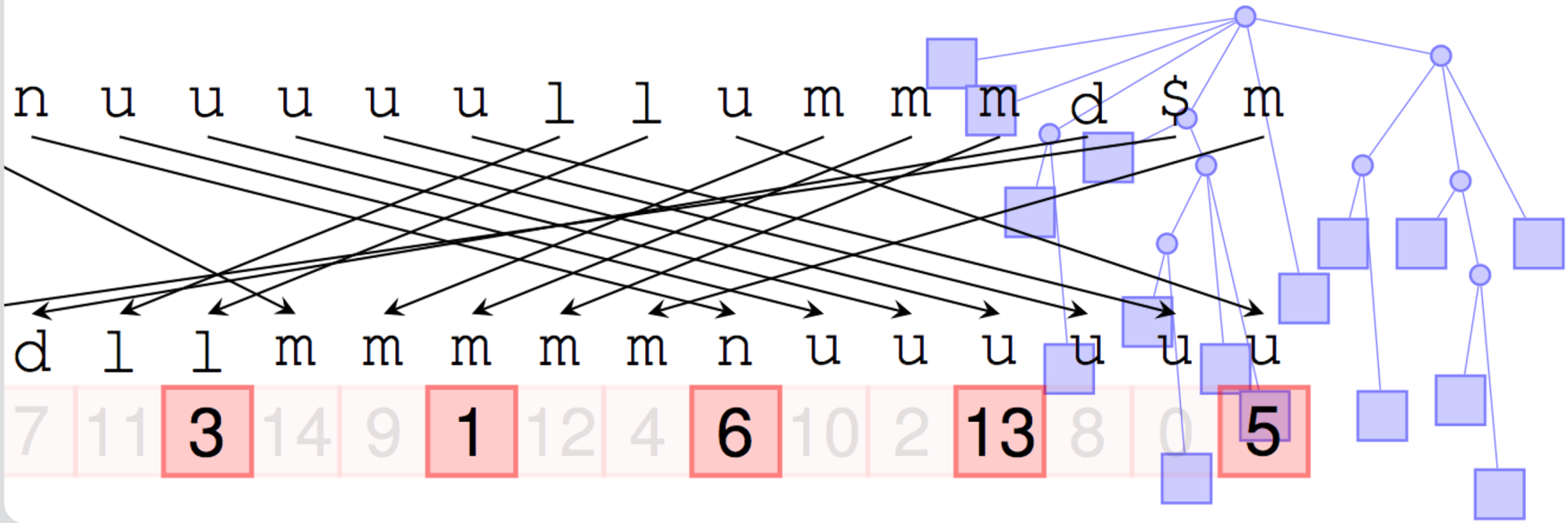
Karlsruhe Institute of Technology

Preview: Text Indexing

Simon Gog – *gog@ira.uka.de*

Institute of Theoretical Informatics - Algorithmics



Text Indexing

Motivation

Problems

Given a text $\mathcal{T}$ of length n over an alphabet Σ of size σ and a pattern $\mathcal{P}$ of length m. Typical questions we want to answer efficiently:

- Does $\mathcal{P}$ occur in $\mathcal{T}$? (Existence query)
- How often does $\mathcal{P}$ occur in $\mathcal{T}$? (Count query)
- Where does $\mathcal{P}$ occur in $\mathcal{T}$? (Locate query)

Two scenarios:

- Scan the whole text $\mathcal{T}$ for each query (time complexity $\mathcal{O}(n+m)$)).
- Build an index for $\mathcal{T}$ once. Use index to answer the query (time complexity not linear dependent on n or even independent!).

Text Indexing

Motivation

Index solution

Suffix Array (SA) was already explained in this lecture.

- Existence/count time: $\mathcal{O}(m \log n)$
- Locate time: $\mathcal{O}(m \log n + occ)$, where occ is the number of occurrences.

Drawbacks

- SA is larger than $\mathcal{T}$: $n \log n$ compared to $n \log \sigma$. For $\sigma = 256$ and $n = 2^{32}$ we get factor 4.
- SA requires $\mathcal{T}$ to answer queries.

Other classical indexes: Suffix trees (ST), String B-Tree (SBT), and uncompressed positional Inverted Indexes (PII).

Text Indexing

Suffix Array

i	$SA[i]$	$\mathcal{T}[SA[i]..n-1]$
0	18	$
1	17	a$
2	10	abarbara$
3	7	abrabarbara$
4	0	abracadabrabarbara$
5	3	acadabrabarbara$
6	5	adabrabarbara$
7	15	ara$
8	12	arbara$
9	14	bara$
10	11	barbara$
11	8	brabarbara$
12	1	bracadabrabarbara$
13	4	cadabrabarbara$
14	6	dabrabarbara$
15	16	ra$
16	9	rabarbara$
17	2	racadabrabarbara$
18	13	rbara$

- $\mathcal{T}[SA[i]..n-1] < \mathcal{T}[SA[i+1]..n-1]$
- $SA[i]$ contains the starting position of the ith lex. smallest suffix of $\mathcal{T}$.

- Matching algorithm: binary search (forward, left-to-right)

Compressed Text Indexes

The FM-Index

Ferragina and Manzini [2000]

- Index based on Burrows-Wheeler-Transform (BWT)
- Matching algorithm works backwards (right-to-left)
- Existence and count queries in time $\mathcal{O}(m \log \sigma)$

BWT

- $BWT[i] = \mathcal{T}[SA[i] - 1 \bmod n]$
- uncompressed size: $n \log \sigma$ bits
- compressed size: $nH_k(\mathcal{T})$ bits (+information for contexts of length k)

Backward Search

i	$SA[i]$	BWT	$\mathcal{T}[SA[i]..n-1]$
0	18	a	$
1	17	r	a$
2	10	r	abarbara$
3	7	d	abrabarbara$
4	0	$	abracadabrabarbara$
5	3	r	acadabrabarbara$
6	5	c	adabrabarbara$
7	15	b	ara$
8	12	b	arbara$
9	14	r	bara$
10	11	a	barbara$
11	8	a	brabarbara$
12	1	a	bracadabrabarbara$
13	4	a	cadabrabarbara$
14	6	a	dabrabarbara$
15	16	a	ra$
16	9	b	rabarbara$
17	2	b	racadabrabarbara$
18	13	a	rbara$

- $BWT[i] = \mathcal{T}[SA[i]-1]$, for $SA[i] > 0$
- $BWT[i] = \mathcal{T}[n-1]$, for $SA[i] = 0$
- I.e. $BWT[i]$ is the character preceding suffix $SA[i]$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

Array C contains for each $c \in \Sigma$ the position of the first suffix in SA which starts with c:

$	a	b	c	d	r	r+1
0	1	9	13	14	15	19

- Operation $rank(i, X, BWT)$ returns how often character $X \in \Sigma$ occurs in the prefix $BWT[0..i-1]$.
- Example: search for $\mathcal{P} = bar$.

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = C[r] + rank(sp_0, r, BWT)$
  $ep_1 = C[r] + rank(ep_0 + 1, r, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15 + rank(0, r, BWT)$
  $ep_1 = 15 + rank(19, r, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15 + 0$
  $ep_1 = 15 + rank(19, r, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Initial interval: $[sp_0, ep_0] = [0..n-1]$
- Determine interval for r:
  $sp_1 = 15+0 = 15$
  $ep_1 = 15+4-1 = 18$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = C[a] + rank(sp_1, a, BWT)$
  $ep_2 = C[a] + rank(ep_1 + 1, a, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + rank(15, a, BWT)$
  $ep_2 = 1 + rank(ep_1, a, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + rank(15, a, BWT)$
  $ep_2 = 1 + rank(ep_1, a, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1 + 6$
  $ep_2 = 1 + rank(19, a, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_1, ep_1] = [15..18]$
- Determine interval for *ar*:
  $sp_2 = 1+6 = 7$
  $ep_2 = 1+8-1 = 8$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = C[b] + rank(sp_2, b, BWT)$
  $ep_3 = C[b] + rank(ep_2 + 1, b, BWT) - 1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C

$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9 + rank(7, b, BWT)$
  $ep_3 = 9 + rank(ep_1, b, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9 + rank(7, b, BWT)$
  $ep_3 = 9 + rank(ep_1, b, BWT)$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9+0$
  $ep_3 = 9+rank(9, b, BWT)-1$

Backward Search

i	BWT	$\mathcal{T}[SA[i]..n-1]$
0	a	$
1	r	a$
2	r	abarbara$
3	d	abrabarbara$
4	$	abracadabrabarbara$
5	r	acadabrabarbara$
6	c	adabrabarbara$
7	b	ara$
8	b	arbara$
9	r	bara$
10	a	barbara$
11	a	brabarbara$
12	a	bracadabrabarbara$
13	a	cadabrabarbara$
14	a	dabrabarbara$
15	a	ra$
16	b	rabarbara$
17	b	racadabrabarbara$
18	a	rbara$

C					
$	a	b	c	d	r
0	1	9	13	14	15

- Search backwards for *bar*.
- Interval: $[sp_2, ep_2] = [7..8]$
- Determine interval for *bar*:
  $sp_3 = 9+0 = 9$
  $ep_3 = 9+2-1 = 10$

Backward Search

Summary

- Only C and a data structure R supporting the *rank* operation on *BWT* are required for existence and count queries.
- Space: $\sigma \log n$ bits for C + space for R
- Time: $\mathcal{O}(m \cdot t_{rank})$, where t_{rank} is time for one rank operation. Independent from n?
- Next: How to implement *rank*?

Rank operation

- Constant time and $o(n)$ extra space solution on bitvectors (Jacobson [1989])
- Solution on general sequences: Wavelet Tree (Grossi et al. [2003])

Backward Search

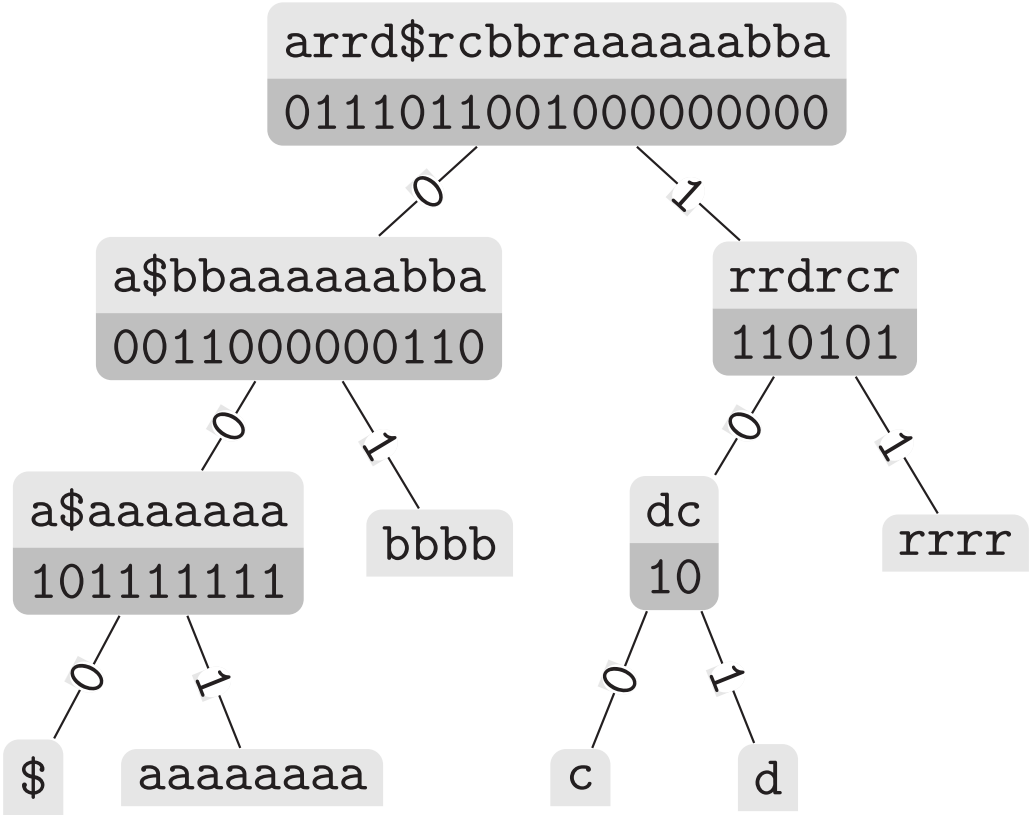
Summary

- Only C and a data structure R supporting the *rank* operation on *BWT* are required for existence and count queries.
- Space: $\sigma \log n$ bits for C + space for R
- Time: $\mathcal{O}(m \cdot t_{rank})$, where t_{rank} is time for one rank operation. Independent from n? If t_{rank} is independent from n
- Next: How to implement *rank*?

Rank operation

- Constant time and $o(n)$ extra space solution on bitvectors (Jacobson [1989])
- Solution on general sequences: Wavelet Tree (Grossi et al. [2003])

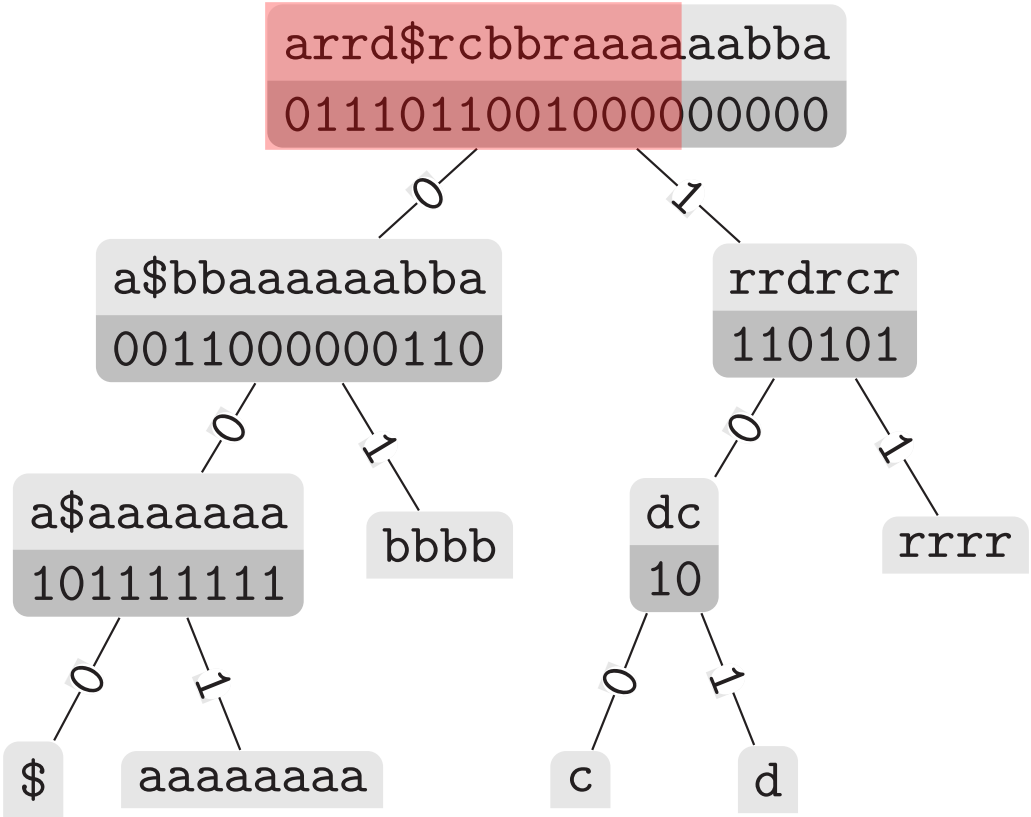
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

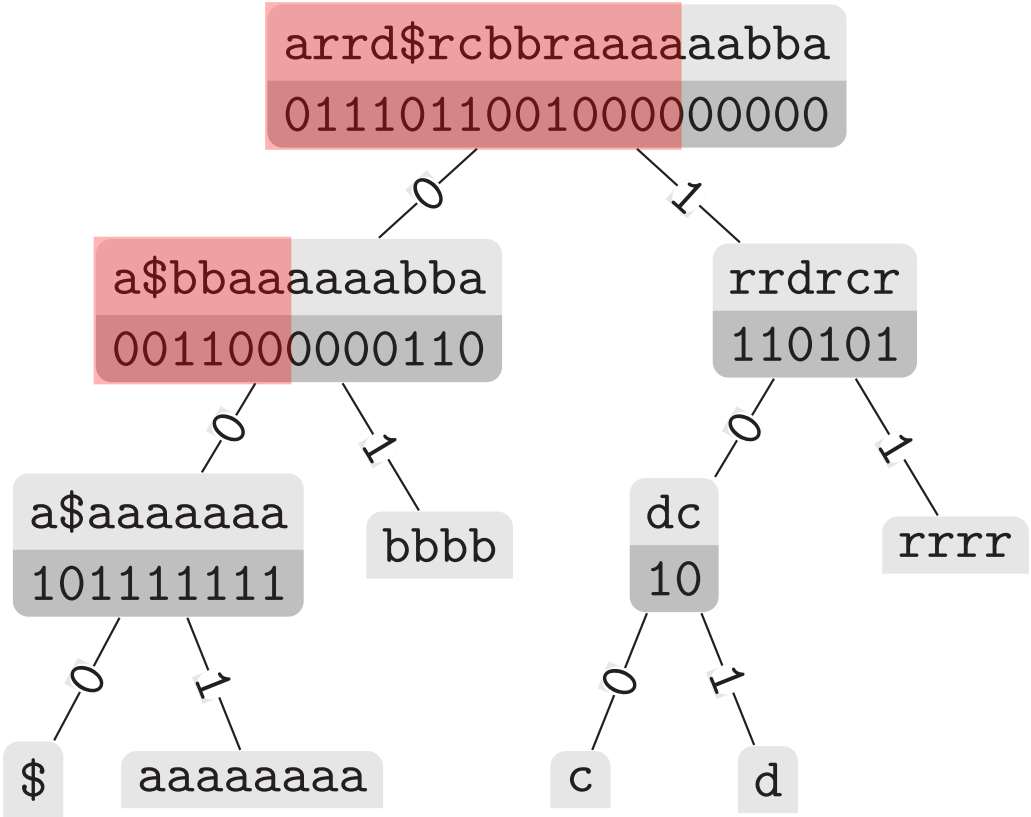
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

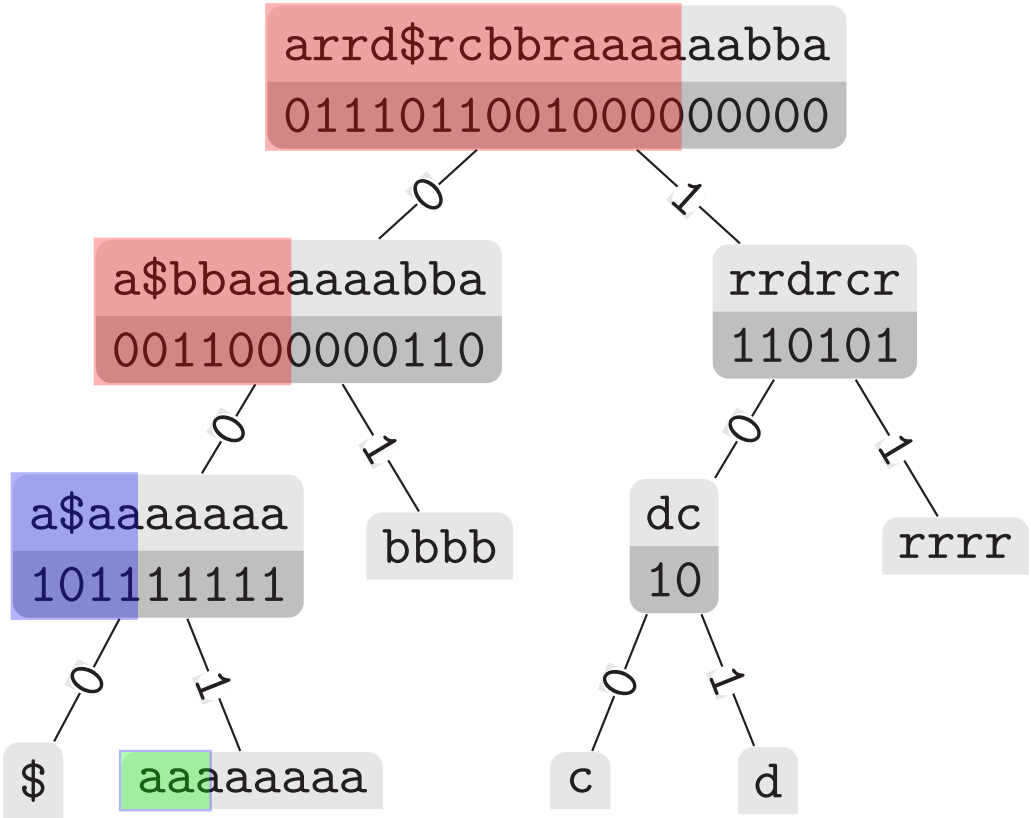
Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

Wavelet Tree Example: Calculate Rank



a = 001

$$rank(11, a, WT) = rank(rank(rank(11, 0, b_\epsilon) = 5, 0, b_0) = 3, 1, b_{00}) = 2$$

Compressed Text Indexing

meets Algorithm Engineering

Karlsruhe Institute of Technology

State-of-the-art

Recent FM-Indexes are as small as the output of state-of-the-art compressors (like gzip,xz) while matching takes microseconds per character.

This is the result of theoretical and practical improvements:

- Shape of the WT (balanced, Huffman, Hu-Tucker,...)
- Bitvector representation (uncompressed/compressed)
- Hardware (popcount instruction, page size)
- Different sampling strategies for SA values (for locate queries)
- ...

Compressed Text Indexing

Combining a H_0-compressed bitvector with a Huffman shaped wavelet tree results in $H_k(T)$ bits of space.

	WT-HUFF		WT-HUFFcompr	
200 MB test instance	Time (μs)	Space (%)	Time (μs)	Space (%)
DBLP.XML	1.09	70	2.04	17
DNA	0.40	29	1.66	24
ENGLISH	0.97	61	2.64	27
PROTEINS	0.87	56	3.22	48
SOURCES	1.20	73	2.82	26

Compressed Text Indexing

Our Toolbox for Compact/Succinct Data Structures

Karlsruhe Institute of Technology

Succinct Data Structure Library (SDSL)

- A C++ template library for compact/succinct structures
- Parametrizable structures
  - Bitvectors
  - Compressed Integer Vectors
  - Rank/Select Structures
  - Wavelet Trees/Wavelet Matrices
  - Compressed Suffix Arrays/Trees
  - Search Engines

Available at https://github.com/simongog/sdsl-lite.

Lecture Text Indexing

Content

Theory:

- Classical indexes (Suffix Arrays/Suffix Trees/Inverted Indexes)
- Building blocks for compact/succinct structures
  - Compressed Bitvector
  - Rank Structures
  - Select Structures
  - Range-Min-Max-Tree
- Compressed indexes
  - FM-Indexes/Compressed Suffix Arrays
  - Versions for highly-repetitive text
  - Compressed Suffix Trees
- Search Engines

Practice

- Use SDSL to implement and analyze structures.
- Design a code search engine.

Bibliography

Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science, (FOCS 2000)*, pages 390–398, 2000.

Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 841–850, 2003.

Guy Jacobson. Space-efficient static trees and graphs. In *FOCS*, pages 549–554, 1989.

H_k of selected *Pizza&Chili* 200MB test cases

Karlsruhe Institute of Technology

	$H_k(T)$ contexts/$\lvert T \rvert$ in percent							
k	DBLP.XML		DNA		ENGLISH		PROTEINS	
0	5.26	0.0	1.97	0.0	4.53	0.0	4.20	0.0
1	3.48	0.0	1.93	0.0	3.62	0.0	4.18	0.0
2	2.17	0.0	1.92	0.0	2.95	0.0	4.16	0.0
3	1.43	0.1	1.92	0.0	2.42	0.0	4.07	0.0
4	1.05	0.4	1.91	0.0	2.06	0.3	3.83	0.1
5	0.82	1.3	1.90	0.0	1.84	1.0	3.16	1.7
6	0.70	2.7	1.88	0.0	1.67	2.7	1.50	17.4
7	0.63	4.3	1.86	0.0	1.51	5.5	0.34	45.0
8	0.57	6.0	1.83	0.0	1.34	9.9	0.10	53.8
9	0.54	7.7	1.80	0.1	1.15	15.8	0.07	55.9
10	0.51	9.6	1.76	0.5	0.96	22.9	0.06	57.0