A survey on the construction algorithms for suffix array type index

Taura lab. M1
48-126435  Shinya Hayashi
Introduction

- The growing amount of text data is now available on the internet
- String matching is a basic technique to extract information from those data
  - ex.) genome analysis, search engine etc.

\[ P = \text{“penguin”} \]

… and their wings have evolved into flippers. Most penguins feed on krill, fish, squid, and other forms of sealife caught while …

http://en.wikipedia.org/wiki/Penguin
Index

- Index is indispensable for rapid pattern search

P = “penguin”

search

return position list [26, 213, ...]

access [26, 213, ...]
Space problem of classical index

- Mainly two classical indexes exist
  - Suffix tree
  - Suffix array
- These indexes use much memory space
  - Much larger than the original text
Compressed index

- Compression technique is effective for space problem of classical index
- How to construct compressed index is the problem
Agenda

- Introduction
- **Suffix array**
- String matching
- Compressed index
- Index construction
- Summary
Suffix array

- Lexicographically sorted suffixes of text T
  - The sentinel '$' is defined to be the smallest character in the alphabet

<table>
<thead>
<tr>
<th>i</th>
<th>SA[i]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11 $</td>
</tr>
<tr>
<td>1</td>
<td>10 i$</td>
</tr>
<tr>
<td>2</td>
<td>7 ippi$</td>
</tr>
<tr>
<td>3</td>
<td>4 issippi$</td>
</tr>
<tr>
<td>4</td>
<td>1 ississippi$</td>
</tr>
<tr>
<td>5</td>
<td>0 mississippi$</td>
</tr>
<tr>
<td>6</td>
<td>9 pi$</td>
</tr>
<tr>
<td>7</td>
<td>8 ppi$</td>
</tr>
<tr>
<td>8</td>
<td>6 sippi$</td>
</tr>
<tr>
<td>9</td>
<td>3 sissippi$</td>
</tr>
<tr>
<td>10</td>
<td>5 ssippi$</td>
</tr>
<tr>
<td>11</td>
<td>2 ssissippi$</td>
</tr>
</tbody>
</table>

We can find any substrings in the text with SA
Space problem

- Suffix array uses $O(n \log n)$ bit of space
  - Much larger than original text
- Though compression is effective, suffix array itself is not so compressible
- Some transformation is required to compress
Burrows Wheeler transform (BWT)

- Sort cyclic shifts of text T
- The same character tends to appear consecutively

```
<table>
<thead>
<tr>
<th></th>
<th>i</th>
<th>SA[i]</th>
<th>bwt</th>
</tr>
</thead>
<tbody>
<tr>
<td>mississippi$</td>
<td>0</td>
<td>11 $mississippi</td>
<td>i</td>
</tr>
<tr>
<td>ississippi$m</td>
<td>1</td>
<td>10 i$mississip</td>
<td>p</td>
</tr>
<tr>
<td>ssissippi$mi</td>
<td>2</td>
<td>7 ippi$mississ</td>
<td>s</td>
</tr>
<tr>
<td>ssissippi$mis</td>
<td>3</td>
<td>4 issippi$miss</td>
<td>s</td>
</tr>
<tr>
<td>issippi$miss</td>
<td>4</td>
<td>1 ississippi$m</td>
<td>m</td>
</tr>
<tr>
<td>ssippi$missi</td>
<td>5</td>
<td>0 mississippi$</td>
<td>$</td>
</tr>
<tr>
<td>sippi$missis</td>
<td>6</td>
<td>9 pi$mississip</td>
<td>p</td>
</tr>
<tr>
<td>ippi$mississ</td>
<td>7</td>
<td>8 ppi$mississi</td>
<td>i</td>
</tr>
<tr>
<td>ppi$mississi</td>
<td>8</td>
<td>6 sippi$missis</td>
<td>s</td>
</tr>
<tr>
<td>pi$mississip</td>
<td>9</td>
<td>3 sissippi$mis</td>
<td>s</td>
</tr>
<tr>
<td>i$mississipp</td>
<td>10</td>
<td>5 ssippi$missi</td>
<td>i</td>
</tr>
<tr>
<td>$mississippi</td>
<td>11</td>
<td>2 ssissippi$mi</td>
<td>i</td>
</tr>
</tbody>
</table>
```
Agenda

• Introduction
• Suffix array
• **String matching**
• Compressed index
• Index construction
• Summary
Backward search

- Pattern can be searched by **using only BWT**
- Seek pattern P from backward
- ex.) Search "iss" in the text "mississippi$"

```
i      SA[i]
0      11  $mississippi  $mississippi  $mississippi
1      10  i$mississippi i$mississippi i$mississippi
2      7   ippi$mississ ippi$mississ ippi$mississ
3      4   issippi$miss issippi$miss issippi$miss
4      1   ississippi$m ississippi$m ississippi$m
5      0   mississippi$ mississippi$ mississippi$
7      8   ppi$mississip ppi$missis ppi$mississippi
8      6   siippi$missip siippi$missi siippi$missis
9      3   sissippi$mi sissippi$mi sissippi$mis
10     5   ssippi$missi ssippi$missi ssippi$missi
11     2   ssissippi$mi ssissippi$mi ssissippi$mi
```

Start with "s"
Start with "iss"
Start with "ss"
**LF-mapping**

- **Mapping from L to F**
  
  \[ LF(i) = C(Li) + Occ(Li, \ i) - 1 \]

- **C(c):** # of character in L which is smaller than c

- **Occ(c, i):** # of character c in L which appears by position i

- The same symbol appears in the same order in both L and F
Agenda

- Introduction
- Suffix array
- String matching
- Compressed index
- Index construction
- Summary
FM-index family

- Compression of BWT
- FM-index family mainly aims to reduce time and space complexity of Occ on backward search
  - Occ(c, i): # of character c in L which appears by position i
  \[ LF(i) = C(L_i) + Occ(L_i, i) - 1 \]
- FM-index, wavelet tree, wavelet matrix etc.
Agenda

- Introduction
- Suffix array
- String matching
- Compressed index
- Index construction
- Summary
FM-Index construction

- One has to construct BWT first
  → Focus on the BWT construction algorithm
- Constructing suffix array uses much memory
  → Direct construction algorithm is required

![Diagram showing the construction process from text to BWT, including the suffix array (SA) and the BWT.](image-url)
Fast bwt in small space by blockwise suffix sorting

Juha Karkkainen,
Theoretical Computer Science,
Sample sort base method

1. Select sample suffixes
2. Sort sample suffixes
3. Set all suffixes in the correct bucket
4. Sort each bucket independently

Construct SA only for each bucket
Long common prefix problem

- It takes too much time to compare two suffixes both of which has long common prefix

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaz
```

Cannot tell which suffix is larger until here!
Difference cover sample (DCS)

- Difference cover \((\text{mod } v)\) is a set such that difference of two elements cover \([0, v)\)
- DCS is a set of integer whose elements are multiple of the elements in DC
- Compute SA of DCS in advance

\[
\begin{array}{ccccccccccccccc}
i & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & \ldots \\
T & \text{bbbbaaaaaaaac} & \ldots \\
DCS & 5 & 4 & 0 & \text{(1, 2)} & 3 & 6 & \ldots \\
\end{array}
\]

The order can be determined here!
Result

- 2.6GHz Intel Pentium 4 processor
- 4GB of main memory

<table>
<thead>
<tr>
<th>Text</th>
<th>Text size = 256 MB</th>
<th></th>
<th></th>
<th></th>
<th>Text size = 1 GB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bwt</td>
<td>dnbwt</td>
<td>MF</td>
<td>BK</td>
<td>bwt</td>
</tr>
<tr>
<td>english</td>
<td>546</td>
<td>-</td>
<td>287</td>
<td>573</td>
<td>2746</td>
</tr>
<tr>
<td>random-64</td>
<td>511</td>
<td>-</td>
<td>241</td>
<td>605</td>
<td>2566</td>
</tr>
<tr>
<td>repeat-64</td>
<td>2994</td>
<td>-</td>
<td>43751</td>
<td>1372</td>
<td>13082</td>
</tr>
<tr>
<td>DNA</td>
<td>585</td>
<td>1974</td>
<td>223</td>
<td>589</td>
<td>-</td>
</tr>
<tr>
<td>random-DNA</td>
<td>574</td>
<td>1876</td>
<td>237</td>
<td>582</td>
<td>2898</td>
</tr>
<tr>
<td>repeat-DNA</td>
<td>2986</td>
<td>12619</td>
<td>70125</td>
<td>1323</td>
<td>12555</td>
</tr>
<tr>
<td>Memory</td>
<td>0.46</td>
<td>0.23</td>
<td>1.3</td>
<td>1.5</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Lightweight data indexing and compression in external memory

Block splitting

- Split text $T$ into blocks so that each block can be loaded to memory
- Compute suffix array and BWT for one block and merge to the result of previous iteration
How to merge BWT

- Read text from back and determine insert point

\[ \text{C('s') + Occ('s', 1) - 1 = 3} \]
Result

- 2.5Ghz AMD Phenom 9850 Quad Core processor (only one CPU was used)
- 3.7GB of RAM
Agenda

- Introduction
- Suffix array
- String matching
- Compressed index
- Index construction
- Summary
Summary

- Explored basic techniques on compressed index for string matching
- BWT is very important data structure for compressed index
  - Introduced some efficient construction algorithms
- Parallel and distributed algorithm will change the situation even dramatically
Thank you!
Backward search

- Iteratively locate the range of cyclic shifts where each suffix starts with the character $P_{i-1}$

\[
sp_{i-1} = C(P_{i-1}) + Occ(P_{i-1}, sp_i - 1)
\]

\[
ep_{i-1} = C(P_{i-1}) + Occ(P_{i-1}, ep_i) - 1
\]
Compressed index

- There are mainly two types of compressed index
  - Compressed suffix array family
    - Compress the inverse function of LF-mapping
    - See my paper
  - FM-index family
    - Compress BWT of text
FM-index

- Compress $T^{BWT}$ as follows
  1. Move-to-front (MTF) transform $T^{BWT}$
  2. Run-length encode zeros
  3. Compress by variable length prefix code
Sampling sort base method

suffixes

S₁
S₂
S₃
. . .
Sₙ

select r-1 elements at random

sort

set as splitters

set in the correct bucket

Sort each bucket independently

B₁ B₂ B₃ .... Bᵣ
Wavelet tree

- A balanced search tree where each symbol from the alphabet corresponds to a leaf
How to search?

- Occ('b', 10)
  - Occ(0, 10) = 6 Six characters go down to left
  - Occ(0, 6 - 1) = 3 Three characters go down to left
  - Occ(1, 3 - 1) = 2 Two 'b's go down to right

The answer is 2