Search Engines

WS 2009 / 2010

Lecture 8, Thursday December 10th, 2009

(Error-Tolerant Search)

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

UNI FREIBURG

Goal of Today's Lecture

UNI FREIBURG

- Learn about error-tolerant search
  - it's about typing errors in the query and in the documents
  - a natural word similarity measure: Levenshtein distance
    - definition
    - intuition
    - how to compute it (dynamic programming)
  - given a query word and a large set of words, find the most similar word(s) in the set
  - context-sensitive query correction (Did you mean?)
- Last half hour:
  - open discussion about the exercise sheets
  - how much work they are, how hard, etc.

What is Error-Tolerant Search

UNI FREIBURG

- We query a search engine and don't get results
  - … or few results
  - for example: algoritm
  - Reason 1: we misspelled the query (should be: algorithm)
  - Reason 2: in the document(s) we are looking for, the words are misspelled [show examples]
- Solution for problem due to Reason 1
  - find words that are "similar" to the (misspelled) query words
- Solution for problem due to Reason 2
  - find words that are "similar" to the (correctly spelled) query

in any case, we need to find "similar" words

Similarity Measures

UNI FREIBURG

- Levenshtein distance a.k.a. Edit distance
  - given two strings / words x and y
  - consider the following transformations
    - replace a single character: alkorithm → algorithm
    - insert a character: algoritm → algorithm
    - delete a character: allgorithm → algorithm
  - the Levenshtein or edit distance ED(x, y) is then defined as
    - the minimal number of transformation to get from x to y
    - for example: x = allgorithm y = aigorytm
    - then ED(x, y) = 4

more about other measures later

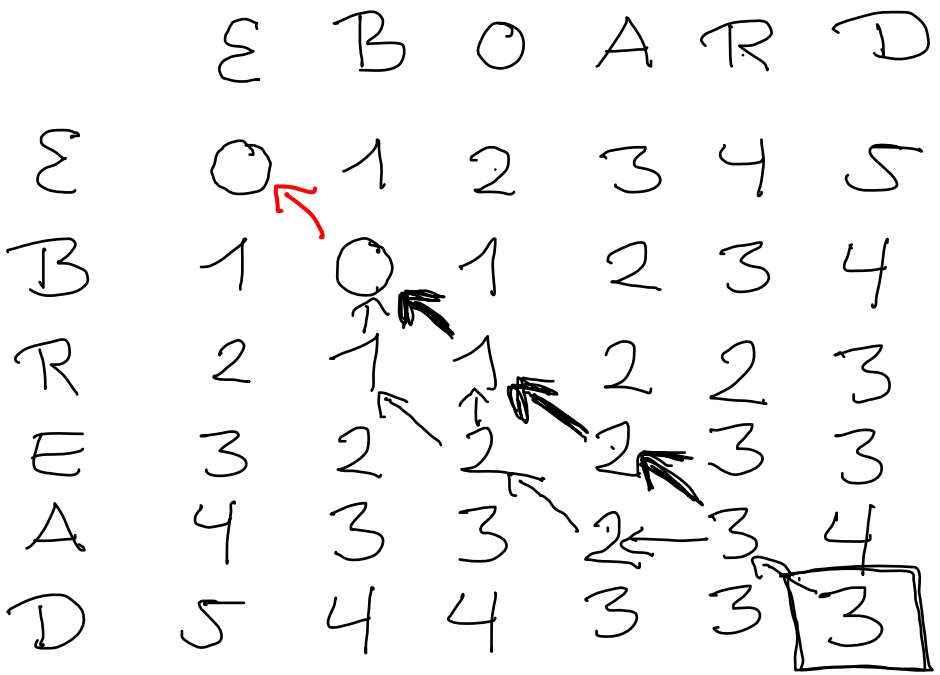
Efficient computation

UNI FREIBURG

- Terminology
  - x(i) = the prefix of x of length i in particular, x(|x|) = x
  - y(j) = the prefix of y of length j in particular, y(|y|) = y
  - ε denotes the empty word
- Recursion
  - it seems that we can compute ED(x,y) recursively from the ED of prefixes of x and y, namely
  - ED(x(i), ε) = i and ED(ε, y(j)) = j
  - for i and j both > 0, we have ED(x(i), y(j)) = the minimum of
    - ED(x(i), y(j-1)) + 1 ~ insert y[j]
    - ED(x(i-1), y(j)) + 1 ~ delete x[i]
    - ED(x(i-1), y(j-1)) + 1 ~ replace x[i] by y[j]
    - ED(x(i-1), y(j-1)) if x[i] = y[j]

An Example

- Let's compute ED("bread", "board")
  - assuming that the recursion from the previous slide is correct (which we will prove on one of the next slides)



Time and Space Complexity

UNI FREIBURG

- Let $|x| = n$ and $|y| = m$
  - then the time complexity is $O(n \cdot m)$
    - $O(1)$ time per table entry
    - and this seems hard to improve in the general case
  - the space complexity is $O(n \cdot m)$
    - again $O(1)$ space per table entry
    - but this can be easily improved
    - we can go column by column and only store the last column
    - or we can go row by row and only store the last row
    - this gives a space complexity of $O(\min(n, m))$

Correctness Proof

UNI FREIBURG

- The recursion looks correct
  - but it's actually not easy to prove that it's correct
  - it is easy to prove that the recursion gives a possible sequence of transformations …

    … and thus an upper bound on the edit distance
  - but it's not clear that it gives an optimal sequence of transformations
  - I will give you the proof idea
  - and a sketch of some parts
  - one important part you will do as an exercise

Proof Outline

- Lemma 1:
  - in an optimal sequence of transformations, if a character is inserted, it is not later deleted again or replaced [next slide]
- Lemma 2:
  - a sequence of transformations for $x \rightarrow y$ is called monotone if the transformations on x occur at strictly increasing positions (except that the next operation after a delete may be at the same position)
  - the recursion computes an optimal monotone sequence of transformations [slide after the next]
- Lemma 3:
  - for each optimal sequence of transformations, there is a monotone one with the same length [Exercise!]

Proof Sketch of Lemma 1

UNI FREIBURG

- Proof sketch of Lemma 1:
  - if a character gets inserted and later deleted again, we can remove both operations and get a shorter sequence
  - if a character gets inserted and later replaced, we can remove the replace and insert the replaced character right away, and thus get a shorter sequence

Proof Sketch of Lemma 2

UNI FREIBURG

- Proof sketch of Lemma 2:
  - proof is by induction on $|x| + |y| = n + m$
  - Case 1: last transformation occurs at position $|y|$
    - then the previous transformation do one of
      - $x(n) \rightarrow y(m-1)$ if last transformation was delete
      - $x(n-1) \rightarrow y(m)$ if last transformation was insert
      - $x(n-1) \rightarrow y(m-1)$ if last transformation was replace
    - and by way of induction these are optimal
  - Case 2: last transformation occurs at position $< |y|$
    - then $x[n] = y[m]$ and these transformations do
      - $x(n-1) \rightarrow y(m-1)$
    - and by way of induction this is optimal

Finding Similar Words in a Large Set

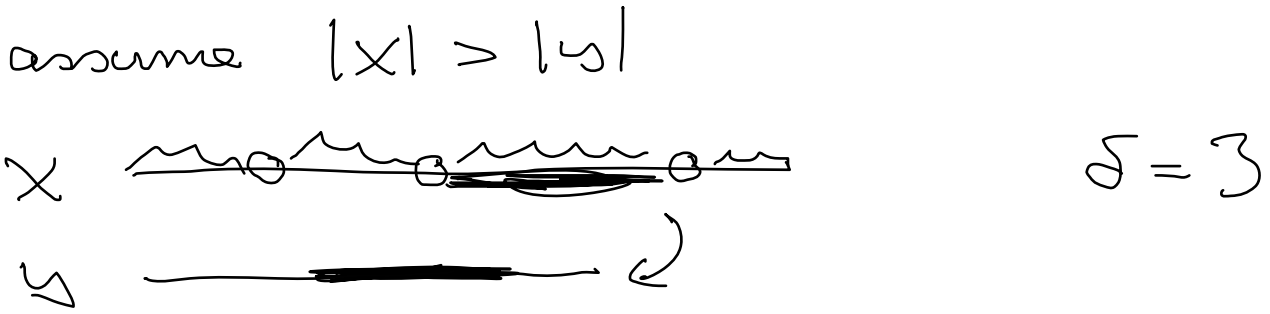
UNI FREIBURG

- Given
  - a query word q
  - a large set of words S
  - a threshold δ
- Find
  - all words in S with edit distance $\leq \delta$ to the query word q
- Naïve algorithm
  - for each word in S compute the edit distance to q
  - one edit distance computation takes around 1 µsec
  - so for 10 million words in S → 10 seconds
  - that is inacceptable as response time for a search engine

Filtering with a Permuterm Index

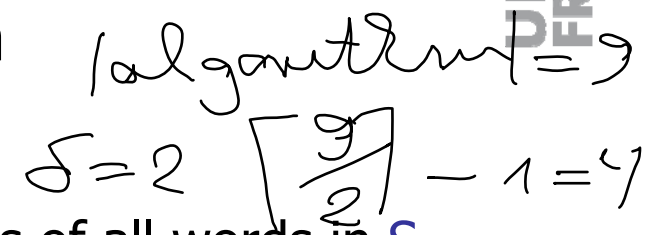
- Given x and y with ED(x, y) ≤ δ
  - then if x and y are not too short they have at least a certain substring in common
  - actually one can prove that there is a rotation x′ of x and a rotation y′ of y such that x′ and y′ have a common prefix of size at least L = ceil(max(|x|, |y|) / δ) − 1

    [where ceil(…) means rounded upwards]
  - it's one of the exercises to prove this!
  - here is an intuitive illustration of why this holds true:



Filtering with a Permuterm Index

- This suggests the following algorithm
  - build a Permuterm index for S
    - that is, compute all possible rotations of all words in S and sort them
  - let L be the size of a common prefix from the slide before
  - then for each rotation of the query word q
    - let q' be the prefix of size L of q
    - find all matches of q'* in the Permuterm index for S
  - for all matches thus found, compute the actual edit distance
  - this will find all words s with ED(q, s) ≤ δ
  - let's see by an example how effective the filtering is …



Filtering with a K-Gram Index

UNI FREIBURG

- Let's build a k-gram index for S
  - that is, for each string of length k have a list of all words in S containing that string as a substring, for example (k=2)

    bo: aboard, about, boardroom, border, …

    or: border, lord, morbid, sordid, …

    rd: aboard, ardent, boardroom, border, …
  - take q = bord as an example query string
  - then using the lists above, we can easily compute:
    - for each word s in S
    - the number of k-grams q and s have in common
    - for example: bord and boardroom …
      - … have exactly two 2-grams in common (bo and rd)

Filtering with a K-Gram Index

UNI FREIBURG

- Jaccard distance between two words x and y
  - the Jaccard coefficient of two sets A and B is defined as

    $J(A, B) = |A \cap B| / |A \cup B|$
  - for example, for $A = \{1, 2, 3\}$ and $B = \{2, 3, 4\}$

    $A \cap B = \{2, 3\}$, $A \cup B = \{1, 2, 3, 4\}$ → $J(A, B) = 1/2$
  - given two words x and y
    - let A be the set of k-grams of x
    - let B be the set of k-grams of y
    - then the k-gram Jaccard distance $J(x, y) = J(A, B)$
  - for example, for x = bord and y = boardroom
    - $A = \{bo, or, rd\}$, $|A \cap B| = 2$ (last slide)
    - $|A \cup B| = 3 + 8 - 2 = 9$ → $J(x, y) = 2 / 9 \approx 0.22$

Filtering with a K-Gram Index

UNI FREIBURG

- So scanning the inverted lists of the k-gram index …
  - … quickly gives us all words with Jaccard distance below a given threshold
  - unfortunately, the Jaccard distance between two words does not always correspond well with their "intuitive" similarity

    Example 1: J("weigh", "weihg") = 2 / 6 = 1/3 (too low)

    Example 2: J("aster", "terase") = 3 / 6 = 1/2 (too high)
  - the k-gram index can also be used to filter out words with too large edit distance
    - if the edit distance between x and y is $\leq \delta$
    - then x′ and y′ must have at least

      $\max(|x'|,|y'|) - 1 - (\delta-1)\cdot k$ k-grams in common

      where x′ and y′ are x and y with k-1 # padded left and right

Context-Sensitive Query Correction

UNI FREIBURG

- The "Did you mean …?" you know from Google & Co
  - the simplest way to realize this is as follows
  - for each query word, find the most similar words as before

    example query: infomatik fribourg

    infomatik: informatik, information, informatics

    fribourg: freiburg, freiburger, friedburg
  - now try out all combinations and suggest the "best" one to the user, where "best" can mean:
    - retrieves the largest number of hits
    - is most frequent in the query log
    - a combination of the two
  - trying out all combinations is too expensive in practice
  - simple trick: precompute statistics about co-occurrence of words