Humboldt-Universität zu Berlin
Institut für Informatik
HK Text Analytics

n-gram Indexes for Regular Expression Search

Hung Le, Nora Unger
Outline

1. Overview
2. Basics
3. “Usefulness”
4. Multigram Index Construction
5. Shortest Common Suffix Rule
6. Runtime Matching Engine
   • Generation of Logical Access Plan
   • Generation of Physical Access Plan
7. Experiments
Overview

Figure 1. The architecture of FREE
Overview

Figure 2. High level structure of an index
Basics

• *n-gram:*
  
  string $x_1x_2...x_n$ where $x_i$ is a character.

• *multigram / gram:*
  
  *n-gram* where $n$ is arbitrary

• *data unit:*
  
  unit in which the *raw data* is partitioned
  (e.g. web page, paragraph, ...)

---

5
Basics

• selectivity of a gram:
  \[ \text{sel}(x) = \frac{M(x)}{N} \]
  where \( N \) is the overall number of data units and \( M(x) \) is the number of data units which contain \( x \).

• prefix-free set:
  A set of grams \( X \) is prefix-free if no \( x \in X \) is a prefix of any other \( x' \in X \).
“Usefulness”

• For any $c \leq 1$, a gram $x$ is \textit{c-useful} if $\text{sel}(x) \leq c$.
• A gram is \textit{c-useless} if it is not \textit{c-useful}.
• The parameter $c$ is tunable and depends on individual system performance.
“Usefulness”

- The multigram index should contain only useful grams.
- Since every extension of a useful gram is also useful, the index will still be quite large.
- Therefore, only minimal useful grams will be used.
Multigram Index Construction

• *multigram index (FREE)*: prefix-free set of useful grams

```
Algorithm 3.1 Multigram index
Input: database
Output: index: multigram index
Procedure
  [1] \( k = 1 \), expand = \{.\}  // . is a zero-length string
  [2] While (expand is not empty)
  [3] k-grams := all k-grams in database
      whose (k-1)-prefix \( \in \) expand
  [4] expand := \{
  [5] For each gram \( x \) in k-grams
  [6] If \( sel(x) \leq c \) Then  // check selectivity
  [7]     insert(\( x \), index)  // the gram is useful
  [8] Else
  [9]     expand := expand \( \cup \) \{\( x \}\}
  [10] \( k := k + 1 \)
```
Shortest Common Suffix Rule

- **suffix-free set:**
  A set of grams $X$ is **suffix-free** if no $x \in X$ is a suffix of any other $x' \in X$.

- **presuf shell**
  $Y$ is a **presuf shell** of a prefix-free set of grams $X$ if $Y$ is suffix-free, $Y \subseteq X$ and for every $x \in X$, $x \in Y$ or a suffix of $x$ is in $Y$.

- **The presuf shell** of any prefix-free set of grams is unique.
Runtime Matching Engine

1. Identify grams in regex
2. Look up grams from the index
3. Combine the resulting postings lists appropriately
4. Read the identified candidate data units
Algorithm 4.1 Logical index access plan
Input: a regular expression \( r \)
Output: logical index access plan
Procedure

// Generate logical access plan
[1] Rewrite regular expression \( r \) so that it only uses OR (1) and STAR (*) connectives
[2] Construct a parse tree based on rewritten \( r \)
[3] Replace * node with NULL
[4] Remove NULL nodes using Table 2

<table>
<thead>
<tr>
<th>Left child</th>
<th>Regular</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>–</td>
<td>Left</td>
</tr>
<tr>
<td>NULL</td>
<td>Right</td>
<td>NULL</td>
</tr>
</tbody>
</table>

AND Node

<table>
<thead>
<tr>
<th>Left child</th>
<th>Regular</th>
<th>NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>–</td>
<td>NULL</td>
</tr>
</tbody>
</table>

OR Node
Example

\[ r = (\text{Bill} | \text{William}).*\text{Clinton} \]

[1] \[ r = (\text{Bill} | \text{William})(a|b|c)^*\text{Clinton} \]

[2] [3] [4]
Generation of Physical Access Plan

- entries could be unavailable in the index
  1. corresponding gram is a useless gram
  2. gram is useful, but not minimal useful

(a) Generation of physical access plan
(b) Final physical access plan
Experiments

- 700,000 random web pages
- Total size around 4.5 GB

1. MP3 URLs (mp3):
   \(<a\ s+href=s*="|'?)[^>]*\ .mp3\ ('|'?)?>\n
2. US city name, state and ZIP code (zip):
   \((a+s+)*a+s*, s*a\ a+s+d\ d\ d\ d\ d (-d\ d\ d\ d)\ ?\n
3. Invalid HTMLs (html): All html pages with starting tag "("), but with another "")" before the end tag "(" \n   \(<[^>]*\</a>\n
4. Middle name of President Clinton (clinton): \n   william\ s+[a-z]+\ s+clinton

5. Motorola PowerPC chip numbers (powerpc): Motorola PowerPC chip part numbers starts with XPC or MPC followed by digits.
   \motorola.*(xpc|mpc)[0-9]+[0-9a-z]*\n
6. HTML scripts on web pages (script):
   \<script>.*</script>\n
7. US phone numbers (phone):
   \((d/d/d)\ d/d/d/d/d/d/d/d/d/d\ d/d/d/d/d/d/d/d/d\ \d\ d/d/d/d/d\ d/d/d/d/d\ d/d/d/d/d\ d/d/d/d/d\ d/d/d/d/d/d\)

8. SIGMOD papers and their locations (sigmod): The URLs ending with .ps or .pdf and with the word SIGMOD within 200 characters from the URL.
   \<a\ s+href=s*="|'?)[^>]*\ (.ps|\ .pdf\ ('|'?)?>.\{0,200\}sigmod\n
9. Stanford email addresses (stanford):
   \((a|d|l|l|l|l|l)+((a|d)+\.)*stanford\ .edu\n
10. Pages pointing to deep links of eBay (ebay):
    \<a\ s+href=s*="|'?)http://(www\.)?ebay\.com/[^>]+/[>]*("'|'?)?>
Index Construction

<table>
<thead>
<tr>
<th></th>
<th>Complete</th>
<th>Multigram</th>
<th>Suffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Construction time</td>
<td>63 h</td>
<td>8 h 23 min</td>
<td>6 h 10 min</td>
</tr>
<tr>
<td>Number of gram-keys</td>
<td>103,151,302</td>
<td>988,627</td>
<td>64,656</td>
</tr>
<tr>
<td>Number of postings</td>
<td>18,193,048,399</td>
<td>1,744,677,072</td>
<td>820,396,717</td>
</tr>
</tbody>
</table>

- **Complete**: 9 n-gram indexes for n = 2, 3, 4, ..., 10
- **Multigram**: a plain multigram index (without the shortest suffix tree)
- **Suffix**: a multigram index based on a presuf shell
Total Execution Time
Response Time (First 10 Results)
Shortest Suffix Rule

The diagram illustrates the response time (sec) for different file types and names using Plain and Suffix methods. The x-axis represents various file types, while the y-axis shows the response time in seconds. The bars indicate the time taken for each method, with the Plain method shown in dark gray and the Suffix method in light gray. The results show that the Suffix method generally takes less time compared to the Plain method for most file types.