Information Retrieval

WS 2012 / 2013

Lecture 1, Wednesday October 24th, 2012

(Introduction, Organizational, Inverted Index)

Prof. Dr. Hannah Bast
Chair of Algorithms and Data Structures
Department of Computer Science
University of Freiburg

UNI FREIBURG

Overview of this lecture

UNI FREIBURG

- Introduction
  - Demos + what you will learn in this course
- Organizational
  - Style of the course
  - Course Systems: Wiki, Forum, Daphne, SVN, Jenkins, ...
  - Exercises + Exam
- And then let's start
  - Inverted Index (INV) ... we will implement one together
  - How to answer arbitrary keyword queries with an INV
  - **Exercise Sheet #1:** implemented a mini search engine that answers two-keyword queries based on an INV

Demos + what you will learn

UNI FREIBURG

- Demos
  - **CompleteSearch** ... fast search-as-you-type search
  - **Broccoli** ... fast and easy-to-use semantic search
  - At the end of the course you will know how to build something like this
  - In particular, you will learn something about:

    index construction, list intersection, ranking, web applications for search, compression, prefix search, error-tolerant search, some machine learning and natural language processing stuff, ontologies, ...

Style of this course

UNI FREIBURG

- What I will do
  - Explain the basics, often by example
  - Underlying theory, wherever needed
  - Give implementation advice + provide code skeletons
- What you will do
  - Implement basic algorithms and data structures
  - Do some experiments
  - Some theoretical tasks ... but not too many
  - Maybe have a look at some of the relevant research papers

Course systems

UNI FREIBURG

- All linked from the Wiki page for the course:
  - **Daphne** is our course management system
  - 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 lecture recordings, the slides, the exercise sheets, and 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

Exercises + Exam

UNI FREIBURG

- 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 exercise sheets are **key** to a real understanding
- Exam in the end
  - You need 50% of the points to be admitted
  - The date of the exam has not been fixed yet, stay tuned ...

Our Coding Standards

UNI FREIBURG

- Please follow these guidelines when writing code
  - Write your programs 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

How much work is it / ECTS points

UNI FREIBURG

- ECTS points = working time
  - You get 6 ECTS points for this lecture
  - That is 6 x 30 = 180 hours of work for the whole course
    - 60 hours for the exam + preparation
    - 120 hours for the lectures + exercise sheets
    - There are 12 lectures with exercise sheets
    - That's about 10 hours per week for this course
    - That's about 8 hours per exercise sheet
  - **Note:** when you have done all the exercise sheets (yourself) you are pretty much fit for the exam, without much more preparation needed

Keyword Search

UNI FREIBURG

- 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:** (will be dealt with in 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

UNI FREIBURG

- 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 one second
    - Query times of ≤ 100 milliseconds feel interactive
  - But already for 1 TB it would be 20 minutes ...
  - Current web: ≈ 50 billion pages / 250 TB of text

    Source: www.worldwidewebsize.com ... assuming 50 KB / page

Inverted Index 2/2

UNI FREIBURG

- 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
  - An interleaved left-to-right scan does it in **linear time**

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

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

Result list: 57, 257, 1345, ...

List Intersection 2/2

UNI FREIBURG

- For more than two lists
  - Also maintain an index for each of the lists
  - Always advance index with currently smallest element
  - Maintain elements at current index positions in a priority queue (PQ)
  - Time $O(N \cdot \log k)$, where N = total #elements, k = #lists
    - each element is looked at exactly once
    - getMin on a PQ with k elements takes time $O(\log k)$
  - There are more sophisticated algorithms ... later lecture

Parsing / Tokenization

UNI FREIBURG

- 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
  - 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

    Donaudampfschifffahrtskapitängesellschaftsvorsitzender

    ich schwÃ¶re bei^M meiner MÃ¶hre
  - For now, let's stick with the simple approach
  - More about UTF8 and language-stuff in a later lecture …

Inverted Index Construction 1/3

UNI FREIBURG

- **Approach 1:** 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 show you a bit about Daphne, our coding standards, SVN, our style checker, Jenkins, ...

Inverted Index Construction 2/3

UNI FREIBURG

- **Approach 2:** one big sort
  - Store lists in an Array<Array<int>>
  - Consider these two example records

    Record 1: uni freiburg ist doof

    Record 2: uni freiburg gar nicht doof
  - Then let the parser output the following, and then sort it:

Inverted Index Construction 3/3

UNI FREIBURG

- Comparison of the two approaches
  - Let N be the total number of all words in all records

    (that is, all word occurrences, not the #distinct words)
  - The map-based approach takes $\Theta(N)$ operations
  - The sort-based approach takes $\Theta(N \cdot \log N)$ operations
  - Looks like a clear win, but in reality it's not so clear:
    - The map-based approach has bad locality:

      The inverted list entry for two subsequent words are written in two completely different places in memory
    - When the data is so large that the lists have to reside on disk during construction, this is even worse
  - More about the importance of locality in a later lecture ...

Zipf's Law

UNI FREIBURG

- How long are the inverted lists?
  - Let N_i be the frequency = number of occurrences of the i-th most frequent word
  - Then it turns out that: $\mathbf{N_i \approx \varepsilon \cdot 1 / i}$ for some constant ε

    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

UNI FREIBURG

- 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/Merge_algorithm

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

  Wikipedia articles on basic algorithms stuff are quite good !