicates in the individual lists

List Intersection 2/2

- For more than two lists L_1, L_2, L_3, \dots
 - Can be reduced to sequence of pairwise intersects:
First intersect $L_1 \cap L_2 \rightarrow$ intersection L_{12}
Then intersect $L_{12} \cap L_3 \rightarrow$ intersection L_{123}
And so on ...
 - **Optimization:** order such that $|L_1| \leq |L_2| \leq |L_3| \leq \dots$
Then lengths of intermediate results is minimized
Not necessary for Exercise Sheet 1
 - **Even better:** "multi-way" intersect of all lists at once
More about that in a later lecture ...

Parsing / Tokenization

- We need to break the text into "words"

- Conceptually simple: just define a set of characters that belong to words and a set of characters that don't

For Exercise Sheet 1, you can simply consider a-z and A-Z as word characters, all others as separators

- Words are then maximal sequences of word characters
- In reality it's a bit more complicated

高見 順 : 娘よりの聞書きにつき誤引用の可能性あり

Donaudampfschiffahrtsgesellschaftsvorsitzender

ich schwÄÑre bei^M meiner MÄÑhre

More about **UTF-8** and language stuff in a later lecture ...

Inverted Index Construction

- Basic idea: map from words to inverted lists

- In (pseudo-) code: `Map<String, Array<int>>`
- Construction then goes as follows:

Iterate over all word occurrences in all records

Maintain record ids in increasing order

For each word occurrence, add id of current record to respective inverted list (create it, if new word)

Let's code this together now !

Along with that, I will introduce you to [Daphne](#), our [coding standards](#), our [SVN](#), our [style checker](#), our continuous build system [Jenkins](#), and so on

Zipf's Law

- How long are the inverted lists?
 - Let N_i be the frequency of the i -th most frequent word
frequency := total number of occurrences
 - Then it turns out that: $N_i \approx \epsilon \cdot 1 / i$ for some constant ϵ
that is, the function $i \rightarrow N_i$ forms a hyperbola
for most text collections and most (word-based) languages
 - This empirical observation is called **Zipf's Law**
(after [George Kingsely Zipf](#), 1902-1950, American linguist)

Let's verify that law on our test collection ...

References

- Text book

 - Introduction to Information Retrieval**

 - C. Manning, P. Raghavan, H. Schütze

 - Available online under <http://nlp.stanford.edu/IR-book>

 - Good, up-to-date, comprehensive information on the basics

- Wikipedia articles relevant for this lecture

 - http://en.wikipedia.org/wiki/Inverted_index

 - http://en.wikipedia.org/wiki/Zipf's_law

 - Wikipedia articles on basic algorithms stuff are quite good

 - However: no good article on list intersection yet, it seems!