Some Theory and Practice of Greedy Off-Line Textual Substitution

Alberto Apostolico   Stefano Lonardi

Purdue University and Università di Padova

Lossless Compression by Textual Substitution

the optimum encoding problem for most macro schemes is $\mathcal{NP}$-complete [Storer, Szymanski 82]

⇕

steepest descent OFF-LINE scheme

⇕

LZ macro schemes ([Ziv, Lempel 77], [Ziv, Lempel 78])

- have linear time implementations (e.g., [Rodeh, Pratt, Even 81])
- are highly constrained (unidirectional pointers, . . .)

Findings

- uniform improvement over PACK (Huffman) and COMPRESS (LZ-78)
- improvement over GZIP (LZ-77) and BZIP [Burrows, Wheeler 94] for highly random inputs (e.g., genetic sequences)
- computationally intensive
- viable to parallel implementation where advantageous
- some unexpected tradeoffs
- some interesting algorithmic and programming problems

Overall structure of OFF-LINE

x = $<$ read the original text $>$;

repeat

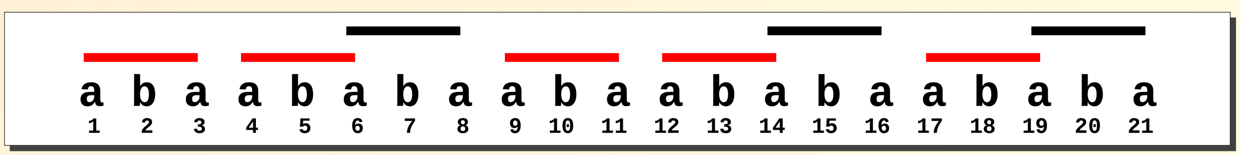
D = $<$ build a data structure containing, for every substring of the text x, the number of its non overlapped occurrences $>$;

s = $<$ choose from D the substring that maximizes the compression $>$;

x = $<$ substitute all the occurrences of s in x $>$;

until $<$ no further compression of x can be obtained $>$;

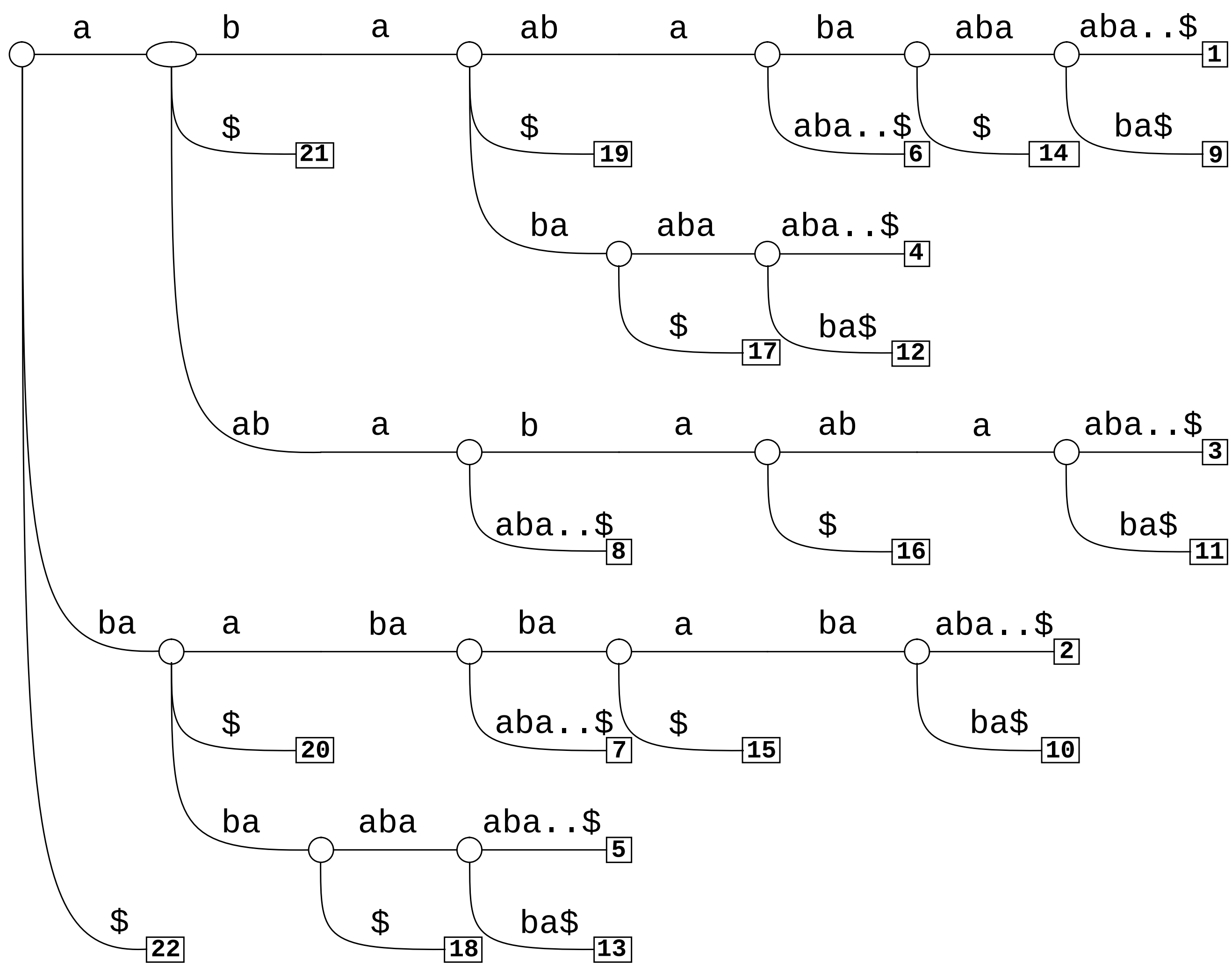
$<$ run Huffman on the encoding $>$;



File	*Size* (bytes)	Huffman PACK	LZ-78 COMPRESS	OFF-LINE	LZ-77 GZIP	BWT BZIP2
bib	111,261	5.23	3.34	2.98	2.52	1.97
book1	768,771	4.56	3.45	3.43	3.26	2.42
book2	610,856	4.82	3.28	2.88	2.70	2.06
geo	102,400	5.69	6.07	5.57	5.35	4.44
news	377,109	5.22	3.86	3.26	3.07	2.51
obj1	21,504	6.07	5.22	4.45	3.84	4.01
obj2	246,814	6.30	4.17	3.50	2.64	2.47
paper1	53,161	5.03	3.77	3.29	2.79	2.49
paper2	82,199	4.64	3.51	3.19	2.89	2.43
pic	513,216	1.66	0.96	0.96	0.87	0.77
progc	39,611	5.25	3.86	3.29	2.68	2.53
progl	71,646	4.81	3.03	2.50	1.81	1.73
progp	49,379	4.91	3.11	2.70	1.82	1.73
trans	93,695	5.57	3.26	2.40	1.62	1.52
average	224,402	4.98	3.63	3.17	2.70	2.36
mito	78,521	1.84	1.82	1.73	1.97	1.84
chrI	230,195	2.19	2.18	2.16	2.30	2.16
chrVI	270,148	2.19	2.18	2.17	2.33	2.18

How to . . .

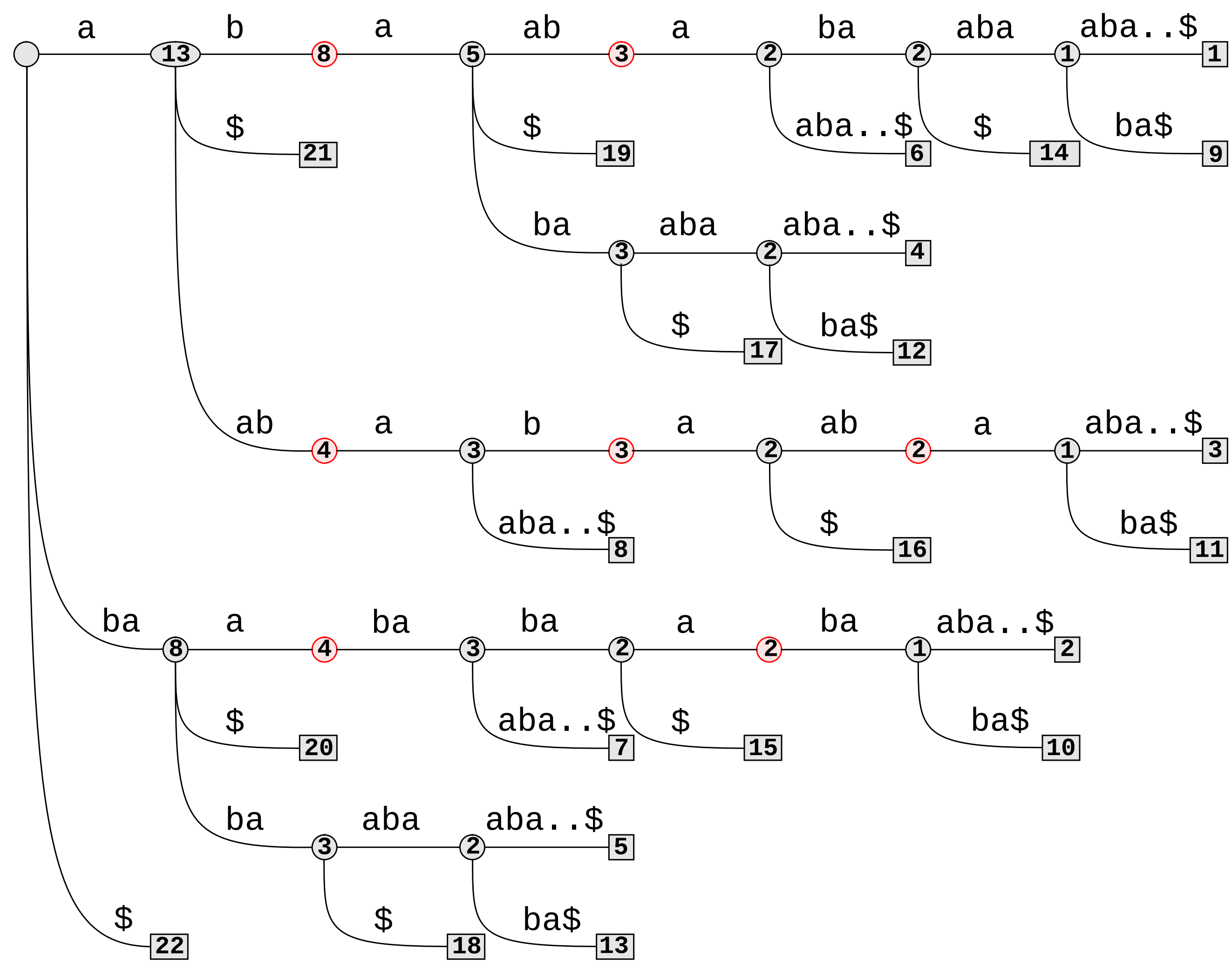
- . . . count the number of non overlapped occurrences of each substring
  - ➠ augmented suffix tree
- . . . search and substitute all the occurrences of a particular substring
  - ➠ balanced tree of text fragments



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**$**

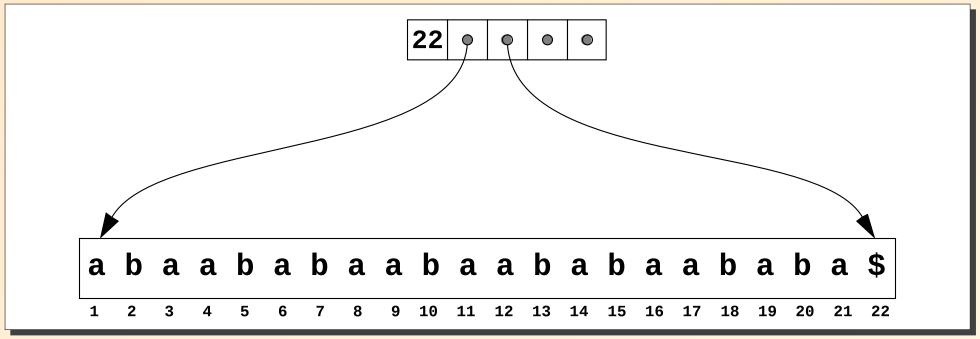
Augmented Suffix Tree

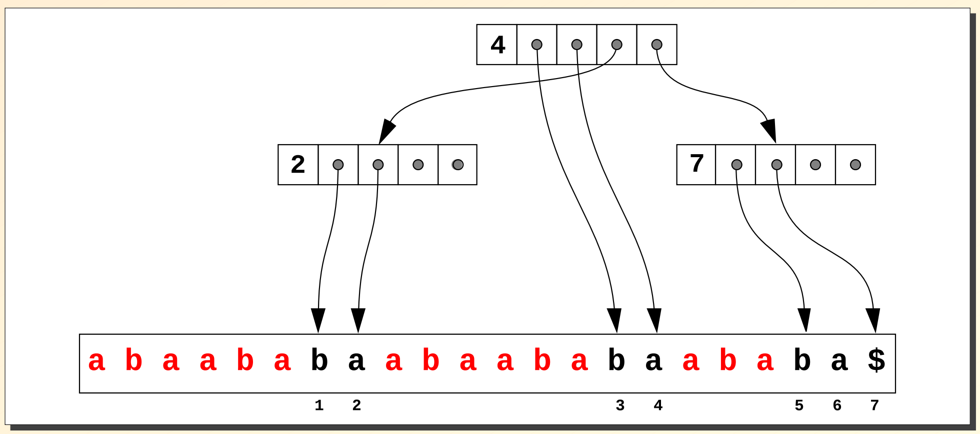
- collects compactly all the suffixes of a string and counts the number non-overlapped occurrences
- construction: brute force $O(n^2)$; clever $O(n \log^2 n)$ [Apostolico, Preparata 96]
- query: $O(m)$
- space: $O(n \log n)$ (probably $O(n)$ [Mignosi, Breslauer p.c.])
- brute force construction: on average $O(n \log n)$



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**$**

Balanced Tree of Text Fragments





Choosing and Computing a Gain measure (1/2)

Leave one of the f_w non overlapped occurrences of w in the text, substitute the other $f_w - 1$ with a pointer to the original one

Assume an integer z can be encoded with $l(z)$ bits, $m_w = |w|$, $B =$ the average length of a symbol in bits

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a	$

$$B f_w m_w$$

(9,3)			(9,3)			b	a	a	b	a	(9,3)			b	a	(9,3)			b	a	$
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

$$B m_w + (f_w - 1)(1 + l(n) + l(m_w)) + 1$$

Choosing and Computing a Gain Measure (2/2)

Remove *all* the f_w non overlapped occurrences of w in the text, save w, $m_w = |w|$, f_w and the list of occurrences, compact the text

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	b	a	a	b	a	b	a	a	b	a	a	b	a	b	a	a	b	a	b	a	$

$$Bf_w m_w$$

b	a	b	a	b	a	$
1	2	3	4	5	6	7

a b a	1 4 9 12 17
3	5

$$Bm_w + l(m_w) + l(f_w) + f_w l(n)$$

Remark: compute G only at explicit nodes of the tree

OFF-LINE variants

- Q substrings selection/substitution are performed between two consecutive updates of the tree
  - ➠ OFF-LINE-SLOPPY
- all the prefixes of the current selection are replaced if capable to produce compression
  - ➠ OFF-LINE-PREF
- only substrings which have length less than H are considered
  - ➠ OFF-LINE-PRUNED

OFF-LINE-SLOPPY on `mito` (size 78521)

Heap Size (Q)	Substitutions	Trees	Ratio	Time	size	%
1	787	788	1.0	100.0%	32,798	100.00%
10	799	83	9.6	12.11%	32,837	100.11%
100	910	13	70.0	4.21%	33,113	100.96%
1,000	1,174	4	293.5	4.44%	33,688	102.71%

OFF-LINE-SLOPPY on `paper2` (size 82201)

Heap Size (Q)	Substitutions	Trees	Ratio	Time	size	%
1	165	165	1.0	100.0%	17,074	100.0%
10	170	22	7.7	14.8%	17,141	100.4%
100	303	7	43.3	7.06%	17,440	102.1%
1,000	619	3	206.3	8.86%	17,861	104.6%

File	*Size* (bytes)	*Iterations* OFF-LINE
`bib`	111,261	927
`book1`	768,771	5,255
`book2`	610,856	4,193
`geo`	102,400	764
`news`	377,109	2,902
`obj1`	21,504	215
`obj2`	246,814	1,751
`paper1`	53,161	663
`paper2`	82,199	811
`pic`	513,216	113
`progc`	39,611	537
`progl`	71,646	611
`progp`	49,379	453
`trans`	93,695	616
`mito`	78,521	170
`chrI`	230,195	77
`chrVI`	270,148	35

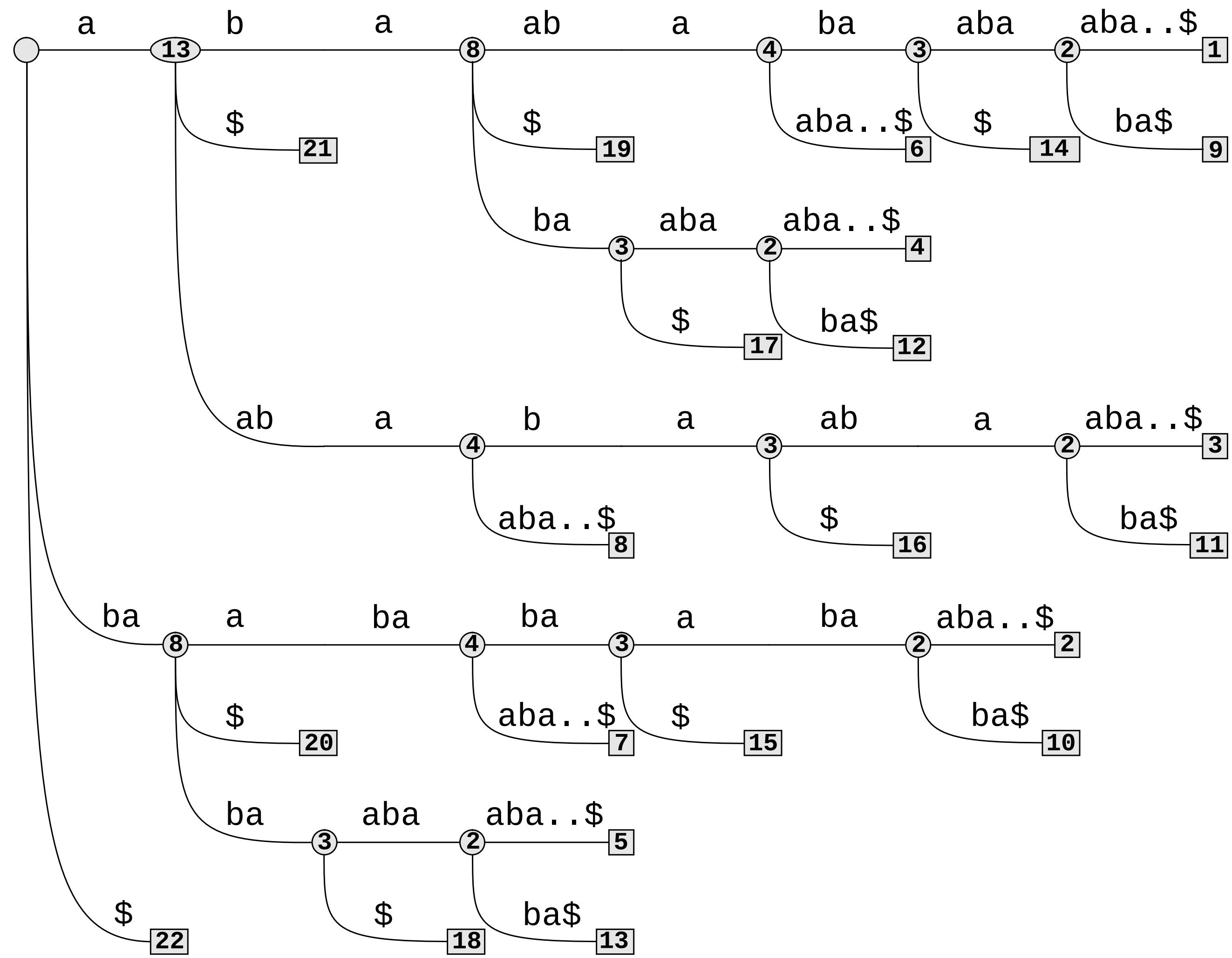
Final remarks

- Data structures and algorithms
  - parallel implementation
  - update the (pruned) augmented suffix tree
- Empirical studies
  - fine-tune the function G
  - reiterate the compression on the substrings removed
  - experiment other encodings (arithmetic, move to front)
  - hybrid with other schemes

Suffix Tree

- collects compactly all the suffixes of $x\$$
- construction: brute force $O(n^2)$; clever $O(n)$ [Weiner 73], [McCreight 76], [Ukkonen 95] - in parallel $O(\log n)$ using n processors [AILSV 83]
- query time: $O(m)$
- space: $O(n)$
- brute force construction: on average $O(n \log n)$ (e.g. [Aho, Hopcroft, Ullman 74], [Apostolico, Szpankowski 92], [Chang, Lawler 94])
- occurrences of a substring w = leaves reachable from the node rooted at w

But . . . we need the statistic of non overlapped occurrences



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
a	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**a**	**b**	**a**	**b**	**a**	**$**

Allocating the Augmented Suffix Tree

- structure of the node (array, linked list, balanced search tree, global hash table)
- space considerations (linked list 20 bytes per node), avg 1.5 node per symbol ➠ avg 30 bytes per symbol (bps)
  - two indices $[i, j]$
  - one pointer to list of children
  - one pointer to list of siblings
  - one counter for the number of non overlapped occurrences
- variations (Patricia 12bps [Morrison 68], suffix-array 6bps [Manber, Myers 93], suffix-cactus 9bps [Kaerkkaeinen 95], level compressed trie 11bps [Andersson, Nilsson 95])

Dynamic text and statistics indexing problem

- the augmented suffix tree is a suitable data structure for our needs
- how the tree is modified if we delete a char in the text?
- what happens if we delete *all* the occurrences of a substring?
- is there an algorithm to “update” efficiently the tree and its statistics?

➪ dynamic text problem [McCreight 76], [Fiala, Green 89], [Gu, Farach, Beigel 94], [Ferragina 97]