Enhanced and Extended Suffix Arrays

Adrian Regenfuß

Technische Universität München

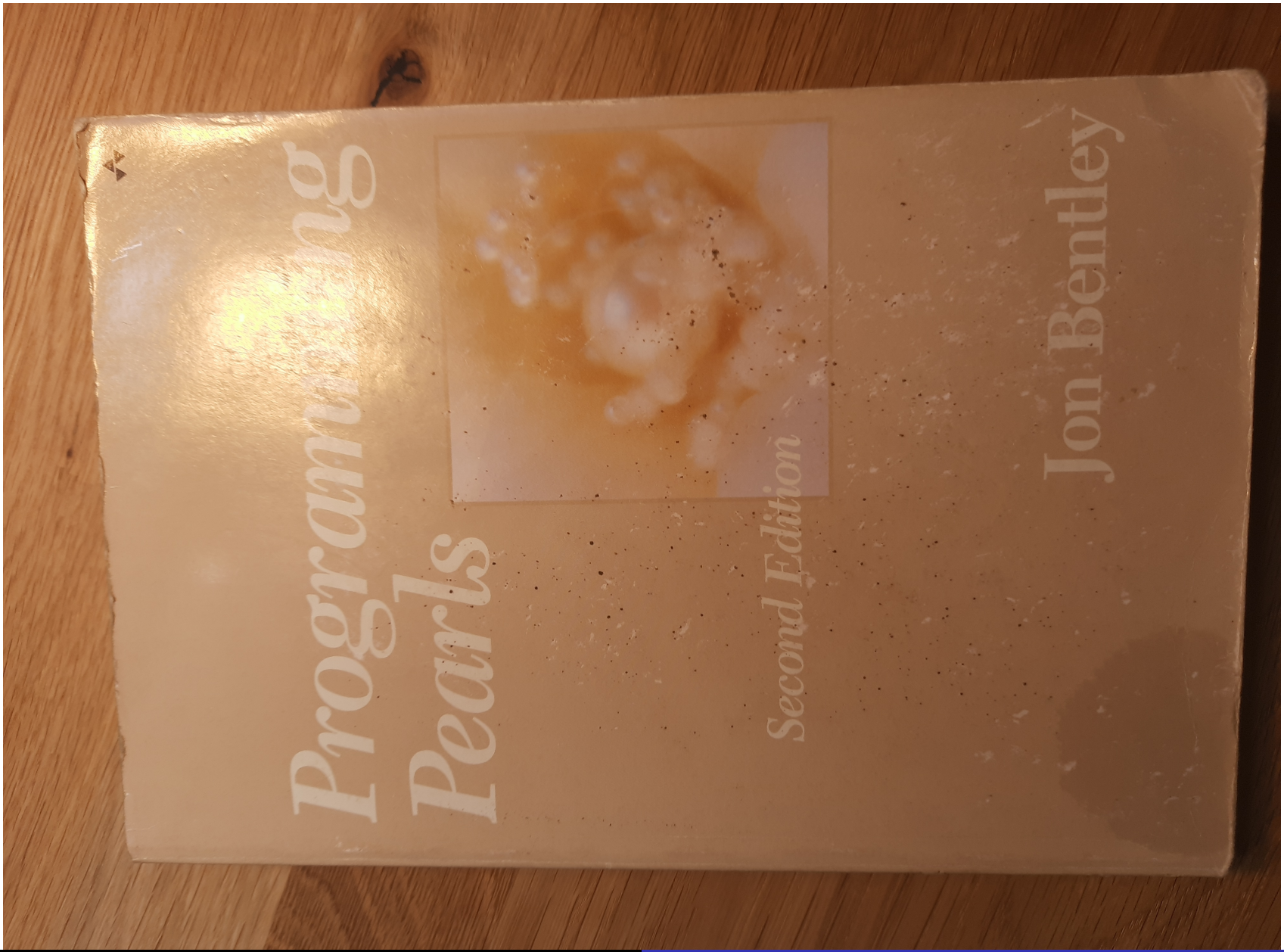
Sommersemester 2020

Übersicht

4 years ago

- Me in Nepal
- No computer with me, only pen, paper and Programming Pearls

My old, damaged copy of Programming Pearls



Thinking a lot about Algorithms

Problems back then

- Thinking about searching, sorting, collision detection, string matching
- Specifically: What is the longest repeated substring of a string?
- Or: what is the longest supermaximal repeat of a string?
- Devising very complex algorithms

Discovering Suffix Arrays

- On the flight back reading the 15th chapter of Programming Pearls
- Describes this exact problem, solved using suffix arrays

Übersicht

Structure

- Description of data structures
- Clarification of terminology
- Description of algorithms
- Discussion of advantages/disadvantages

Übersicht

Contents of an Enhanced Suffix Array

- Suffix Array suftab
- LCP Array lcptab
- Burrows-Wheeler Transformation Array bwttab
- Inverse Suffix Array suftab^{-1}

Suffix Array suftab

Take string over finite alphatbet Σ, e.g. "cag"
Add character "$"(is greater than any other character in Σ)
Sort suffixes
Positions of sorted suffixes in the string are entries of suftab

Example for suftab

For "cag"the suffixes are ("cag$", "ag$", "g$", "$").
The sorted suffixes: ("ag$", "cag$", "g$", "$")
The suffix array then is $[1, 0, 2, 3]$

LCP Array lcptab

LCP Array is the array of the lengths for the prefixes of neighbouring suffixes in the suffix array

More formal for a string S:

$\text{lcptab}[i] = k \Leftrightarrow S[\text{suftab}[i]..\text{suftab}[i] + k] = S[\text{suftab}[i + 1]..\text{suftab}[i + 1] + k]$

Value is 0 for the last suffix.

LCP Array Example

i	suftab	lcptab	SortSuf
0	5	0	acat$
1	1	1	agccacat$
2	7	1	at$
3	4	0	cacat$
4	0	2	cagccacat$
5	6	2	cat$
6	3	1	ccacat$
7	2	0	gccacat$
8	8	0	t$
9	9	0	$

Burrows-Wheeler Transformation Array bwttab

Informally, bwttab contains the character before the suffix referenced in suftab.

bwttab$[i] = c \Leftrightarrow S[suftab[i] - 1] = c$

Empty (or placeholder character) for the suffix that is also S.

With BWT array, the Enhanced Suffix Array contains the whole string.

LCP Array with BWT Array Example

i	suftab	lcptab	bwttab	SortSuf
0	5	0	c	acat$
1	1	1	c	agccacat$
2	7	1	c	at$
3	4	0	c	cacat$
4	0	2		cagccacat$
5	6	2	a	cat$
6	3	1	g	ccacat$
7	2	0	a	gccacat$
8	8	0	a	t$
9	9	0	t	$

Inverse Suffix Array suftab^{-1}

Only sometimes part of the Enhanced Suffix Array. Is, as the name suggests, the inverse of the suffix array.
$S[\text{suftab}^{-1}[\text{suftab}[i]]] = S[i]$.
Example "cag":
$\text{suftab} = [1, 0, 2, 3]$, $\text{suftab}^{-1} = [1, 0, 2, 3]$

LCP-Intervals

ℓ-interval: Roughly an interval in the LCP array where the LCP value is always $\geq \ell$, and which is surrounded by LCP values $< \ell$.

More formally: The interval $[i..j]$ is an LCP-interval of value ℓ (short $\ell - [i..j]$) iff

- lcptab$[i] < \ell$
- lcptab$[k] \geq \ell$ for all $k : i + 1 \leq k \leq j$
- lcptab$[k = \ell$ for at least one $k : i + 1 \leq k \leq j$
- lcptab$[j + 1] < \ell$

Example for an ℓ-Interval

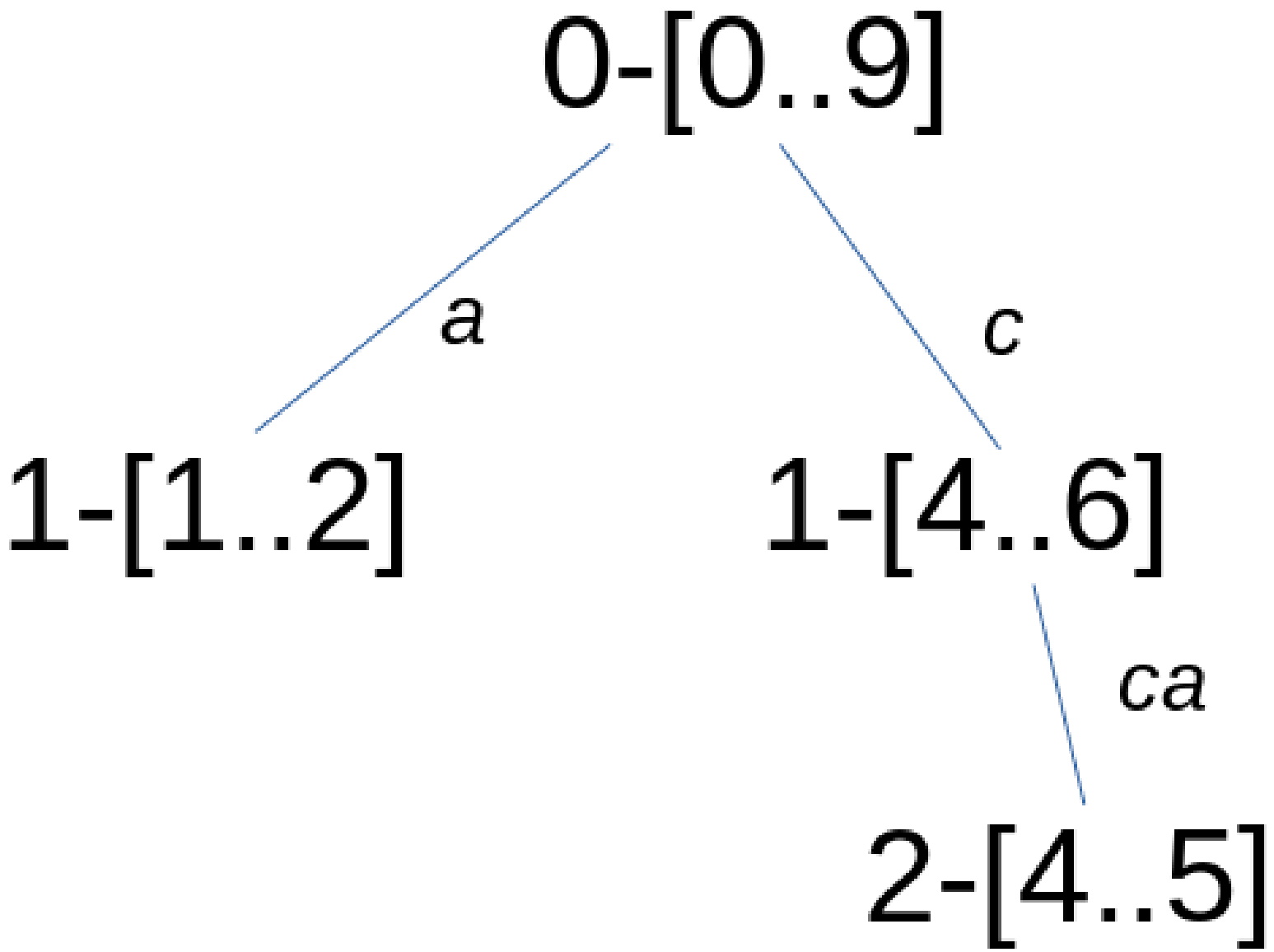
$1 - [4..6]$ is an LCP interval, $2 - [4..5]$ is also an interval (note that they are embedded in each other!

i	suftab	lcptab	SortSuf
0	5	0	acat$
1	1	1	agccacat$
2	7	1	at$
3	4	0	cacat$
4	0	2	cagccacat$
5	6	2	cat$
6	3	1	ccacat$
7	2	0	gccacat$
8	8	0	t$
9	9	0	$

LCP-Interval Trees

LCP-intervals are embedded within each other, a 1-interval can contain a 2-interval.
This forms a so-called LCP-interval tree: the root contains the 0-interval, then the 1-intervals in the next level, & so on.

LCP-Interval Tree for given string



LCP-Intervals not part of the Enhanced Suffix Array

Abouelhoda et al. 2002 stresses repeatedly that neither LCP-intervals nor LCP-interval trees are stored, they are merely inferred by LCP array & BWT array.

Übersicht

Enhanced Suffix Array vs. Extended Suffix Array

Terminology a bit jumbled...
We have:

- Suffix Array: Introduced by Abouelhoda et al. 2002 (but likely used before), only contains suffix array
- Suffix Array with LCP-Array: Very common, also mentioned by Abouelhoda et al. 2002 and Abouelhoda et al. 2002
- Extended Suffix Array: Rarely used, only mentioned by Abouelhoda et al. 2002, seems to be just suffix array with LCP array.
- Enhanced Suffix Array: Introduced by Abouelhoda et al. 2002, **focus of this talk**. Contains suffix array, BWT array, LCP array

In Short

- There's Suffix Arrays
- There's Suffix Arrays with LCP Information=Extended Suffix Arrays
- There's Enhanced Suffix Arrays (Suffix Arrays with LCP & more)

Übersicht

Many Different String Matching Problems

- Exact String Matching
- Finding Maximal/Supermaximal Repeats
- Finding Longest Common Substring (of ≥ 2 strings)

Many of these can be implemented by traversal of LCP-interval tree.

Exact String Matching (Substring Search)

Find all occurrences of T in S.
Or, more formal: Given strings S and T of lengths n and m, $m \leq n$, return indices $I = \{i_1, \ldots, i_k\}$ so that $\forall i \in I : S[i..i+m] = T$

Using only the Suffix Array

There exists an interval in the suffix array (possibly empty) where T is the prefix of the referenced suffixes.
Use binary search to find the starting position of that interval, then another binary search to find the end position.
Second search uses start position of the interval as left starting point.
Runtime $\mathcal{O}(m + \log n)$ (Abouelhoda et al. 2002).

Example

i	suftab	lcptab	bwttab	SortSuf
0	5	0	c	acat$
1	1	1	c	agccacat$
2	7	1	c	at$
3	4	0	c	cacat$
4	0	2		cagccacat$
5	6	2	a	cat$
6	3	1	g	ccacat$
7	2	0	a	gccacat$
8	8	0	a	t$
9	9	0	t	$

Using the Suffix Array with LCP information

Abouelhoda et al. 2002 already describes methods of speeding this binary search up using lcp information:
When the upper and lower bound lie within the same ℓ-interval, comparing T and S[suftab[mid]] is can be started at position $T[\ell]$, S[suftab[mid] $+ \ell$].

Generating the LCP Information

For this, Abouelhoda et al. 2002 proposes pre-computing lcp information for binary-tree-like pairs $((0, n), (0, \frac{n}{2}), (\frac{n}{4}, \frac{3*n}{4})$ etc.), but there are other possible approaches, using e.g. traversal of lcp-interval trees.

Example

T="cac", low $= 3$, high $= 6$.

	i	suftab	lcptab	bwttab	SortSuf
	0	5	0	c	acat$
	1	1	1	c	agccacat$
	2	7	1	c	at$
low →	3	4	0	c	cacat$
mid →	4	0	2		cagccacat$
	5	6	2	a	cat$
high →	6	3	1	g	ccacat$
	7	2	0	a	gccacat$
	8	8	0	a	t$
	9	9	0	t	$

Gusfield's Super-Accelerant

Abouelhoda et al. 2002, pp. 152 describes a further accelerant to searching using lcp data.
Note that these accelerants don't improve worst-case performance, which is still $\mathcal{O}(m + \log n)$.

Personal Idea

Personal idea: use Interpolation Search (Abouelhoda et al. 2002) for searching the suffix array.
Reasons: runs in $\mathcal{O}(\log \log n)$ average case, and genomic data seems uniformly distributed. Maybe look into this.

Traversing the LCP-Interval Tree

Finding Maximal Repeats

Finding Supermaximal Repeats

Finding Longest Common Substrings

Finding Maximum Unique Matches

Übersicht

Of the Suffix Array

Of the LCP Array

Of the BWT Array

Übersicht

Motivation

First explain, then convince

- Suffix tree very succesful & fast, but uses a lot of memory
- Often used in Genome search/alignment
- Use about 20 bytes per input character (Abouelhoda et al. 2002)
- Need datastructure with same time complexity as suffix tree, but lower space requirements

Advantages of Enhanced Suffix Arrays

- $6n$ bytes per input character ($4n$ for the suffix array, n for the lcp array, n for bwttab)
- Caveat: works only with strings shorter than 2^{32} bytes, and lcp values < 255 (although there are some workarounds here, described by Abouelhoda et al. 2002)
- Construction & algorithms same time complexity as suffix tree for most matching problems
- Better cache coherence (due to linear layout in memory)
- Empirical speedups & easier to implement

Implementations

Used in the software Vmatch, see http://www.vmatch.de

Maybe I'll have enough time to implement with interpolation search this summer :-)

Questions

?

Übersicht

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. The enhanced suffix array and its applications to genome analysis. In *International Workshop on Algorithms in Bioinformatics*, pages 449–463. Springer, 2002.

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. Replacing suffix trees with enhanced suffix arrays. *Journal of discrete algorithms*, 2(1):53–86, 2004.

Dan Gusfield. Algorithms on strings, trees, and sequences. 1997. *Computer Science and Computational Biology. New York: Cambridge University Press*, 1997.

Stefan Kurtz. Reducing the space requirement of suffix trees. *Software: Practice and Experience*, 29(13):1149–1171, 1999.

Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *siam Journal on Computing*, 22(5): 935–948, 1993.

Yehoshua Perl, Alon Itai, and Haim Avni. Interpolation search—a log log n search. *Communications of the ACM*, 21(7):550–553, 1978.

Mikaël Salson, Thierry Lecroq, Martine Léonard, and Laurent Mouchard. Dynamic extended suffix arrays. *Journal of Discrete Algorithms*, 8(2):241–257, 2010.