Improving Indexing of Text using the Ziv-Lempel Trie

Luís M. S. Russo* and Arlindo L. Oliveira

November 8, 2005

*Supported by the Portuguese Science and Technology Foundation through program POCTI and project POSI/EEI/10204/2001.
OVERVIEW:

Introduction

Modified LZIndex

Results

Future Work
Introduction:

PROBLEM (Exact Matching): find all occurrences of pattern string P in tree string T.

Suffix Tree are optimal index structures, solve the problem in \(O(m + R)\) time and \(O(u)\) space.

Classical compression algorithms can reduce the space requirements of index structures.

FM-index - Burrows-Wheeler transform, \(O(m + R \log^e u)\) time, 
\(5uH_k + O(u \frac{\log \log u + \sigma \log \sigma}{\log u})\) bits

LZ-index - LZ78 or LZW algorithm, \(O(m^3 \log \sigma + (m + R) \log n)\) time, \(4n \log_2 n(1 + o(1))\) bits
EXAMPLE:

0000111101100101000$
0000 1110 1001 1000
0001 1101 0010 000$
  0011 1011 0101
  0111 0110 1010
  1111 1100 0100
Modified LZIndex:

LZ78 - parsing

0 000111101100101000$
LZ78 - parsing

0 00 0111101100101000$
LZ78 - parsing

0 00 01 11101100101000$
LZ78 - parsing

0 0 0 0 1 1 1 0 1 1 0 0 1 0 1 0 0 0 $
LZ78 - parsing

0 00 01 1 11 01100101000$
LZ78 - parsing

0 00 01 1 11 011 00101000$
LZ78 - parsing

0 00 01 1 11 011 001 01000$
LZ78 - parsing

0 00 01 1 11 011 001 010 00$
LZ78 - parsing

0 00 01 1 11 011 001 010 00$
LZ78 - LZTrie

0 00 01 1 11 011 001 010 00$
DEFINITION: The RevTrie is defined as the tree obtained from the reversed strings of the LZTrie.

0 00 01 1 11 011 001 010 00$
LZ78 - LZTrie and RevTrie (dual correspondence given by colors)

0 00 01 1 11 011 001 010 00$
OBSERVATION:

The RevTrie is an implicit generalized suffix tree.

$10$ $200$ $301$ $41$ $511$ $6011$ $7001$ $8010$ $900$
OBSERVATION: The duality between the LZTrie and the RevTrie can be used to compute suffix links.
LZ78 - left maximal parsing

00001111011001010 00$

```
0
0

1 0
1

1 0 0
1

1 0
$
0 0
```
LZ78 - left maximal parsing

00001111011001 010 00$
LZ78 - left maximal parsing

00001111011 001 010 00$

```
0 0
1 0
1 0
$ 0 0
```
LZ78 - left maximal parsing

00001111 011 001 010 00$
LZ78 - left maximal parsing

000011 11 011 001 010 00$

```
0  
|
1  
|
1  
|
$  
```

```
0  
|
1  
|
1  
|
$  
```

```
0  
|
1  
|
1  
|
$  
```
LZ78 - left maximal parsing

000 011 11 011 001 010 00$
LZ78 - left maximal parsing

0 00 011 11 011 001 010 00$
LZ78 - left maximal parsing

0 00 011 11 011 001 010 00$

Each node now stores a list of blocks.

Lemma: The LZ78 left maximal parsing has at most as many blocks as the LZ78 parsing.

Original: _________________________

LeftMax: _________________________
Searching for pattern P

- Descend and Suffix Walk for $p^R$

- Classify type of match
  
  one P is inside a block

  two P spans 2 blocks

  more than two P spans more than 2 blocks
1: **procedure** `Descend_and_Suffix(P)`
2: \hspace{1em} point $\leftarrow$ Root
3: \hspace{1em} **for** $i \leftarrow m, 0$ **do**
4: \hspace{2em} **while** NOT `Descend?(point, S[i])` **do**
5: \hspace{3em} point $\leftarrow$ `Suffix_Link(point)`
6: \hspace{2em} **end while**
7: \hspace{1em} point $\leftarrow$ `Descend(point, S[i])`
8: \hspace{1em} **end for**
9: **end procedure**
Pat  #  0  0  1  1  1  1
Left
Left–Child
Right
Pat | # | 0 | 0 | 1 | 1 | 1 | 1
---|---|---|---|---|---|---|---
Left | | | | | | | |
Left-Child | | | | | | | |
Right | | | | | | | |
<table>
<thead>
<tr>
<th>Pat</th>
<th>#</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left-Child</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Pat  #  0  0  1  1  1  1
Left
Left-Child
Right
Pat  #  0  0  1  1  1  1  1
Left
Left-Child
Right
Pat  #  0  0  1  1  1  1
Left  □ □ □ □ □ □ □
Left–Child □ □ □ □ □ □ □
Right  □ □ □ □ □ □ □

Break the pattern in all the points until the condition of the while loop is true for the first time.
For this point the only possible case is 1.

No occurrences of type 1 may exist since neither the Left nor the Left–Child overlap the hole pattern.
From this point the only possible cases are 2 and >2.

Since neither the Left nor the Left–child overlap the hole pattern it is not case 2.
Checking case >2

Failed
From this point the only possible cases are 2 and >2.
Since neither the Left nor the Left–child overlap the hole pattern it is not case 2.
Checking case >2

Found

Pat    #  0  1  1  1  1  1  1
Left

Left–Child
Right

0  0  0  0  00  011  11  011  011  00  010  00$
Checking case >2
Failed
Breaking points.
Skipping points that fail.
Either perform a dfs on the LZtrie from the green node or a dfs on the RevTrie from the orange node. Since the subtree of the orange node is smaller we choose the RevTrie.
Pat # 1 1 0 0
Left   □ □ □ □ Failed
Left−Child □ □ □ □
Right   □ □ □ □

0 0 0 0 11 011 011 001 010 00$

00 011 11 011 001 010 00$
Perform a dfs search on the RevTrie that for each visited node starts a dfs on the dual node of the LZTrie.

Searching only for type 1 occurrences of this pattern

<table>
<thead>
<tr>
<th>Pat</th>
<th>#</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left–Child</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Control the growth of the LZTrie by keeping counters at the internal nodes.

Typical maximum value 2.
Results:

Processed Newgroups. Main Memory space.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>20ng-train-all-terms.txt</td>
<td>16 291 Kbytes</td>
</tr>
<tr>
<td>LZIndex</td>
<td>26 607 Kbytes</td>
</tr>
<tr>
<td>Modified - 0</td>
<td>47 244 Kbytes</td>
</tr>
<tr>
<td>Modified - 2</td>
<td>34 605 Kbytes</td>
</tr>
</tbody>
</table>

Approx. +30%.
Results: Report positions. Time in seconds.
Results:

Gutenberg Project. Main Memory Space.

<table>
<thead>
<tr>
<th></th>
<th>txt</th>
<th>LZIndex</th>
<th>Modified - 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>96 413 Kbytes</td>
<td>155 660 Kbytes</td>
<td>205 224 Kbytes</td>
</tr>
</tbody>
</table>

Approx. +32%
Results: Report positions. Time in seconds.
Conclusions:

3 main ideas:

- The RevTrie is an implicit generalized suffix tree

- The text should be left maximal parsed to improve search times

- Controling the groth of the LZTrie
Future Work:

Re-implement the tree data structures.

Store information relative to both trees in one location eliminating duality links.

Store this nodes in bfs time of the revtrie.

pros: Allows to branch in $\log |\Sigma| +$ cache locality.

cons: Uses more space.
Future Work:

Implement lazy Descend and Suffix Walk.

All the queries in the example matched.
Future Work:

Further testing
Acknowledgements:

Professor Gonzalo Navarro for LZ-index code,

Luís Coelho,

Thanks
Questions ?