Announcements

- Homework #2 due in two days (February 26)
  - Typo corrected in Problem 5
  - You may work in groups of three, but then you must complete the optional part of either 8(c) or 8(d)
- Midterm next Monday (March 3)
  - Everything up to (including) today’s lecture
  - Open-book, open-notes
- Course project proposal due in 9 days (March 5)
  - By email to Junyang@cs.duke.edu
- Reading assignment
  - Two papers on cache-sensitive indexing, by Rao and Ross, VLDB 1999 and SIGMOD 2000

Keyword search

What are the documents containing both “database” and “search”?
Keywords × documents

<table>
<thead>
<tr>
<th>Keywords</th>
<th>Document 1</th>
<th>Document 2</th>
<th>Document 3</th>
<th>Document 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;cat&quot;</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&quot;database&quot;</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>&quot;dog&quot;</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>&quot;search&quot;</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 means keyword appears in the document
0 means otherwise

- Inverted lists: store the matrix by rows
- Signature files: store the matrix by columns
- With compression, of course!

Inverted lists

- Store the matrix by rows
- For each keyword, store an inverted list
  - \((\text{keyword}, \text{doc-id-list})\)
  - ("database", \{3, 7, 142, 857, …\})
  - ("search", \{3, 9, 192, 512, …\})
  - It helps to sort \text{doc-id-list} (why?)
- Vocabulary index on keywords
  - B+-tree or hash-based
- How large is an inverted list index?

Using inverted lists

- Documents containing "database"
  - Use the vocabulary index to find the inverted list for "database"
  - Return documents in the inverted list
- Documents containing "database" AND "search"
  - Return documents in the intersection of the two inverted lists
- OR? NOT?
What are “all” the keywords?

- All sequences of letters (up to a given length)?
  - … that actually appear in documents!
- All words in English?
- Plus all phrases?
  - Alternative: approximate phrase search by proximity
- Minus all stop words
  - They appear in nearly every document; not useful in search
  - Example: a, of, the, it
- Combine words with common stems
  - They can be treated as the same for the purpose of search
  - Example: database, databases

Frequency and proximity

- Frequency
  - `⟨keyword, ⟨⟨doc-id, number-of-occurrences⟩, ⟨doc-id, number-of-occurrences⟩, …⟩⟩`
- Proximity (and frequency)
  - `⟨keyword, ⟨⟨doc-id, ⟨position-of-occurrence1, position-of-occurrence2, …⟩⟩, ⟨doc-id, ⟨position-of-occurrence1, …⟩⟩, …⟩⟩`
  - When doing AND, check for positions that are near

Signature files

- Store the matrix by columns and compress them
- For each document, store a $w$-bit signature
- Each word is hashed into a $w$-bit value, with only $s < w$ bits turned on
- Signature is computed by taking the bit-wise OR of the hash values of all words on the document

- Does doc contain
  - `hash("database") = 0110` doc contains "database": 0110 "database"?
  - `hash("dog") = 1100` doc contains "dog": 1100
  - `hash("cat") = 0010` doc contains "cat" and "dog": 1110

- Some false positives; no false negatives
Bit-sliced signature files

- Motivation
  - To check if a document contains a word, we only need to check the bits that are set in the word’s hash value.
  - So why bother retrieving all $w$ bits of the signature?
- Instead of storing $n$ signature files, store $w$ bit slices.
- Only check the slices that correspond to the set bits in the word’s hash value.
- Start from the sparse slices.

---

Inverted lists versus signatures

- Inverted lists are better for most purposes (*TODS*, 1998)
- Problems of signature files

---

Suffix arrays (*SODA*, 1990)

- Another index for searching text
- Conceptually, to construct a suffix array for string $S$
  - Enumerate all $|S|$ suffixes of $S$
  - Sort these suffixes in lexicographical order
- To search for occurrences of a substring
  - Do a binary search on the suffix array
Suffix array example

\( S = \text{mississippi} \quad q = \text{sip} \)

<table>
<thead>
<tr>
<th>Suffixes</th>
<th>Sorted suffixes</th>
<th>Suffix array:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mississippi</td>
<td>i</td>
<td>10</td>
</tr>
<tr>
<td>mississippi</td>
<td>ippi</td>
<td>7</td>
</tr>
<tr>
<td>mississippi</td>
<td>isippi</td>
<td>4</td>
</tr>
<tr>
<td>mississippi</td>
<td>ississippi</td>
<td>1</td>
</tr>
<tr>
<td>ssippi</td>
<td>mississippi</td>
<td>0</td>
</tr>
<tr>
<td>ssippi</td>
<td>pi</td>
<td>9</td>
</tr>
<tr>
<td>sippi</td>
<td>ppi</td>
<td>8</td>
</tr>
<tr>
<td>ippi</td>
<td>sippi</td>
<td>6</td>
</tr>
<tr>
<td>ippi</td>
<td>sissippi</td>
<td>3</td>
</tr>
<tr>
<td>i</td>
<td>ssippi</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>issippi</td>
<td>2</td>
</tr>
</tbody>
</table>

No need to store the suffix strings; just store where they start.

One improvement

\[ \diamond \text{Remember how much of the query string has been matched} \]

\( q = \text{sisterhood} \)

\[ \text{low: } \text{missipi...} \quad \text{matched } 3 \text{ characters} \]
\[ \text{middle: } \text{sisterhood...} \quad \text{start checking from the } 4^{th} \text{ character} \]
\[ \text{high: } \text{sistering...} \quad \text{matched } 5 \text{ characters} \]

Another improvement

\[ \diamond \text{Pre-compute the longest common prefix information between suffixes} \]

- For all \((\text{low, middle})\) and \((\text{middle, high})\) pairs that can come up in a binary search

\( q = \text{sisterhood} \)

\[ \text{low: } \text{missipi...} \quad \text{matched } 3 \text{ characters} \]
\[ \text{middle: } \text{sisterhood...} \quad \text{start checking from the } 6^{th} \text{ character} \]
\[ \quad \text{matched } 5 \text{ characters (pre-computed)} \]
\[ \text{high: } \text{sistering...} \quad \text{matched } 5 \text{ characters} \]
Suffix arrays versus inverted lists

- Suffix arrays are more powerful because they index all substrings (not just words)

- Suffix arrays use more space than inverted lists?
  - Check out compressed suffix arrays (STOC 2000)