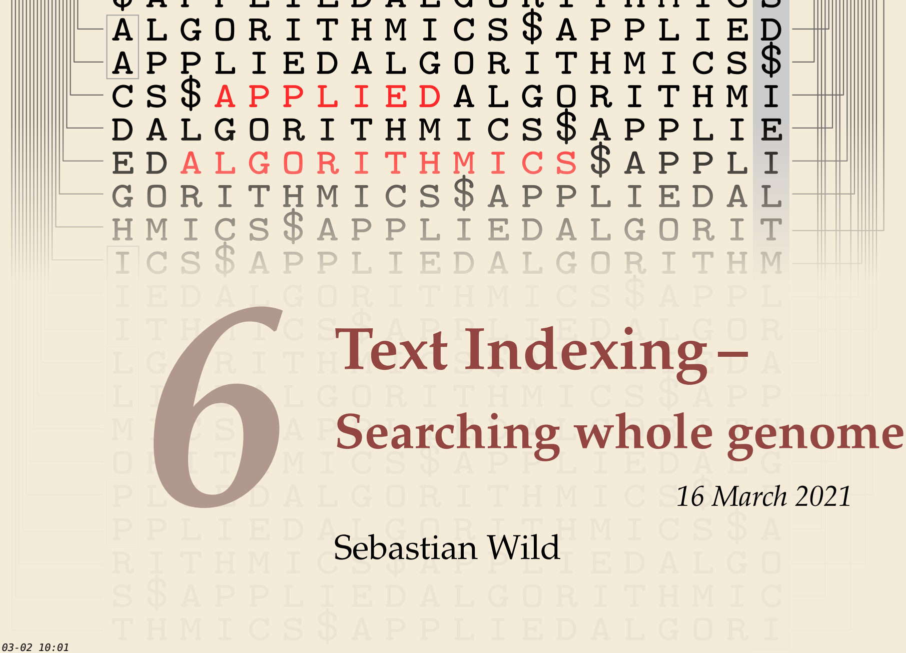


6 Text Indexing – Searching whole genomes

16 March 2021

Sebastian Wild

version `2021-03-02 10:01`

Outline

6 Text Indexing

6.1 Motivation

6.2 Suffix Trees

6.3 Applications

6.4 Longest Common Extensions

6.5 Suffix Arrays

6.6 Linear-Time Suffix Sorting

6.7 The LCP Array

6.1 Motivation

Text indexing

- *Text indexing* (also: *offline text search*):
  - case of string matching: find $P[0..m-1]$ in $T[0..n-1]$
  - but with *fixed* text ⇝ preprocess T (instead of P)
  - ⇝ expect many queries P, answer them without looking at all of T
  - ⇝ essentially a data structuring problem: "building an *index* of T"

    Latin: "one who points out"

- application areas
  - web search engines
  - online dictionaries
  - online encyclopedia
  - DNA/RNA data bases
  - ... searching in any collection of text documents (that grows only moderately)

Inverted indices

same as "indexes"

- original indices in books: list of (key) words ↦ page numbers where they occur
- assumption: searches are only for **whole** (key) **words**

⇝ often reasonable for natural language text

Inverted indices

- original indices in books: list of (key) words ↦ page numbers where they occur

same as "indexes"

- assumption: searches are only for **whole** (key) **words**

⇝ often reasonable for natural language text

Inverted index:

- collect all words in T
  - can be as simple as splitting T at whitespace
  - (actual implementations typically support *stemming* of words
    `goes` ⟶ `go`, `cats` ⟶ `cat`)
- store mapping from words to a list of occurrences ⇝ *how?*

BST?

but $O(\log n)$ time

like a dictionary!

keys = words

values = list of occurences,

Clicker Question

Do you know what a *trie* is?

- **A** A what? No!
- **B** I have heard the term, but don't quite remember.
- **C** I remember hearing about it in a module.
- **D** Sure.

`sli.do/comp526`

Click on "Polls" tab

Tries

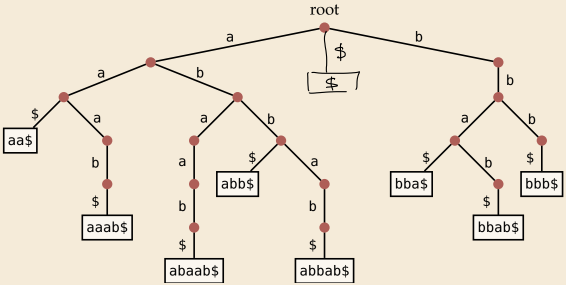
- efficient dictionary data structure for strings
- name from re**trie**val, but pronounced "try" ≈ tree
- tree based on symbol comparisons

- **Assumption:** stored strings are *prefix-free* (no string is a prefix of another)
  - strings of same length ✓
  - strings have "end-of-string" marker $ ✓

some character $\notin \Sigma$

- **Example:** $\Sigma = \{a, b\}$

  `{aa$, aaab$, abaab$, abb$, abbab$, bba$, bbab$, bbb$}, $}`



Clicker Question

Suppose we have a trie that stores n strings over $\Sigma = \{\texttt{A}, \dots, \texttt{Z}\}$. Each stored string consists of m characters.
We now search for a query string Q with $|Q| = q$. $(q \le m)$
How many **nodes** in the trie are **visited** during this **query**?

- **A** $\Theta(\log n)$
- **B** $\Theta(\log(nm))$
- **C** $\Theta(m \cdot \log n)$
- **D** $\Theta(m + \log n)$
- **E** $\Theta(m)$
- **F** $\Theta(\log m)$
- **G** $\Theta(q)$
- **H** $\Theta(\log q)$
- **I** $\Theta(q \cdot \log n)$
- **J** $\Theta(q + \log n)$

`sli.do/comp526`

Click on “Polls” tab

Clicker Question

Suppose we have a trie that stores n strings over $\Sigma = \{\texttt{A}, \ldots, \texttt{Z}\}$. Each stored string consists of m characters.
We now search for a query string Q with $|Q| = q$.
How many **nodes** in the trie are **visited** during this **query**?

- **A** ~~$\Theta(\log n)$~~
- **B** ~~$\Theta(\log(nm))$~~
- **C** ~~$\Theta(m \cdot \log n)$~~
- **D** ~~$\Theta(m + \log n)$~~
- **E** ~~$\Theta(m)$~~
- **F** ~~$\Theta(\log m)$~~
- **G** $\Theta(q)$ ✓
- **H** ~~$\Theta(\log q)$~~
- **I** ~~$\Theta(q \cdot \log n)$~~
- **J** ~~$\Theta(q + \log n)$~~

sli.do/comp526

Click on “Polls” tab

Clicker Question

Suppose we have a trie that stores n strings over $\Sigma = \{\texttt{A}, \ldots, \texttt{Z}\}$. Each stored string consists of m characters.
How many **nodes** does the trie have **in total** *in the worst case*?

A $\Theta(n)$

B $\Theta(n + m)$

C $\Theta(n \cdot m)$

D $\Theta(n \log m)$

E $\Theta(m)$

F $\Theta(m \log n)$

`sli.do/comp526`

Click on “Polls” tab

Clicker Question

Suppose we have a trie that stores n strings over $\Sigma = \{\texttt{A}, \dots, \texttt{Z}\}$. Each stored string consists of m characters.
How many **nodes** does the trie have **in total** *in the worst case*?

A	~~$\Theta(n)$~~	D	~~$\Theta(n \log m)$~~
B	~~$\Theta(n + m)$~~	E	~~$\Theta(m)$~~
C	$\Theta(n \cdot m)$ ✓	F	~~$\Theta(m \log n)$~~

`sli.do/comp526`

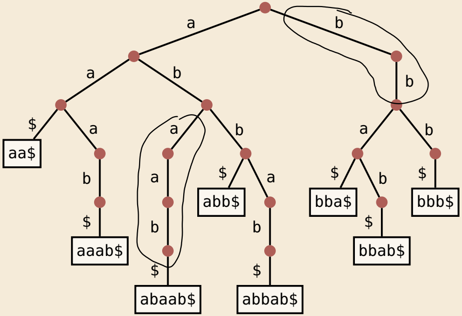
Click on “Polls” tab

Compact tries

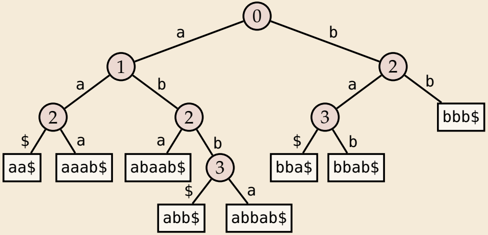
=1 child

- compress paths of unary nodes into single edge
- nodes store index of next character

standard trie



compact trie



⇝ searching slightly trickier, but same time complexity as in trie not $O(n \cdot m)$

- all nodes ≥ 2 children ⇝ #nodes ≤ #leaves = #strings ⇝ linear space $O(n)$

Tries as inverted index

👍 simple

👍 fast lookup

👎 cannot handle more general queries:

- search part of a word
- search phrase (sequence of words)

Tries as inverted index

👍 simple

👍 fast lookup

👎 cannot handle more general queries:

- search part of a word
- search phrase (sequence of words)

👎 what if the ‘text’ does not even have words to begin with?!

- biological sequences

```
ACAAGATGCCATTGTCCCCCGGCCTCCTGCTGCTGCTGCTCTCCGGGGCCACGGCCACCGCTGCCCTGCCCCTGGAGGGTGGCCCCACCGGC
CGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGAAAAGCAGCCTCCTGACTTTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGC
CAGTGCCGGGCCCCTCATAGGAGAGGAAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGCAATCCGCGCGCCGGGACAGAA
TGCCCTGCAGGAACTTCTTCTGGAAGACCTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCACGCAAGTTTAATTACAGACCTGAA
```

- binary streams

```
0000001010100111101011100000111110001111011111001101101000011100010011011110000010001101010
0110110000110101101000000001000000001110101100000100001111010111011001000110010110111011111111
110001010001011001010000001110101010011000000001101100001100111110000101 0101011101111000011
1010111001001010101010000001111101001100000011110011010100000000100100100000010110001100011 0111
```

⇝ need new ideas