Information Retrieval

WS 2012 / 2013

Lecture 6, Wednesday November 28th, 2012

(Error-tolerant prefix search, Web application)

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

- Organizational
  - Your experiences with ES#5 (approximate matching)
- A search engine web application
  - We will build a toy web app together today
  - You will learn about and see applied:

    HTML, DOM, CSS, JavaScript, jQuery, jQuery UI, AJAX, JSON / JSONP, socket communication, …
  - How to combine prefix search and error-tolerant search
  - Exercise Sheet 6: Build a web app for error-tolerant prefix completion + search as you type

    It won't be hard with what I show you in the lecture today !

Experiences with ES#5 (approx. matching)

UNI FREIBURG

- Summary / excerpts last checked November 28, 16:00
  - Interesting exercise / topic
  - "Impressed with my own code / felt like a magician"
  - To intersect or to merge: that is the question ... wrong instructions first, sorry, intersect works only for prefix search
  - Lecture / slides did not make it clear how to put it all together ... sorry, was clarified later in the forum though
  - Took more time than expected for many, partly because of the intersect / merge confusion
  - "I love proofs" ... yes !!!
  - Part of video cut off ... sorry, new equipment, we try to fix it
  - Afraid of the exam, are there old exams? ... on the Wiki

Results for ES#5 (approx. matching)

UNI FREIBURG

- Summary
  - Construction of 3-gram index took around 1 second

    Number of words **≈ 300K**
  - Average query time: a few **milliseconds**
  - Average ED computation time: around **1 microsecond**
  - Average number of matches: **28**

    ... with one error allowed for every five characters
  - Note: naive algorithm (one ED computation for each of the 300K words) would take **~ 300 milliseconds**

Error-tolerant prefix search 1/3

UNI FREIBURG

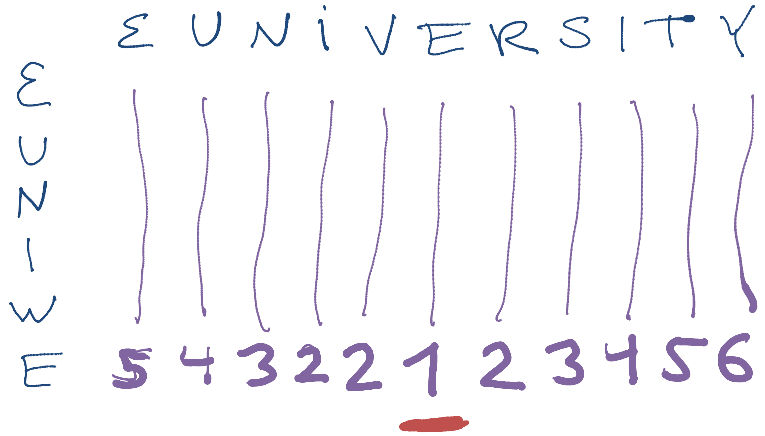
- How to combine the two
  - Note: when typing only part of the word, we don't want approximate matches with the full word, but with a prefix
  - Example: uniwe should match university and universe
  - But ED(uniwe, university) = 6 and ED(uniwe, universe) = 4
  - Solution: use the **prefix edit distance**
  - Definition: $PED(x, y) = \min_{y' \text{ prefix of } y} ED(x, y')$
  - Examples:

    PED(uniwe, university) = ED(uniwe, unive) = 1

    PED(uniwe, universe) = ED(uniwe, unive) = 1

Error-tolerant prefix search 2/3

- How to compute the PED
  - Recall the dynamic programming algorithm for computing ED(x, y)
  - On the way to the final solution, it computes ED(x', y') for all pair of prefixes x' of x and y' of y
  - In particular, the last row of the table contains ED(x, y') for all prefixes y' of y
  - So PED(x, y) is just the minimum of the last row



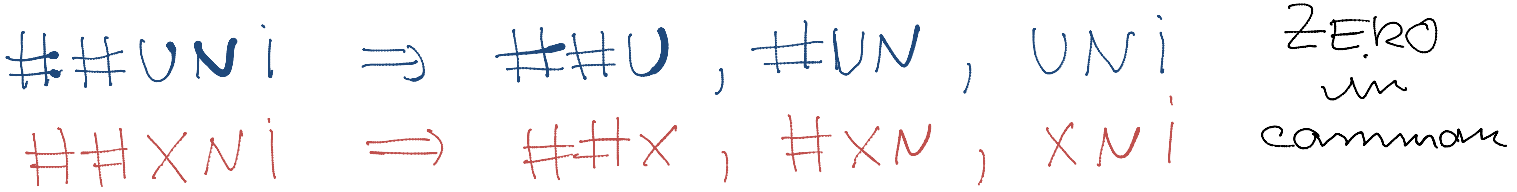
Error-tolerant prefix search 3/3

UNI FREIBURG

- How many k-grams in common ?
  - Assume that PED(x, y) ≤ δ
  - Let x' be x with k-1 # padded to the **left only**, same for y'
  - Then the number of k-grams x' and y' have in common is

    ≥ |x| − k · δ
  - Note: For δ = 1, this is ≥ 1 only for |x| > k
  - Proof sketch: consider x, which has exactly |x| k-grams

    then one tra'fo (insert/delete/replace) "affects" at most k k-grams, and hence δ tra'fos affect at most δ·k k-grams



Components of a search web app

UNI FREIBURG

- Backend
  - Your code to process and answer queries (in Java or C++)
  - Additional code to listen to queries on a given port
  - And to send the result in a form suitable for the frontend
- Frontend
  - Code that runs on the client's browser (in JavaScript)
  - Registers events, like typing a character in the search field
  - Sends queries to the backend
  - Visualizes results sent from backend

Technologies needed 1/2

UNI FREIBURG

- On the side of the backend
  - Socket communication
    - Listen for requests on a given port
    - Parse request string
    - Sent back answer string
  - In C++ easy with boost::asio (asio = asynchronous IO)
  - In Java easy with java.net.Socket / java.net.ServerSocket
  - See code examples in the SVN ... under lectures/lecture-06

Technologies needed 2/2

UNI FREIBURG

- On the frontend side (very briefly, see references for details)
  - HTML: for a web page that holds the map
  - DOM: the elements of the HTML page
  - CSS: the layout of the elements of the HTML page
  - JavaScript: code loaded and run along with the web page
  - jQuery: JavaScript library with lots of useful functions
  - jQuery UI: JavaScript library for all kinds of useful UI components
  - AJAX: sending queries from the client to a server and receiving results **a**synchronously
  - JSON/JSONP: a string containing JavaScript code

Search web application demo

UNI FREIBURG

- We will now write a complete web app together
  - A page with a standard search field
  - Get suggestions after each keystroke
  - Send queries to the backend, receive the results asynchronously, and then display them
  - This will contain **all** the technological elements from the last two slides **...** which you also need for Ex. Sheet 6

References

- JavaScript and CSS
  - http://www.w3schools.com/js/default.asp
  - http://www.w3schools.com/css/default.asp
- jQuery, jQuery UI, Autocomplete
  - http://jquery.com
  - http://jqueryui.com
  - http://jqueryui.com/autocomplete
- AJAX, JSON, JSONP, ...
  - http://en.wikipedia.org/wiki/Ajax_(programming)
  - http://en.wikipedia.org/wiki/JSON
  - http://en.wikipedia.org/wiki/JSONP