

Contents

I	What, Why and How	1
1	Introduction	2
1.1	Publications derived from the Thesis	4
1.2	Awards	6
1.3	Research Stays and Internships	6
2	Basic Concepts	7
2.1	Computation model	7
2.2	Sequences, Documents and Collections	7
2.3	Empirical Entropy	8
2.4	Huffman Codes	8
2.5	Integer Encoding Mechanisms	9
2.5.1	Bit-aligned Codes	10
2.5.2	Byte-aligned Codes	11
2.6	Rank, Select and Access	13
2.6.1	Binary Sequences	13
2.6.2	General Sequences	15
2.7	Wavelet Tree	16
2.7.1	Data Structure	16
2.7.2	Rank, Select and Access	16
2.7.3	Wavelet Trees and Grids	18
2.7.4	Two-Dimensional Range Queries	19
2.8	Directly Addressable Codes	20
2.9	K^2 -tree	21
2.10	Compact Trees	24
2.11	Differentially Encoded Search Trees	25
2.12	Range Maximum Queries	26
2.13	Treaps and Priority Search Trees	27
2.14	Text Indexes	27
2.14.1	Generalized Suffix Tree and Suffix Array	28
2.14.2	Self Indexes	30
2.14.3	Inverted Indexes	33
2.15	Muthukrishnan's Algorithm	33
2.16	Information Retrieval Concepts	35
2.16.1	Scoring	35
2.16.2	Top- k Query Processing and Two-stage Ranking Process	37

II	Inverted Indexes	38
3	Query Processing and Compression of Posting Lists	39
3.1	Query Processing	39
3.1.1	Early Termination	40
3.1.2	Index Organization	40
3.1.3	Query Processing Strategies	40
3.2	Algorithms for Top- k Ranked Retrieval	41
3.2.1	Weak-And (WAND)	42
3.2.2	Block-Max WAND	44
3.2.3	Persin et al.'s Algorithm	46
3.2.4	Other Approaches	46
3.3	Dual-Sorted Inverted Lists	48
3.3.1	Basic operations	48
3.3.2	Complex Operations	50
3.4	Compression of Posting Lists	52
4	Inverted Treaps	54
4.1	Inverted Index Representation	55
4.1.1	Construction	55
4.1.2	Compact Treap Representation	56
4.1.3	Representing the Treap Topology	57
4.1.4	Practical Improvements	61
4.2	Query Processing	63
4.2.1	General Procedure	63
4.2.2	Intersections	64
4.2.3	Unions	67
4.2.4	Supporting Different Score Schemes	67
4.3	Incremental Treaps	69
4.3.1	Supporting Insertions	70
4.3.2	Gradual Growth	71
5	Experimental Evaluation	73
5.1	Datasets and Test Environment	73
5.1.1	Baselines and Setup	74
5.1.2	Dual-Sorted Implementation	74
5.1.3	Inverted Treaps Implementation	75
5.2	Results	75
5.2.1	Index Size	75
5.2.2	Construction Time	77
5.2.3	Ranked Union Query Processing	78
5.2.4	Ranked Intersection Query Processing	80
5.2.5	One-word Queries	89
5.2.6	Incremental Treaps	89
5.3	Discussion	91

III	Practical Top-k on General Sequences	94
6	Top-k Document Retrieval on General Sequences	95
6.1	Linear Space Solutions	96
6.1.1	Hon et al. solution	96
6.1.2	Optimal solution	97
6.2	Practical Implementations	98
6.2.1	Hon et al.	98
6.2.2	GREEDY	100
6.2.3	NPV	100
6.2.4	SORT	101
6.2.5	Patil et al.	101
6.2.6	Other approaches	102
6.2.7	Summary	102
7	Top-k on Grids	104
7.1	Wavelet Trees and RMQ	104
7.2	K^2 -treap	106
7.2.1	Local Maximum Coordinates	107
7.2.2	Local Maximum Values	108
7.2.3	Tree Structure	108
7.2.4	Top- k Query Processing	109
7.2.5	Other Supported Queries	109
8	Practical Implementations	110
8.1	A Basic Compact Implementation	110
8.1.1	Searching for Patterns	110
8.1.2	Mapping to the Grid	110
8.1.3	Finding Isolated Documents	111
8.1.4	Representing the Grid	111
8.1.5	Representing Labels and Weights	111
8.1.6	Summing Up	112
8.1.7	Answering Queries	112
8.2	An Improved Index	114
8.2.1	Mapping the Suffix Tree to the Grid	114
8.2.2	Smaller Grid Representations	116
8.2.3	Efficient Construction for Large Collections	117
8.2.4	Summing Up	117
9	Experimental Evaluation	118
9.1	Datasets and Test Environment	118
9.2	Implementations	119
9.2.1	Baselines	119
9.2.2	Implementation of WRMQ	120
9.2.3	Implementation of K^2 Treap ^H	121
9.2.4	Implementation of W1RMQ ^H	121
9.3	Results	121

9.3.1	Space	122
9.3.2	Retrieval Speed Varying k	124
9.3.3	Varying Pattern Length	129
9.3.4	Word Alphabet Collections	131
9.4	Discussion	133

IV Extensions and Evangelization 139

10 Aggregated 2D Queries on Clustered Points 140

10.1	Motivation	140
10.2	Augmenting the K^2 -tree	142
10.2.1	Supporting Counting Queries	143
10.3	Experiments and Results	145
10.3.1	Experiment Setup	146
10.3.2	Space Comparison	147
10.3.3	Query Processing	148
10.4	Discussion	150

11 Web Access Logs 152

11.1	Motivation	152
11.2	Our Index	154
11.2.1	Construction	154
11.2.2	Queries	155
11.3	Experiments and Results	157
11.3.1	Setup and Implementations	157
11.3.2	Experimental Data	157
11.3.3	Space Usage	158
11.3.4	Time Performance	158
11.4	Discussion	160

12 PcapWT: Indexing Network Packet Traces 161

12.1	Motivation	161
12.2	Related Work	162
12.3	The Design of pcapWT	163
12.3.1	Building Index Data	163
12.3.2	Packet Lookup and Extraction	163
12.4	Performance Evaluation	166
12.4.1	Packet Traces Datasets	166
12.4.2	Baselines	168
12.4.3	Space	168
12.4.4	Multi-threaded File I/O	169
12.4.5	Packet Extraction	169
12.5	Discussion	171

V	And Finally...	174
13	Conclusions and Future Work	175
13.1	Summary of Contributions	175
13.2	Future Work	176
	Bibliography	177

List of Tables

- 2.1 Encoding positive integers using different bit-aligned encoding techniques. 11
- 2.2 Different configurations for Simple9 depending on the selector value b 12
- 2.3 List of compact tree operations 25

- 5.1 Ranked union results grouped by percentiles. 80
- 5.2 Ranked intersection results grouped by percentiles. 85

- 6.1 Comparison of practical results. 103

- 9.1 Collection Statistics for top- k document retrieval experiments 120

- 10.1 Real datasets used, and space required to represent them. 147

- 11.1 Space usage, in bytes, of the data structures used in our index 157
- 11.2 Comparison of the space consumption of the SSA, CSA, and CST. 158

- 12.1 Description of four sample pcap files. 167
- 12.2 Comparison of the index data size. 169
- 12.3 Packet filtering queries and specifications. 172

List of Figures

1.1	Outline of the thesis	5
2.1	Concatenation \mathcal{C} of our 3-document example collection \mathcal{D}	8
2.2	Huffman tree example	10
2.3	Example of RANK and SELECT operations on a binary sequence $\mathcal{S}[1, 11]$	13
2.4	Example of a wavelet tree.	18
2.5	Huffman shaped wavelet tree of an example sequence $\mathcal{S} = 1511863875743288$	18
2.6	Example of grid representation using a wavelet tree	20
2.7	Example of a DAC representation of integers	22
2.8	Example of a K^2 -tree	22
2.9	Example of compact representation of a tree using BP, LOUDS and DFUDS	25
2.10	Example of a differentially encoded search tree	26
2.11	Example of a Cartesian tree to support RMQ over \mathcal{S}	27
2.12	Example of a treap	28
2.13	Example of a generalized suffix tree	29
2.14	Suffix array of collection \mathcal{C}	29
2.15	Matrix of all the cyclic rotations of “ATA#TAAA#TATA#\$”	31
2.16	Example of positional inverted index.	34
2.17	Example of a Document array and C -Array	35
3.1	Example of Dualsorted inverted list	49
4.1	An example of posting list representation using a treap.	56
4.2	Compact treap using BP topology.	58
4.3	Compact treap using LOUDS	59
4.4	Compact treap using HEAP.	62
4.5	Separating frequencies below $f_0 = 2$ in our example treap.	63
4.6	The left-to-right construction of an example treap.	72
5.1	Total sizes of the indexes depending on the scoring scheme.	76
5.2	Total sizes of the indexes depending on the scoring scheme.	76
5.3	Sizes of the topology components depending on the scoring scheme.	77
5.4	Construction time of the indexes depending on the scoring scheme, in minutes.	77
5.5	Ranked union times for distinct k values, in milliseconds.	79
5.6	Ranked union times grouped by number of terms per query using tf-idf.	81
5.7	Ranked union times grouped by number of terms per query using BM25.	82
5.8	Ranked union times for distinct k values as a function of the query length.	83

5.9	Ranked intersection times for distinct k values, in milliseconds.	84
5.10	Ranked intersection times grouped by number of terms per query, using tf-idf.	86
5.11	Ranked intersection times grouped by number of terms per query, using BM25.	87
5.12	Ranked intersection times for distinct k values as a function of the query length.	88
5.13	One-word query times as a function of k	90
5.14	Times for incremental versus static construction of treaps.	91
5.15	Ranked union and intersection times as a function of k	92
6.1	Example of the Hon et al. mapping	98
6.2	Example of Navarro and Nekrich mapping.	99
7.1	Wavelet tree and RMQ combination for top- k two-dimensional queries	106
7.2	Example of K^2 -treap construction from the matrix in Fig. 10.1	108
7.3	Storage of the conceptual tree in our data structures	109
8.1	H -mapping of Navarro and Nekrich's Solution.	115
9.1	Space usage decomposition as fraction of the input, for each structure employed in WRMQ.	122
9.2	Space usage decomposition for each structure employed in K^2 Treap ^H	123
9.3	Space required of grid G using single level Wavelet tree	124
9.4	Space requirements for small character alphabet collection	125
9.5	Detailed breakdown of average query time for the different indexes	126
9.6	Query times for IDX_GN on ENWIKI-BIG ^c with $m = 5$	127
9.7	Comparison of the average time per query required to retrieve the top- k results using a K^2 -treap or the wavelet tree for representing G	128
9.8	Comparison of query time of K^2 -treap and wavelet tree for representing grid G	129
9.9	Average time per query for different k values on small character alphabet collections	130
9.10	Average time per query for varying k and $m = 5$ on big character alphabet collections.	131
9.11	Average time per query in microseconds for different pattern lengths and fixed $k = 10$ value.	132
9.12	Average time per query in microseconds for different pattern lengths and fixed $k = 10$ value for ENWIKI-BIG ^c	133
9.13	Detailed breakdown of average query time for index K^2 Treap ^H on word al- phabet collections.	134
9.14	Query times on GOV2 ^w for index IDX_GN	135
9.15	Results of different grid representations on word alphabet collections	136
9.16	Query time for varying k and $m = 1$ on word alphabet collections	137
9.17	Time per query for different pattern lengths	138
10.1	Weighted matrix.	141
10.2	Binary matrix.	141
10.3	Range query.	141
10.4	Storage of the conceptual tree in our data structures.	144
10.5	Space overhead in GIS and WEB datasets.	148
10.6	Query times of range counting queries in real datasets.	149

10.7	Space-time trade-offs of the compared range counting variants.	150
11.1	Layout of our index organization for representing Web Access Logs	155
11.2	Average microseconds to perform ACCESS and USER_PATH.	159
11.3	Average microseconds to perform COUNT(P) and the MOST_FREQUENT . .	159
11.4	Average microseconds to list distinct users and retrieve top-10 most common path.	160
12.1	Extracting index data from a source packet trace file.	164
12.2	Mapping between the extracted index data and positive integers.	164
12.3	Building wavelet tree chunks.	165
12.4	Listing the final matched packets.	166
12.5	Retrieving the packets using multi-threaded file I/O.	167
12.6	Comparison of the index size.	170
12.7	Comparison of the building time of index data.	170
12.8	Comparison of the packet lookup time for different sampling sizes.	171
12.9	Comparison of packet extraction with various number of threads for parallel file I/O.	172
12.10A	performance comparison of packet extraction using RT-2/3.	173

List of Algorithms

1	Basic wavelet tree algorithms	17
2	Method RANGE_REPORT on a wavelet tree	21
3	Method K^2_RANGE on a K^2 -tree.	23
4	Method BACKWARD_SEARCH on the SSA	32
5	Method LOCATE on the SSA	32
6	Method DOCUMENT_LISTING returns the distinct documents	35
7	Method SVS	42
8	Method WAND	44
9	Method EVALUATE	45
10	Method TRY_REPORT	45
11	Method BLOCK-MAX-WAND	47
12	Method RANGE_INTERSECT on a wavelet tree	51
13	Method TREAP_INTERSECT using treaps.	65
14	Method REPORT using treaps	66
15	Method CHANGEV using treaps.	66
16	Method CHANGED using treaps.	66
17	Method TREAP_UNION using treaps.	68
18	Method CHANGED_UNION using treaps.	69
19	Method WT_GREEDY returns the top- k most frequent symbols	101
20	Top- k algorithm using WRMQ index	113
21	Top- k algorithm using H -mapping index	116