The Burrows Wheeler transform and the FM index

Burrows –Wheeler (bzip2)

Currently best algorithm for text compression

High level:

- Apply the Borrows-Wheeler transform
- Use move-to-front to translate the sorted characters to small integers
- Use Huffman coding

שלב I : יצירת מטריצה M שבנויה מכל ההזזות הציקליות של S:

$$S = abraca$$

$M =$

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

שלב II : מיון השורות בסדר לקסיקוגרפי:

L is the Burrows Wheeler Transform

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Claim: Every column contains all chars.

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

You can obtain F from L by sorting

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

The "a's" are in the same order in L and in F,

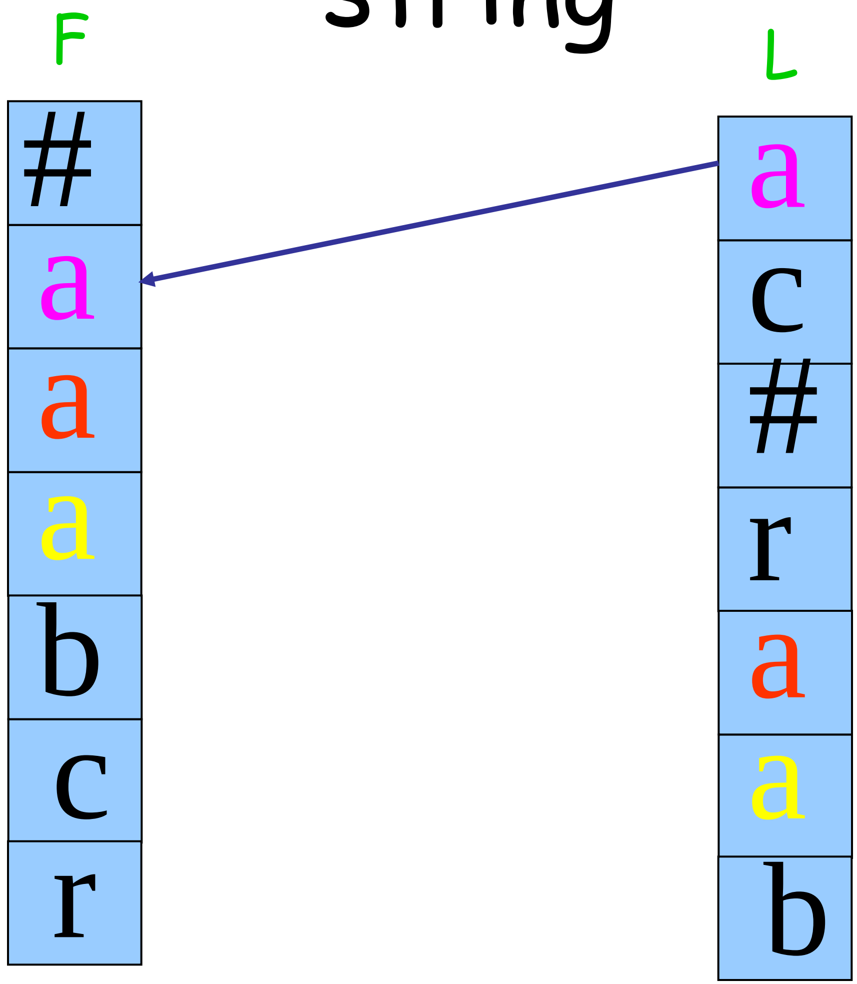
Similarly for every other char.

From L you can reconstruct the string (backwards)

F	L
#	a
a	c
a	#
a	r
b	a
c	a
r	b

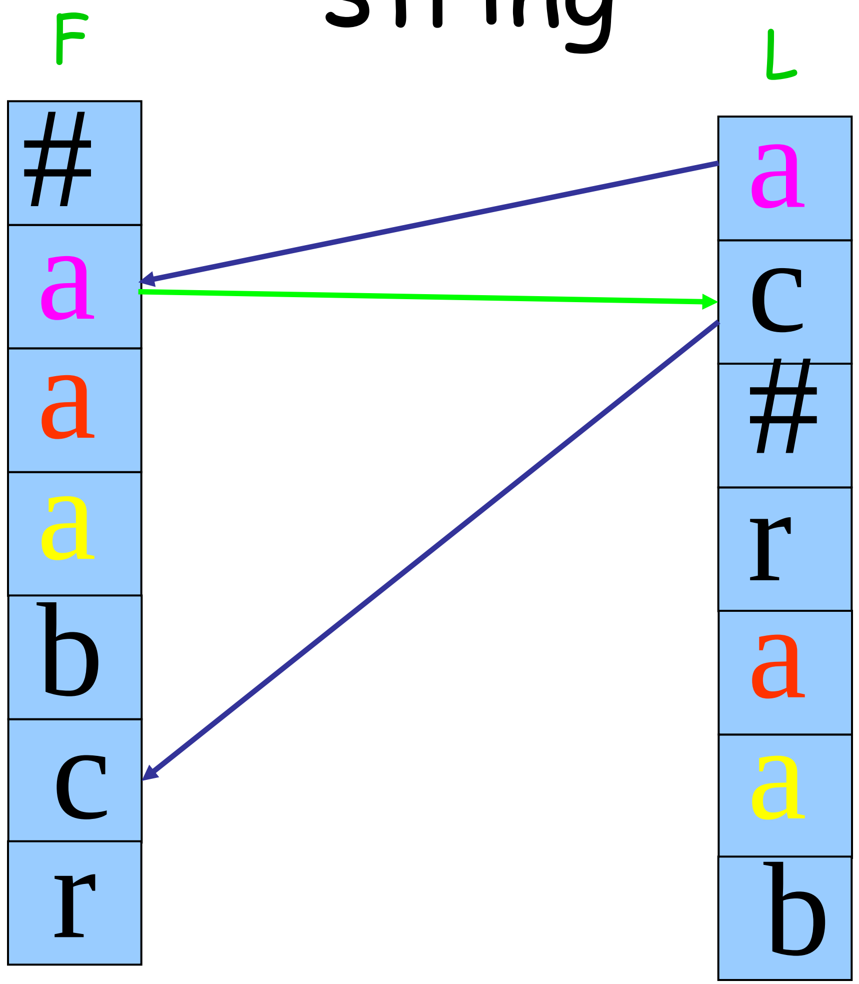
What is the last char of S ?

From L you can reconstruct the string



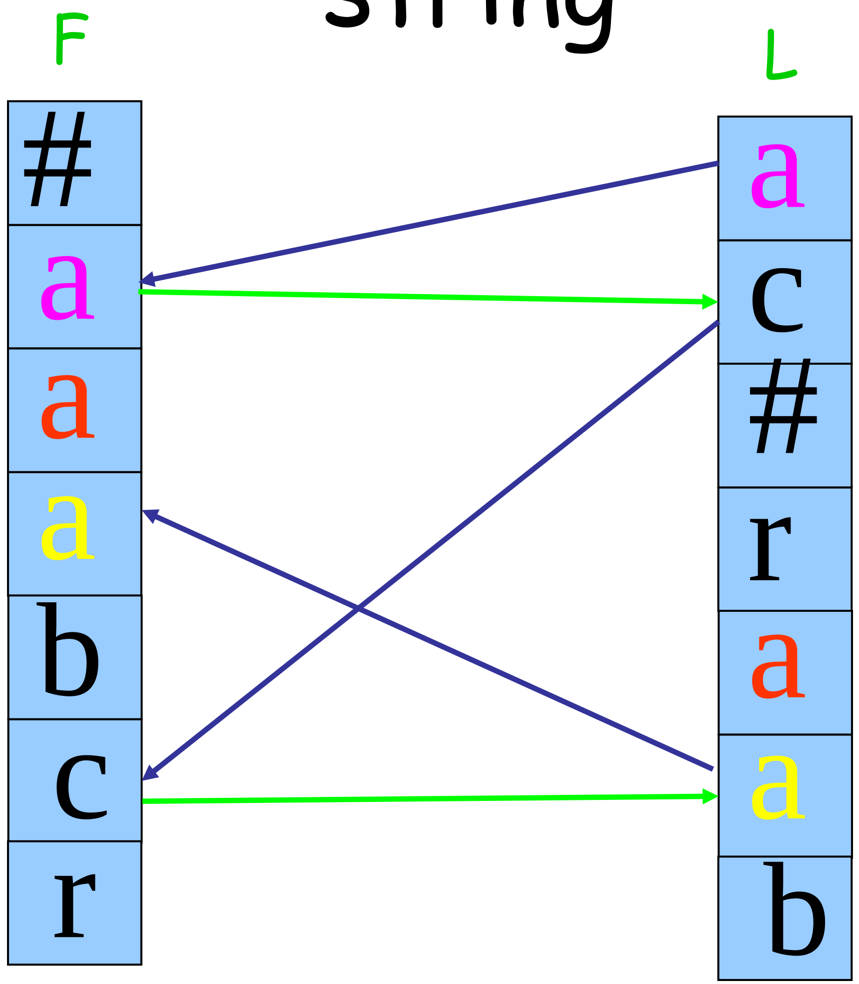
What is the last char of S ? a

From L you can reconstruct the string



ca

From L you can reconstruct the string



aca

Anslysis so far

Sorting is equivalent to computing the suffix array.

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Decoding is also linear time

Two useful arrays

- We do not really need them for decoding but they will be useful later
- $C[\cdot]$ denotes an array of length $|\Sigma|$ such that $C[c]$ contains the total number of text characters which are alphabetically smaller than c.
- $Occ(c,q)$ denotes the number of occurrences of character c in the prefix $L[1,q]$ of the transformed text L.

Two useful arrays

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

C

#	0
i	1
m	5
p	6
s	8

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

$\leftarrow$ occ(s,4) = 2

The LF mapping

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

C

#	0
i	1
m	5
p	6
s	8

$LF(i) = C[L[i]] + Occ(L[i], i)$

$LF(10) = C[s] + Occ(s, 10) = 12$

$L(10) = F(12) = s$

The LF mapping

		L	
	#mississipp	i	
	i#mississip	p	
	ippi#missis	s	
	issippi#mis	s	
	ississippi#	m	
	mississippi	#	
	pi#mississi	p	
	ppi#mississ	i	
	sippi#missi	s	
j	sissippi#mi	s	T[k]
	ssippi#miss	i	
	ssissippi#m	i	T[k-1]

C

#	0
i	1
m	5
p	6
s	8

Say T[k] = j^{th} char of L

Where is T[k-1] ?

T[k-1] = L[LF[j]]

Compression ?

L
a
c
#
r
a
a
b

All we did so far is a rather strange permutation of the text ?

Why is it good ?

a	b	r	a	c	a	#
b	r	a	c	a	#	a
r	a	c	a	#	a	b
a	c	a	#	a	b	r
c	a	#	a	b	r	a
a	#	a	b	r	a	c
#	a	b	r	a	c	a

F						L
#	a	b	r	a	c	a
a	#	a	b	r	a	c
a	b	r	a	c	a	#
a	c	a	#	a	b	r
b	r	a	c	a	#	a
c	a	#	a	b	r	a
r	a	c	a	#	a	b

Characters with the same (right) context appear together

Move to front

L	i	p	s	s	m	#	p	i	s	s	i	i
	1	3	4	0	4	4	3	4	4	0	1	0

- Replace each char in L with the number of distinct char's seen since its last occurrence.
- Usually will result in a lot of consecutive 0's and clusters of small numbers

Move to front – How ?

- Keep an array/list MTF[1,…,|Σ|], initially say sorted
- Replace each char by its index and move it to the front

L												
	i	p	s	s	m	#	p	i	s	s	i	i
	1	3	4	0	4	4	3	4	4	0	1	0

#	**i**	**m**	**p**	**s**
0	1	2	3	4

i	**#**	**m**	**p**	**s**
0	1	2	3	4

p	**i**	**#**	**m**	**s**
0	1	2	3	4

s	**p**	**i**	**#**	**m**
0	1	2	3	4

And so on…

Final step

- Finish the encoding by applying a prefix code such as Huffman's to the integer sequence produced by MTF

Variations

- Different prefix codes
- Use **run-length encoding**: BWT + MFT + **RLE** + prefix

Run length encoding

Replace each sequence 0^m by m+1 encoded in binary using 2 new symbols **0,1**

<u>Example</u>

1. 0 → 1+1 = 2 → **10** → **01** → **0**
2. 00 → 2+1 = 3 → **11** →**11** → **1**
3. 000 → 3+1 = 4 → **100** → **001** → **00**
4. 0000 → 4+1 = 5 → **101** → **101** → **10**
5. 0000000 → 7+1 = 8 → **1000** → **0001**→**000**

MTF(L) = 10002210000023

RLE(MTF(L)) = 100221023

RLE(MTF(L)) is defined over { **0**, **1** ,1,2,…,|Σ|-1}

Run length encoding

You can retrieve as follows

00110 = $b_0b_1b_2b_3b_4$

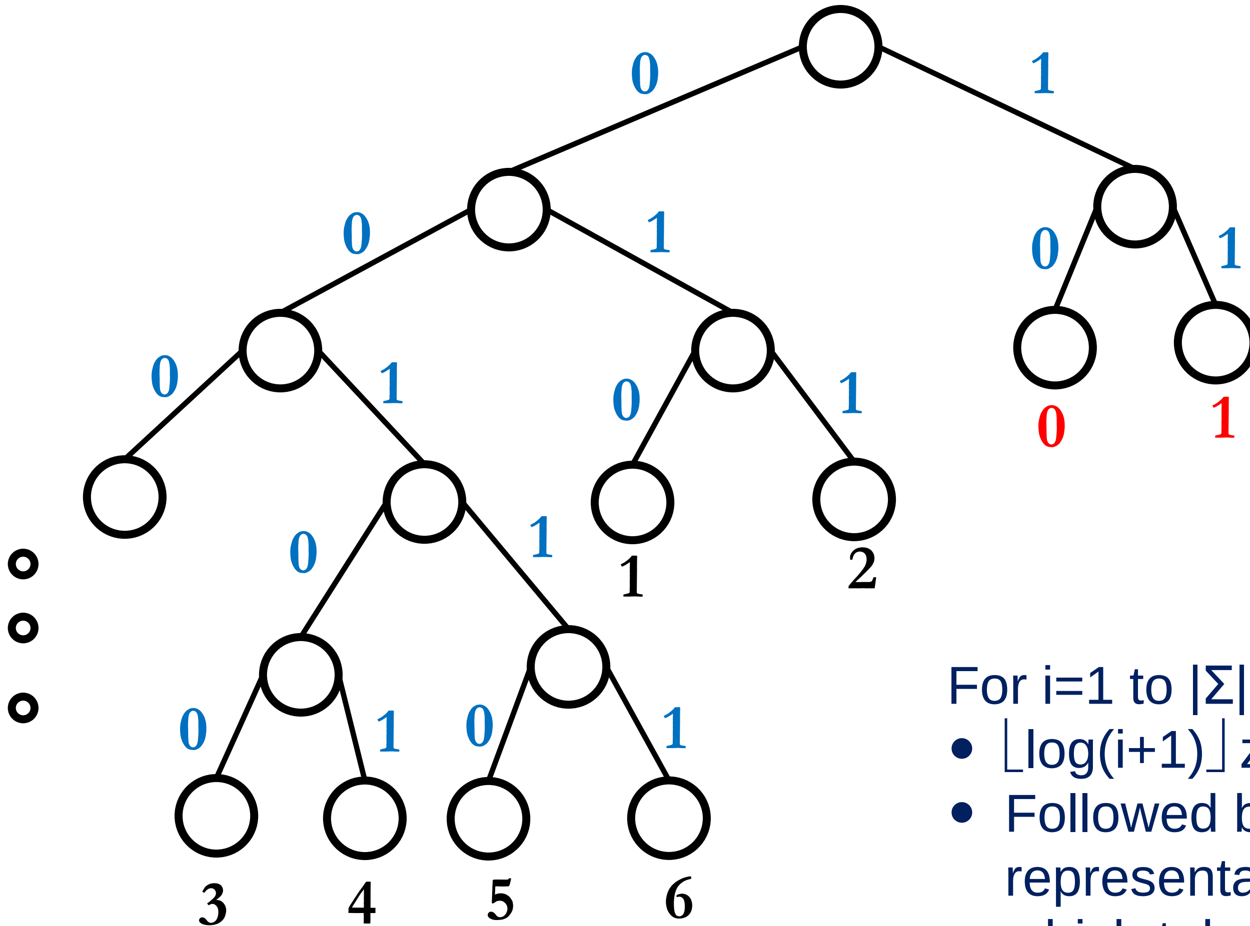
Real # of ones is $(1b_{k-1} \ldots b_3b_2b_1b_0)_2$-1=$(101100)_2$-1

Replace b_i by $(b_i$+1)2^i zeros

Total number of zeros:

$$\sum_{i=0}^{k-1}(b_i+1)2^i = \sum_{i=0}^{k-1} b_i 2^i + \sum_{i=0}^{k-1} 2^i = \left(\sum_{i=0}^{k-1} b_i 2^i + 2^k\right) - 1$$

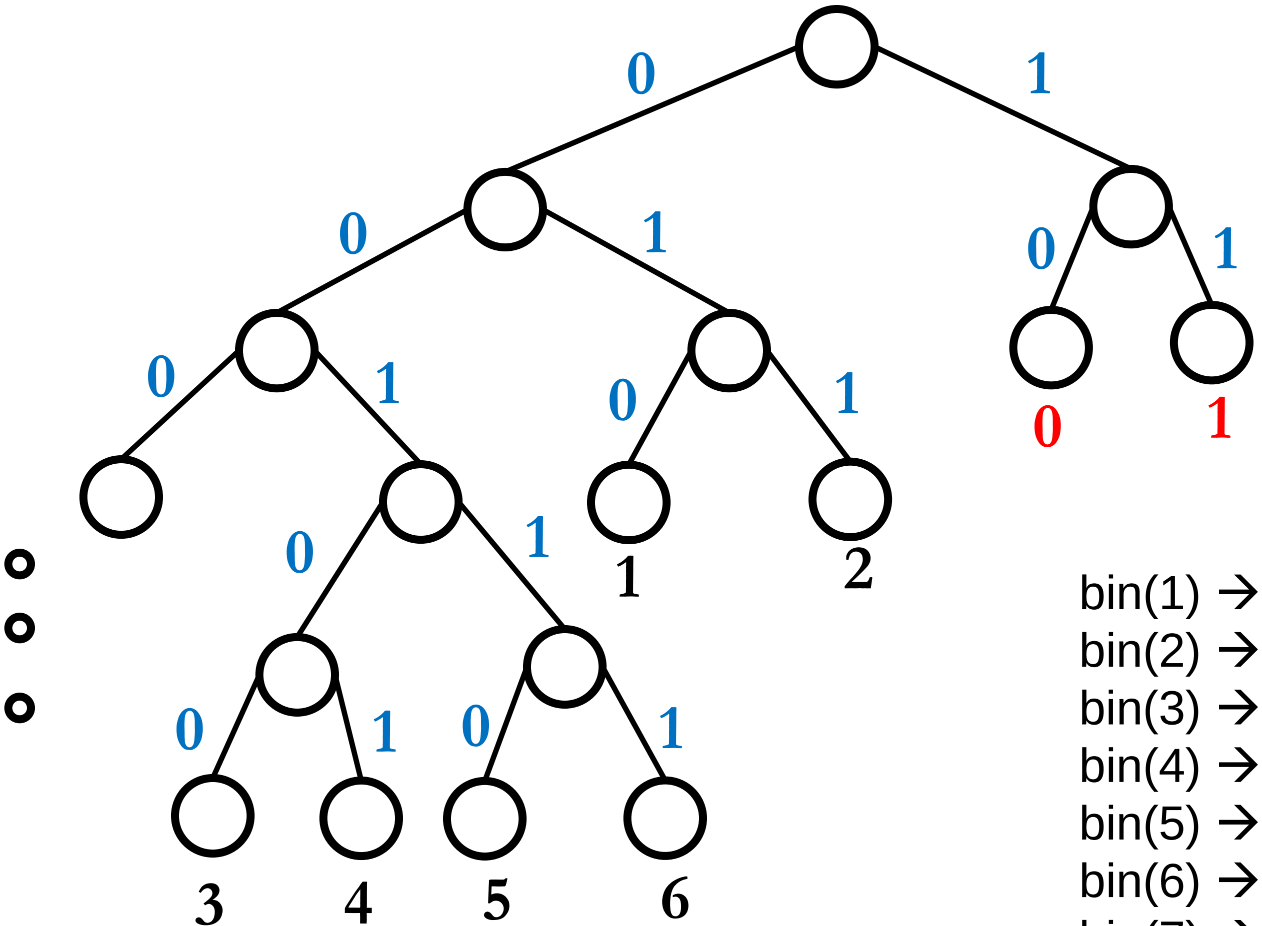
A prefix code



For i=1 to |Σ|-1

- $\lfloor \log(i+1) \rfloor$ zeroes
- Followed by a binary representation of i+1 which takes $1+\lfloor \log(i+1) \rfloor$ bits

A prefix code



bin(1) → **0**10
bin(2) → **0**11
bin(3) → **00**100
bin(4) → **00**101
bin(5) → **00**110
bin(6) → **00**111
bin(7) → **000**1000

How good is the compression?

Theorem (FM 05):

$$prefix(RLE(MTF(BWT(T)))) = 5nH_k(T) + O(\log(n))$$

$H_k(T)$ is the k^{th} order empirical entropy of T

This is true simultaneously for every k

The FM index

Substring search using the BWT
(Count the pattern occurrences)

P[j]

P = si

First step

rows prefixed by char “i”

occ=2
[lr-fr+1]

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

Available info

C

#	0
i	1
m	5
p	6
s	8

Inductive step: Given fr,lr for P[j+1,p]

❶ Take c=P[j]

❷ Find the first c in L[fr, lr]

❸ Find the last c in L[fr, lr]

❹ L-to-F mapping of these chars

slide stolen from Paolo Ferragina@

Substring search using the BWT

(Count the pattern occurrences)

$\boxed{P = si}$

First step

rows prefixed by char "i"

occ=2
[lr-fr+1]

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

Available info

C

#	0
i	1
m	5
p	6
s	8

$fr' = C[s] + occ(s,1) + 1$

$lr' = C[s] + occ(s,5)$

$fr' = C[s] + occ(s,fr-1) + 1$

$lr' = C[s] + occ(s,lr)$

Make a bit vector for each character

L

i
p
s
s
m
#
p
i
s
s
i
i

0 0 1 1 0 0 0 0 1 1 0 0

occ(s,4) = rank(4)

rank(i) = how many ones are there before position i ?

How do you answer rank queries ?

0 0 1 1 0 0 0 0 1 1 0 0

↑

rank(i) = how many ones are there before position i ?

We can prepare a vector with all answers

0 0 1 2 2 2 2 2 3 4 4 4

Lets do it with O(n) bits per character

0 0 1 1 0 0 | 1 0 1 1 0 0 | 0 0 1 1 0 0 | 0 0 1 1 0 0

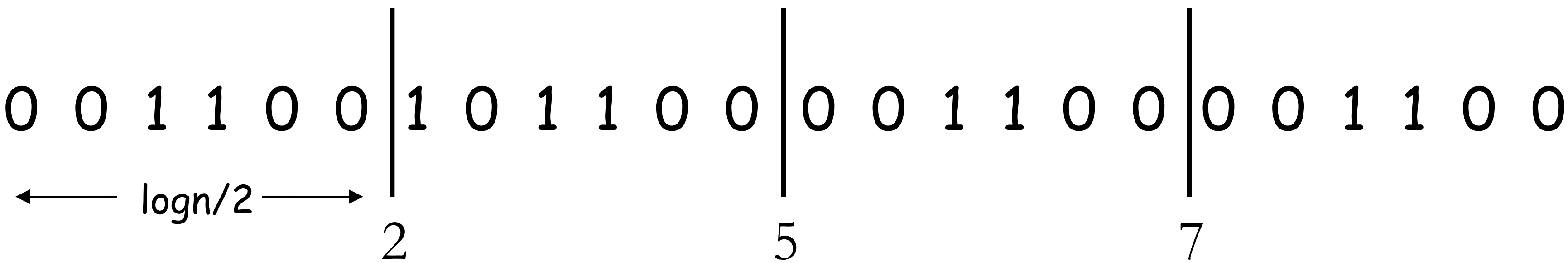
← logn/2 →

2 5 7

Partition in 2n/log(n) blocks of size log(n)/2

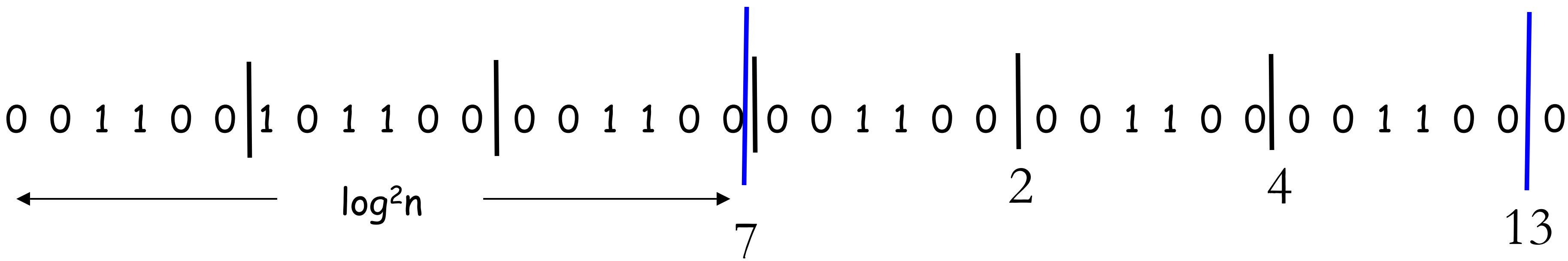
Keep the answer for each prefix of the blocks

There are $\sqrt{n}$ "kinds" of blocks, prepare a table with all answers for each block



In our solution the bit vector takes Θ(n) bits
and also the "additionals" take Θ(n) bits

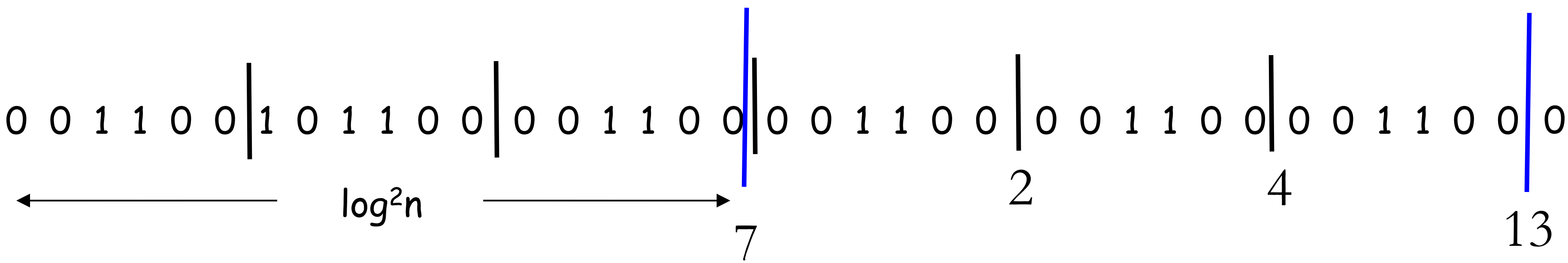
Can we do it with smaller overhead : so additionals would take o(n) ?



superblocks of size $\log^2(n)$

Each block keeps the number of one in previous blocks that are in the same superblock

Analysis



The superblock table is of size n/log (n)

The block table is of size (loglog(n)) * n/log (n)

The tables for the blocks $\sqrt{n}$ log(n)loglog(n)

So the additionals take o(n) space

Summary so far

- We can **count occurrence** using:

- The array *C*
- A bit vector per character $O(|\Sigma|n)$
- Additional tables of size $o(n)$

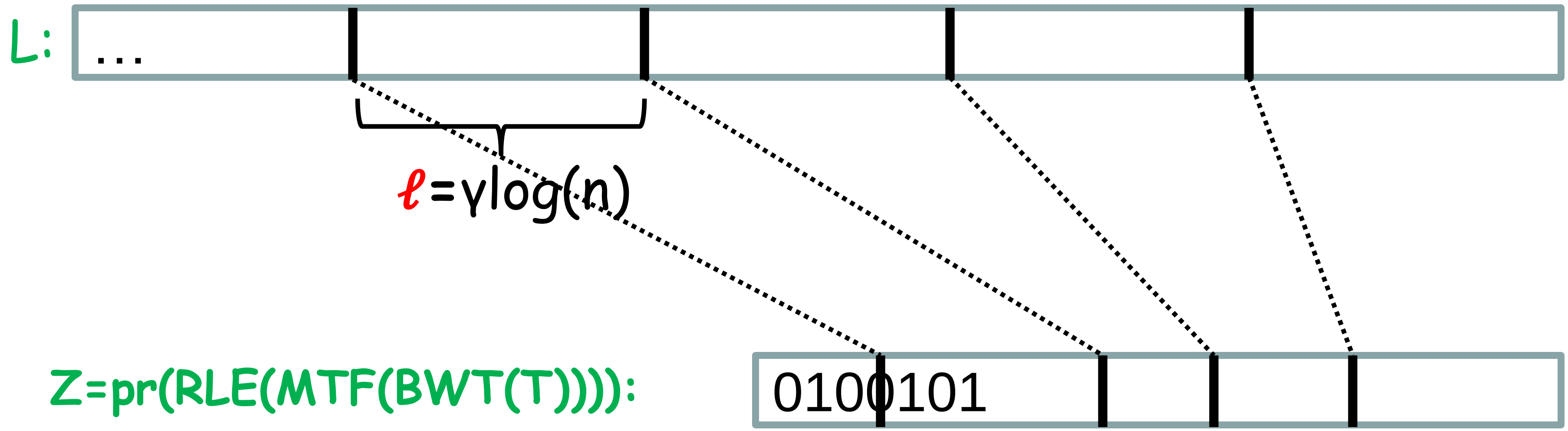
Next steps

Do it without keeping the bit vectors themselves ?

Can we do it while keeping only prefix(RLE(MTF(BWT(T)))) ?

How do we find the occurrences themselves ?

Storing prefix(RLE(MTF(BWT(T))))



Also partition into superblocks (not shown) each consisting of ℓ blocks

The tables

- NOs[c,f]: #occ of c before superblock f
- NOb[c,i]: #occ of c from the beginning of the superblock and before block i

The tables

- MTF[i]: The MTF list at the beginning of block i
- S[c,j,Z,M]: #occ of c up to char j of a block whose compressed rep. is Z and MTF at the beginning is M

We would like to use it by retrieving S[c,j,Z_i,MTF[i]]

How do we find Z_i ?

Two more tables

- Ws[f]: where does the compressed part of superblock f starts
- Wb[i]: The offset of block i within its superblock
- Retrieving block j

$$s = \left\lceil \frac{j}{\ell} \right\rceil$$

$$start = Ws[s] + Wb[j]$$

$$\begin{aligned} end = if\ (j \bmod \ell = 0)\ Ws[s+1]-1\ else \\ Ws[s]+Wb[j+1]-1 \end{aligned}$$

$$Z_j = Z[start, end]$$

Tables sizes

- Ws, Wb, NOs, NOb, MTF: O(nloglog(n)/log(n))
- What is the size of S ?

S[c,j,Z,M]: #occ of c up to char j of a block whose compressed rep. is Z and the MTF at the beginning is M

$$|Z| = \left(1 + 2\lfloor \log|\Sigma| \rfloor\right)\ell = \left(1 + 2\lfloor \log|\Sigma| \rfloor\right)\gamma \log(n)$$

Choose γ such that $|Z| \leq \delta\log(n)$

So the # of possible Z's is $\leq n^{\delta}$

➔ So the size of S is $O(n^{\delta}\ell\log(\ell))$

Find the occurrences

- We find fr,lr, so we know that there are lr-fr+1 occurrences
- But how do we find the positions of these occurrences ?
- If we had the suffix array it would have been easy

The LF mapping..

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

C

#	0
i	1
m	5
p	6
s	8

LF(i) = C[L[i]] + Occ(L[i], i)

LF(9) = C[s] + Occ(s, 9) = 11

➔ The previous suffix is at row 11

The LF mapping..

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

C

#	0
i	1
m	5
p	6
s	8

$$LF(i) = C[L[i]] + Occ(L[i], i)$$

$$LF(11) = C[i] + Occ(i, 11) = 4$$

➔ The previous suffix is at row 4

Finding the occurrences

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

C

#	0
i	1
m	5
p	6
s	8

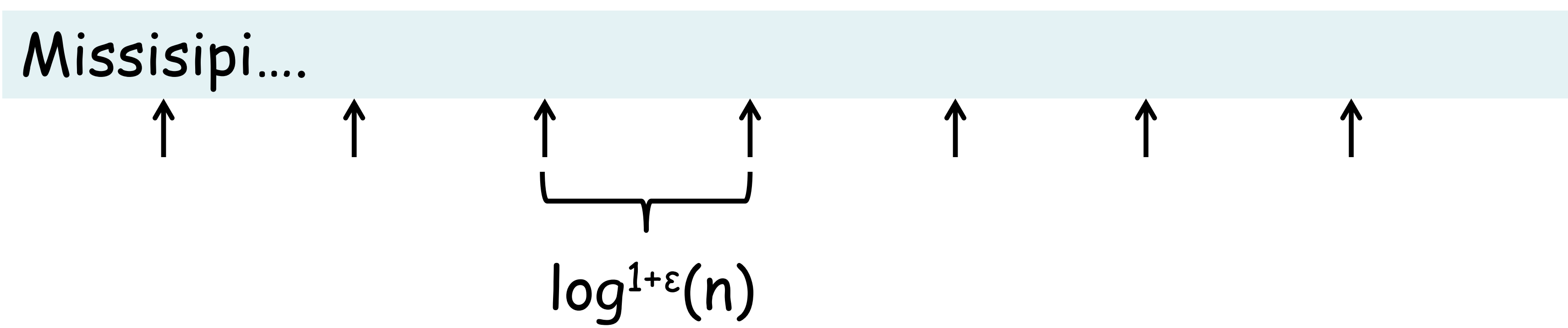
We can keep going until we get to the first prefix (and record where the first prefix is in the array)

By counting steps we discover the position

Finding the occurrences

- This may take too much time…

- Remember the positions of a subset of the suffixes in the suffix array:



Finding the occurrences

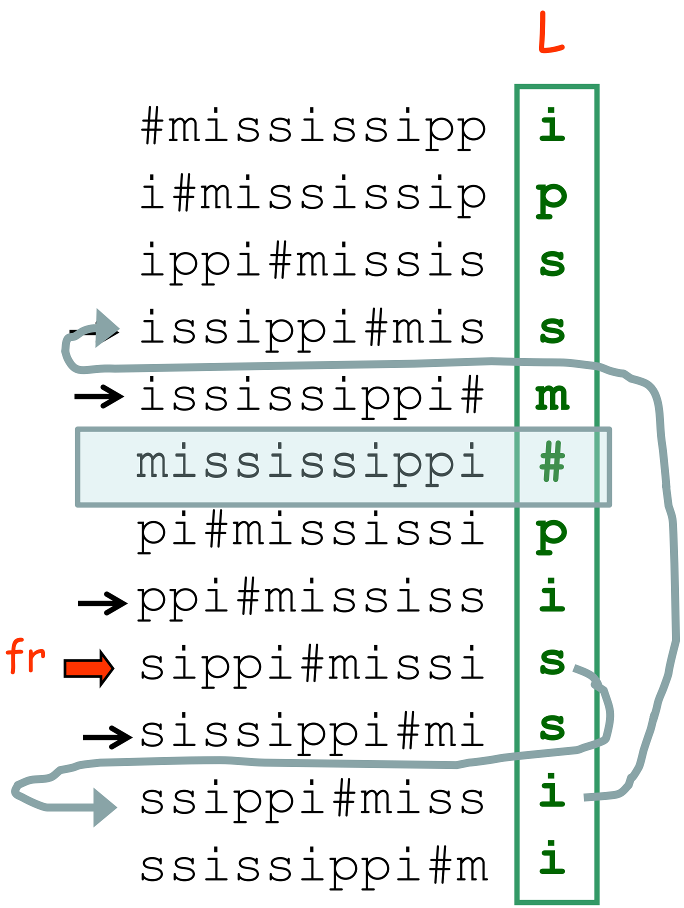
	L
#mississipp	i
i#mississip	p
ippi#missis	s
issippi#mis	s
ississippi#	m
mississippi	#
pi#mississi	p
ppi#mississ	i
sippi#missi	s
sissippi#mi	s
ssippi#miss	i
ssissippi#m	i

These suffixes are equally spaced along the text

But in the suffix array they are in arbitrary positions

Keep then in a dictionary structure like a hash table

Summary



Can find each occurrence in $O(\log^{1+\varepsilon}(n))$ time

Additional space:

$$O\left(\frac{n}{\log^{1+\varepsilon}(n)}\log(n)\right) = O\left(\frac{n}{\log^{\varepsilon}(n)}\right)$$

Time to find an occurrence:

$$O\left(\log^{1+\varepsilon}(n)\right)$$