COMP9319 Web Data Compression and Search

Search on Suffix Array,
FM Index,
Backward Search,
Compressed BWT

1

Suffix Array

- We loose some of the functionality but we save space.

Let s = abab

Sort the suffixes lexicographically:
ab, abab, b, bab

The suffix array gives the indices of the suffixes in sorted order

3	1	4	2

Suffix Array

- We loose some of the functionality but we save space.

Let s = abab

Sort the suffixes lexicographically:
ab, abab, b, bab

The suffix array gives the indices of the suffixes in sorted order

2	0	3	1

Note: If 0-based index: some papers assume 1-based, some are 0-based.

Example

Let S = mississippi

Let P = issi

L	11	i
	8	ippi
	5	issippi
	2	ississippi
	1	mississippi
M	10	pi
	9	ppi
	7	sippi
	4	sisippi
	6	ssippi
R	3	ssissippi

Example

i.e., To find every suffix that begins with issi.
How???

Let S = mississippi

Let P = issi

L	11	i
	8	ippi
	5	issippi
	2	ississippi
	1	mississippi
M	10	pi
	9	ppi
	7	sippi
	4	sisippi
	6	ssippi
R	3	ssissippi

Example

Two binary searches !!
So total O(m log n)

Let S = mississippi
n

Let P = issi
m

L	11	i
	8	ippi
	5	issippi
	2	ississippi
	1	mississippi
M	10	pi
	9	ppi
	7	sippi
	4	sisippi
	6	ssippi
R	3	ssissippi

Example

Let S = mississippi

Let P = issi

M = (L+R) / 2
if P > S[M]:
  L = M+1
else:
  R = M

	SA	Suffix
L →	11	i
	8	ippi
m →	5	issippi
	2	ississippi
	1	mississippi
M →	10	pi
	9	ppi
	7	sippi
	4	sisippi
	6	ssippi
R →	3	ssissippi

Example

Let S = mississippi

Let P = issi

M = (L+R) / 2
if P < S[M]:
  R = M
else:
  L = M+1

	SA	Suffix	
L →	11	i	
	8	ippi	
3 →	5	issippi	← 1st
4 →	2	ississippi	← 2nd
	1	mississippi	
M →	10	pi	
	9	ppi	
	7	sippi	
	4	sisippi	
	6	ssippi	
R →	3	ssissippi	

Backward Search
FM-Index
(Full-text index in Minute space)

Paper by
Ferragina & Manzini

Modified from slides by Yuval Rikover

Motivation

- Combine: Text Compression + Indexing (**discard original text**).
- Count and locate P with length m by looking at only a small portion of the compressed text.
- Do it efficiently:
  - Time: O(m)

10

How does it work?

- Exploit the relationship between the *Burrows-Wheeler Transform* and the Suffix Array data structure.
- Compressed suffix array that encapsulates both the ***compressed text*** and the ***full-text indexing information.***
- Supports two basic operations:
  - **Count** – return number of occurrences of P in T.
  - **Locate** – find all positions of P in T.

11

Burrows-Wheeler Transform

- Every column is a permutation of T.
- Given row i, char L[i] precedes F[i] in original T.
- Consecutive char's in L are adjacent to similar strings in T.
- Therefore – L usually contains long runs of identical char's.

F		L
#	mississipp	i
i	#mississip	p
i	ppi#missis	s
i	ssippi#mis	s
i	ssissippi#	m
m	ississippi	#
p	i#mississi	p
p	pi#mississ	i
s	ippi#missi	s
s	issippi#mi	s
s	sippi#miss	i
s	sissippi#m	i

12

Burrows-Wheeler Transform

Reminder: Recovering T from L

1. Find F by sorting L
2. Last char of T? #
3. Find # in L
4. L[i] precedes F[i] in T. Therefore we get
   i#
5. How do we choose the correct i in F?
   - The i's are in the same order in L and F
   - As are the rest of the char's
6. p precedes i: pi#
7. And so on….

F	L
#	i
i	p
i	s
i	s
i	m
m	#
p	p
p	i
s	s
s	s
s	i
s	i

13

Next: Count P in T

- **Backward-search algorithm**
- Uses only L (output of BWT)
- Relies on 2 structures:
  - C[1,…,|Σ|] : C[c] contains the total number of text chars in T which are alphabetically smaller then c (including repetitions of chars)
  - Occ(c,q): number of occurrences of char c in prefix L[1,q]

Example

- C[ ] for T = mississippi#

1	5	6	8
i	m	p	s

- occ(s, 5) = 2
- occ(s,12) = 4

Occ ≡ Rank

F		L	
#	mississipp	i	1
i	#mississip	p	2
i	ppi#missis	s	3
i	ssippi#mis	s	4
i	ssissippi#	m	5
m	ississippi	#	6
p	i#mississi	p	7
p	pi#mississ	i	8
s	ippi#missi	s	9
s	issippi#mi	s	10
s	sippi#miss	i	11
s	sissippi#m	i	12

14

Backward-Search

- Works in p iterations, from p down to 1

P = msi $\xrightarrow{i=3,\ c=\text{'i'}}$ msi $\xrightarrow{i=2,\ c=\text{'s'}}$ msi $\xrightarrow{i=1,\ c=\text{'m'}}$ *msi*

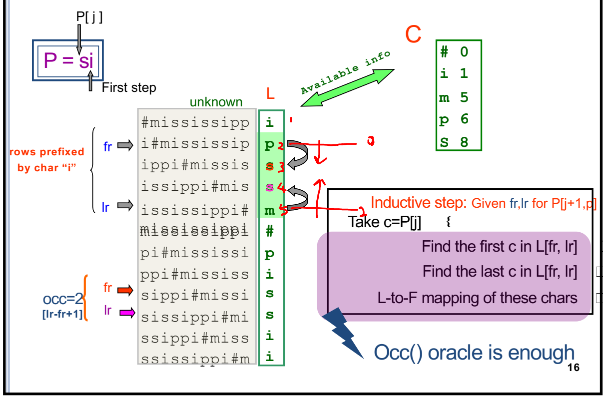
- Remember that the BWT matrix rows = sorted suffixes of T
  - All suffixes prefixed by pattern P, occupy a continuous set of rows
  - This set of rows has starting position ***First***
  - and ending position ***Last***
  - So, (Last – First +1) gives total pattern occurrences
- At the end of the i-th phase, ***First*** points to the first row prefixed by P[i,p], and ***Last*** points to the last row prefiex by P[i,p].

Algorithm backward_search($P[1,p]$)

(1) $i \leftarrow p$, $c \leftarrow P[p]$, First $\leftarrow C[c]+1$, Last $\leftarrow C[c+1]$;
(2) **while** ((First $\le$ Last) **and** ($i \ge 2$)) **do**
(3) $c \leftarrow P[i-1]$;
(4) First $\leftarrow C[c]$ + Occ(c, First -1) $+1$;
(5) Last $\leftarrow C[c]$ + Occ(c, Last);
(6) $i \leftarrow i-1$;
(7) **if** (Last < First) **then return** "no rows prefixed by $P[1,p]$" **else return** ⟨First, Last⟩.

15

Substring search in T (Count the pattern occurrences)



16

Backward-Search Example

- P = pssi
  - i = 3
  - c = 's'
  - First = C['s'] + Occ('s',1) +1 = 8+0+1 = 9
  - Last = C['s'] + Occ('s',5) = 8+2 = 10
  - (Last – First + 1) = 2

First → row 2, Last → row 5

F		L	
#	mississipp	i	1
i	#mississip	p	2
i	ppi#missis	s	3
i	ssippi#mis	s	4
i	ssissippi#	m	5
m	ississippi	#	6
p	i#mississi	p	7
p	pi#mississ	i	8
s	ippi#missi	s	9
s	issippi#mi	s	10
s	sippi#miss	i	11
s	sissippi#m	i	12

C[ ] =

1	5	6	8
i	m	p	s

Algorithm backward_search($P[1,p]$)

(1) $i \leftarrow p$, $c \leftarrow P[p]$, First $\leftarrow C[c]+1$, Last $\leftarrow C[c+1]$;
(2) **while** ((First $\le$ Last) **and** ($i \ge 2$)) **do**
(3) $c \leftarrow P[i-1]$;
(4) First $\leftarrow C[c]$ + Occ(c, First -1) $+1$;
(5) Last $\leftarrow C[c]$ + Occ(c, Last);
(6) $i \leftarrow i-1$;
(7) **if** (Last < First) **then return** "no rows prefixed by $P[1,p]$" **else return** ⟨First, Last⟩.

17

Backward-Search Example

- P = pssi
  - i = 2
  - c = 's'
  - First = C['s'] + Occ('s',8) +1 = 8+2+1 = 11
  - Last = C['s'] + Occ('s',10) = 8+4 = 12
  - (Last – First + 1) = 2

First → row 9, Last → row 10

F		L	
#	mississipp	i	1
i	#mississip	p	2
i	ppi#missis	s	3
i	ssippi#mis	s	4
i	ssissippi#	m	5
m	ississippi	#	6
p	i#mississi	p	7
p	pi#mississ	i	8
s	ippi#missi	s	9
s	issippi#mi	s	10
s	sippi#miss	i	11
s	sissippi#m	i	12

C[ ] =

1	5	6	8
i	m	p	s

Algorithm backward_search($P[1,p]$)

(1) $i \leftarrow p$, $c \leftarrow P[p]$, First $\leftarrow C[c]+1$, Last $\leftarrow C[c+1]$;
(2) **while** ((First $\le$ Last) **and** ($i \ge 2$)) **do**
(3) $c \leftarrow P[i-1]$;
(4) First $\leftarrow C[c]$ + Occ(c, First -1) $+1$;
(5) Last $\leftarrow C[c]$ + Occ(c, Last);
(6) $i \leftarrow i-1$;
(7) **if** (Last < First) **then return** "no rows prefixed by $P[1,p]$" **else return** ⟨First, Last⟩.

18

BACKWARD-SEARCH EXAMPLE

- P = pssi
  - i = 1
  - c = 'p'
  - First = C['p'] + Occ('p',10) +1 = 6+2+1 = 9
  - Last = C['p'] + Occ('p',12) = 6+2 = 8
  - (Last – First + 1) = 0

F		L	
#	mississipp	i	1
i	#mississip	p	2
i	ppi#missis	s	3
i	ssippi#mis	s	4
i	ssissippi#	m	5
m	ississippi	#	6
p	i#mississi	p	7
p	pi#mississ	i	8
s	ippi#missi	s	9
s	issippi#mi	s	10
s	sippi#miss	i	11
s	sissippi#m	i	12

First
Last

C[ ] =	i	m	p	s
	1	5	6	8

```
Algorithm backward_search(P[1,p])
i ← p, c ← P[p], First ← C[c] + 1, Last ← C[c+1];
while ((First ≤ Last) and (i ≥ 2)) do
    c ← P[i − 1];
    First ← C[c] + Occ(c, First − 1) + 1;
    Last ← C[c] + Occ(c, Last);
    i ← i − 1;
if (Last < First) then return "no rows prefixed by P[1,p]" else return ⟨First, Last⟩
```

19

COMPRESSED SUFFIX ARRAY / BWT

SUCCINCT SUFFIX ARRAYS
BASED ON RUN-LENGTH ENCODING *

VELI MÄKINEN†
Dept. of Computer Science, University of Helsinki
Gustaf Hällströmin katu 2b, 00014 University of Helsinki, Finland
vmakinen@cs.helsinki.fi

GONZALO NAVARRO‡
Dept. of Computer Science, University of Chile
Blanco Encalada 2120, Santiago, Chile
gnavarro@dcc.uchile.cl

Slides modified from the original Makinen & Navarro's

SIMPLE FM-INDEX

- Construct the *Burrows-Wheeler-transformed* text bwt(T) [BW94].
- From bwt(T) it is possible to construct the suffix array sa(T) of T in linear time.
- Instead of constructing the whole sa(T), one can add small data structures besides bwt(T) to simulate a search from sa(T).

BURROWS-WHEELER TRANSFORMATION

- Construct a matrix M that contains as rows all rotations of T.
- Sort the rows in the lexicographic order.
- Let L be the last column and F be the first column.
- bwt(T)=L associated with the row number of T in the sorted M.

EXAMPLE

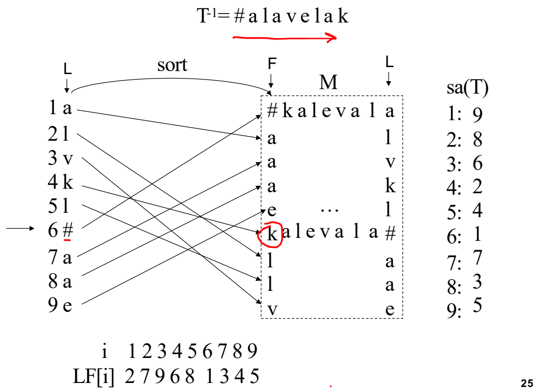
```
pos  123456789
T  = kalevala#
```

```
sa  F  M  L
1:9 #kalevala
2:8 a#kaleval
3:6 ala#kalev
4:2 alevala#k
5:4 evala#kal
6:1 kalevala#
7:7 la#kaleva
8:3 levala#ka
9:5 vala#kale
```

L = alvkl#aae, row 6

==>

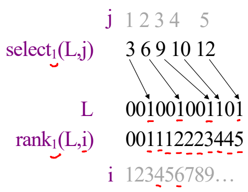
Exercise: Given L and the row number, we know how to compute T. What about sa(T)?

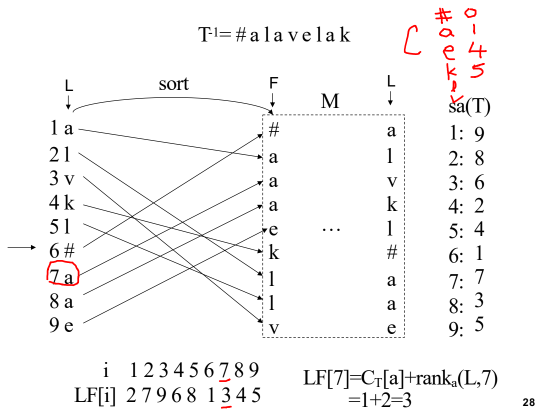


IMPLICIT LF[i]

- Ferragina and Manzini (2000) noticed the following connection:
- $LF[i]=C_T[L[i]]+rank_{L[i]}(L,i)$
- Here
  $C_T[c]$: amount of letters 0,1,...,c-1 in L=bwt(T)
  $rank_c(L,i)$: amount of letters c in the prefix L[1,i]

RANK/SELECT





RECALL: BACKWARD SEARCH ON BWT(T)

- **Observation**: If [i,j] is the range of rows of M that start with string X, then the range [i',j'] containing cX can be computed as

Fr $i' := C_T[c]+rank_c(L,i-1)+1,$
Lr $j' := C_T[c]+rank_c(L,j).$

BACKWARD SEARCH ON BWT(T)...

- Array $C_T[1,\sigma]$ takes $O(\sigma \log |T|)$ bits.
- $L=Bwt(T)$ takes $O(|T| \log \sigma)$ bits.
- Assuming $rank_c(L,i)$ can be computed in constant time for each (c,i), the algorithm takes $O(|P|)$ time to count the occurrences of P in T.

Run-Length FM-index

- We make the following changes to the previous FM-index variant:
  - L=Bwt(T) is replaced by a sequence S[1,n'] and two bit-vectors B[1,|T|] and B'[1,|T|],
  - Cumulative array $C_T[1,c]$ is replaced by $C_S[1,c]$,
  - some formulas are changed.

Run-Length FM-index...

L	B	S	L	F	B'
c	1	c	c	a	1
c	0	a	c	a	0
c	0	g	c	a	1
a	1	a	a	c	1
a	0	t	a	c	0
g	1		g	c	0
g	0		g	g	1
a	1		a	g	0
t	1		t	t	1
t	0		t	t	0

F_S: a a c g t

a a a c c c g g t t

Changes to formulas

- Recall that we need to compute $C_T[c]+rank_c(L,i)$ in the backward search.
- **Theorem:** $C[c]+rank_c(L,i)$ is equivalent to $select_1(B',C_S[c]+1+rank_c(S,rank_1(B,i)))-1$, when $L[i] \neq c$ (e.g., when backward search), and otherwise (e.g., when reverse, sometimes backward search too) to $select_1(B',C_S[c]+rank_c(S,rank_1(B,i)))+ i-select_1(B,rank_1(B,i))$.

Example, reverse L

C	
a	0
c	2
g	3
t	4

L	F	B	S	B'
c	a	1	c	1
c	a	0	a	0
c	a	0	g	1
a	c	1	a	1
a	c	0	t	0
g	c	1		0
g	g	0		1
a	g	1		0
t	t	1		1
t	t	0		0

$LF[8]= select_1(B',C_S[a]+rank_a(S,rank_1(B,8)))+ 8-select_1(B,rank_1(B,8))$
$= select_1(B',0+rank_a(S,4))+8-select_1(B,4)$
$= select_1(B',0+2)+8-8$
$= 3$

- For more details, read the paper

Exercise

- ipsm$pisi
- 111011111010

What is B'

i	B	S
1	1	i
2	1	p
3	1	s
4	0	m
5	1	$
6	1	p
7	1	i
8	1	s
9	1	i
10	0	
11	1	
12	0	

Usually B' is given to save computations

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

Reverse BWT from row 6

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

Reverse BWT

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

$S[rank_1(B, 6)] = \$$

Reverse BWT

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

$S[rank_1(B, 6)] = \$$

LF[6]

$= select_1(B', C_S[\$] + rank_\$(S, rank_1(B, 6))) + 6 - select_1(B, rank_1(B, 6)))$

$= select_1(B', 0 + rank_\$(S, 5)) + 6 - select_1(B\ 5)$

$= 1 + 6 - 6 = 1$

Reverse BWT

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

$S[rank_1(B, 1)] = i$

LF[1]

$= select_1(B', C_S[i] + rank_i(S, rank_1(B, 1))) + 1 - select_1(B, rank_1(B, 1)))$

$= select_1(B', 1 + rank_i(S, 1)) + 1 - select_1(B, 1)$

$= 2 + 1 - 1 = 2$

REVERSE BWT

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

$S[\text{rank}_1(B, 1)] = i$

LF[1]

$= \text{select}_1(B', C_S[i] + \text{rank}_i(S, \text{rank}_1(B, 1))) + 1 - \text{select}_1(B, \text{rank}_1(B, 1)))$

$= \text{select}_1(B', 1 + \text{rank}_i(S, 1)) + 1 - \text{select}_1(B\ 1)$

$= 2 + 1 - 1 = 2$

You can also construct the SA in this way:

12, 11, ….

12,11,8,5,2,1,10,9,7,4,6,3

BACKWARD SEARCH

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

Suppose search for si:

c = i, First = 2, Last = 5

c = s

First = C[c] + Occ(c, First – 1) + 1

Last = C[c] + Occ(c, Last)

BACKWARD SEARCH

i	B	S	B'
1	1	i	1
2	1	p	1
3	1	s	1
4	0	m	1
5	1	$	0
6	1	p	1
7	1	i	1
8	1	s	1
9	1	i	1
10	0		0
11	1		1
12	0		0

c = i, First = 2, Last = 5

c = s

$\text{First} = \text{select}_1(B', C_S[s]+1+\text{rank}_s(S, \text{rank}_1(B,2-1))) -1 + 1$

$=\text{select}_1(B',7+1+\text{rank}_s(S,1))$

$=\text{select}_1(B', 8) = 9$

$\text{Last} = \text{select}_1(B', C_S[s]+1+\text{rank}_s(S, \text{rank}_1(B,5))) -1$

$=\text{select}_1(B',7+1+\text{rank}_s(S,4)) - 1$

$=\text{select}_1(B', 9) -1 = 11 - 1 = 10$