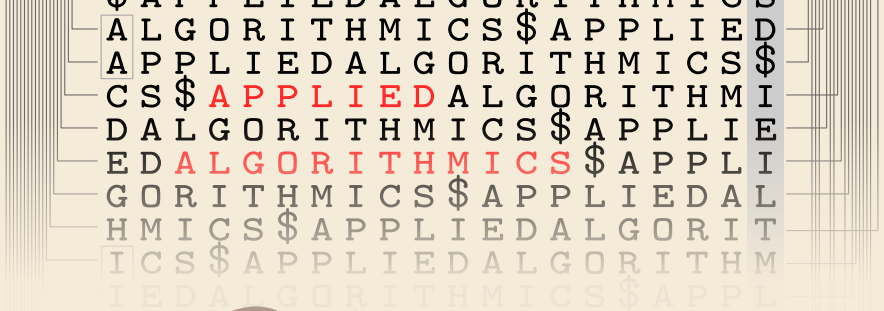


6 Text Indexing – Searching whole genomes

7 March 2022

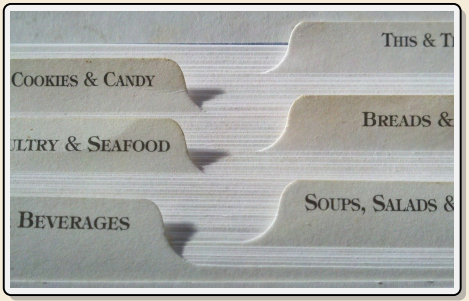
Sebastian Wild

COMP526 (Spring 2022)
University of Liverpool
version **2022-03-07 21:18**

Learning Outcomes

1. Know and understand methods for text indexing: *inverted indices*, *suffix trees*, *(enhanced) suffix arrays*
2. Know and understand *generalized suffix trees*
3. Know properties, in particular *performance characteristics*, and limitations of the above data structures.
4. Design (simple) *algorithms based on suffix trees*.
5. Understand *construction algorithms* for suffix arrays and LCP arrays.

Unit 6: *Text Indexing*



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: Overview
6.7 Linear-Time Suffix Sorting: The DC3 Algorithm
6.8 The LCP Array
6.9 LCP Array Construction

6.1 Motivation

Text indexing

- *Text indexing* (also: *offline text search*):
  - case of string matching: find $P[0..m)$ in $T[0..n)$
  - but with *fixed* text $\rightsquigarrow$ preprocess T (instead of P)
  - $\rightsquigarrow$ expect many queries P, answer them without looking at all of T
  - $\rightsquigarrow$ 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

- original indices (same as "indexes") in books: list of (key) words $\mapsto$ page numbers where they occur
- assumption: searches are only for **whole** (key) **words**

$\rightsquigarrow$ often reasonable for natural language text

Inverted indices

- original indices (same as "indexes") in books: list of (key) words $\mapsto$ page numbers where they occur
- assumption: searches are only for **whole** (key) **words**

$\rightsquigarrow$ 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` $\rightarrow$ `go`, `cats` $\rightarrow$ `cat`
- store mapping from words to a list of occurrences $\rightsquigarrow$ *how?*

could use BST ↳ $\Omega(\log n)$ too slow!

dictionary

keys = words

values = list of starting indices of occurrences

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`

Tries

- efficient dictionary data structure for strings
- name from re**trie**val, but pronounced "try"
- 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**:
  {aa$, aaab$, abaab$, abb$, abbab$, bba$, bbab$, bbb$}