# BACKWARD SEARCH FM-INDEX

(FULL-TEXT INDEX IN MINUTE SPACE)

# **MOTIVATION**

- Combine Text compression with indexing (discard original text).
- Count and locate P by looking at only a small portion of the compressed text.
- o Do it efficiently:
  - Time: O(p)
  - Space:  $O(n H_k(T)) + o(n)$

# 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 Pin T.
  - Locate find all positions of Pin T.

#### PURROWS WHEELER TRANSFORM

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



i	ssippi#mis	S
m	issi	
р	i#mississi	p
р	pi#mississ	i
S	ippi#missi	S
S	issippi#mi	S
S	sippi#miss	i
S	sissippi#m	i

#### PURROWS WHEELER TRANSFORM

•

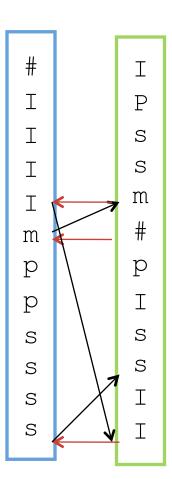
- Find F by sorting L
- First char of T?

m

- 3. Find m in L
- 4. L[i] precedes F[i] in T. Therefore we get

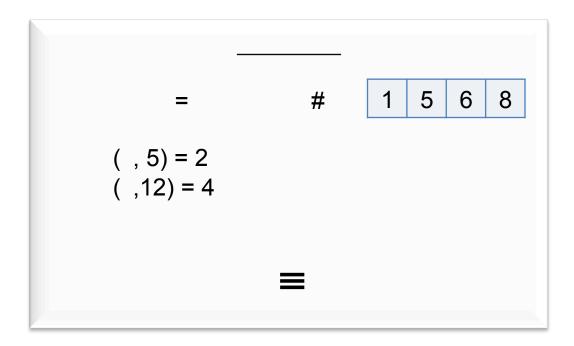
mi

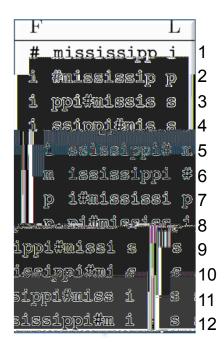
- 5. How do we choose the correct i in L?
  - The i's are in the same order in L and F
  - As are the rest of the char's
- 6. i is followed by s: mis
- 7. And so on....





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

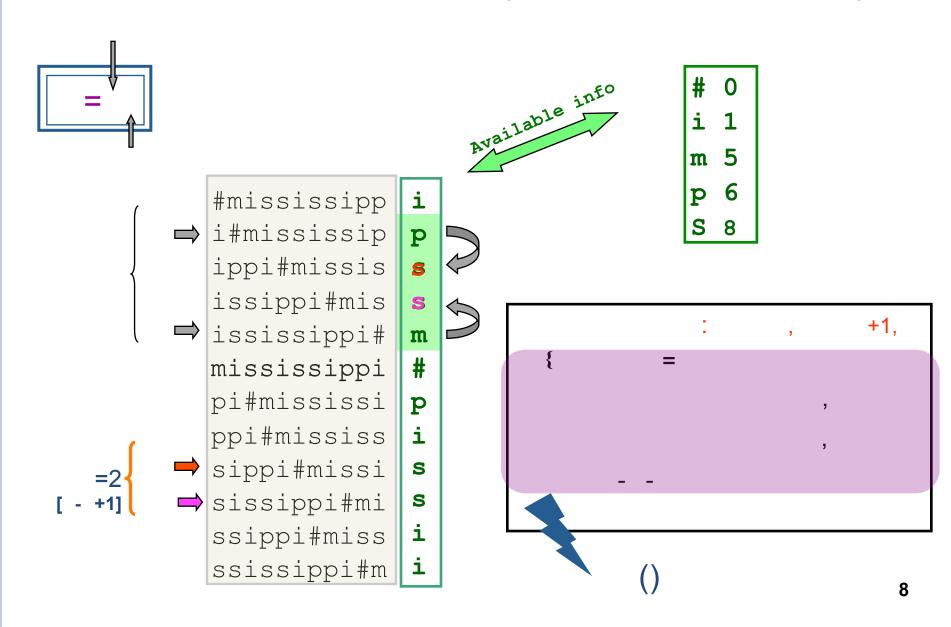




Works in piterations, from p down to 1

- Remember that the BWT matrix rows = sorted suffixes of T
  - All suffixes prefixed by pattern P, occupy a continuous set of rows

#### SUBSTRING SEARCH IN T (COUNT THE PATTERN OCCURRENCES)



# BACKWARD-SEARCH EXAMPLE

```
\circ P = pssi
                                                              # mississipp i
                                                              i Aniscissip p |2
                                                              i ppi#missis s
                                                              i ssippi#mis s
                                                              i ssissippi# m
     C =
                                                              m ississippi #
                                                                piuminusiauseum 7

    First =

                     + ( ,1) +1 = 8+0+1 = 9
                                                                njamiseies i 8
                                                                s ippi#missi 9

    Last =

                             (,5) = 8+2 = 10
                                                                s issippi#mi 10
                                                                s sippi#miss 11
   (Last – First + 1) =
                                                                 s sissippi#m 12
                                                                      5
                                                                          6
                                                                             8
```

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

(P[1,p]) = (P[1,
```

# RACKWARD-STARCH EXAMPLE

```
o P = pssi
                                                                 # mississipp i
                                                                 i Amiscissip p 2
                                                                 i ppi#missis s
                                                                 i ssippi#mis s
                                                                 i ssissippi# m
     C =
                                                                 m ississippi #
                                                                  pitistausistusistus 7

    First =

                             ( ,8) +1 = 8+2+1 = 11
                                                                   njamissies i 8
                                                                   s ippi#missi 9

    Last =

                              (,10) = 8+4 = 12
                        +
                                                                   s issippi#mi 10
                                                                   s sippi#miss 11
   • (Last - First + 1) =
                                                                   s sissippi#m 12
                                                                         5
                                                                             6
                                                                                8
```

```
Algorithm backward_search(P[1,p])
i = p, c + p \text{ of } f \text{ of
```

# RACKWARD-STARCH EXAMPLE

```
\circ P = pssi
                                                                   mississipp i
                                                                 i Anississip p 2
                                                                 i ppi#missis s
                                                                 i ssippi#mis s
                                                                 i ssissippi# m
     C =
                                                                 m ississippi #
                                                                   pitistausistusistus 7

    First =

                             ( ,10) +1 = 6+2+1 = 9
                                                                    njamissies i 8
                                                                    s ippi#missi 9

    Last =

                               (,12) = 6+2 = 8
                                                                    s issippi#mi
                                                                    s sippi#miss 11
   • (Last - First + 1) =
                                                                    s sissippi#m 12
                                                                          5
                                                                             6
                                                                                 8
```

Create a simple search program that implements

- Your C/C++ program, called bwtsearch
  - Bwtsearch -e fileToBeEncoded outputFile
  - Bwtsearch -d fileToBeDecoded
    - ostandard output
  - Bwtsearch -s fileEncoded "queryString"
    - •Output all the lines contain "queryString"
    - oHighlight "queryString" if capable
    - The search results need to be sorted according to their line numbers.

- The first four bytes (an int) of each given BWT encoded file are reserved for storing the position (zero-based) of the BWT array that contains the last character. As a result, a given BWT encoded file in this assignment is 4 bytes larger than its original text file.
- For example, if the original text file contains only banana\$, then the BWT encoded file will be 11 bytes long. The first four bytes contain the integer 4 and the rest of the bytes contain annb\$aa. i.e., The last character is at position 4 (= the fifth character since it is zero -based).

• Since each line is delimited by a newline character, your output will naturally be displayed as one line (ending with a '\n') for each match. No line will be output more than once, i.e., if there are multiple matches in one line, that line will only be output once.

- Your solution can write out one external index file.
- You may assume that the index file will not be deleted during all the tests for a given BWT file, and all the test BWT files are uniquely named. Therefore, to save time, you only need to generate the index file when it does not exist yet.

# LECTURE 5

Compressed suffix array / BWT

# SUCCINCT SUFFIX ARRAYS BASED ON RUN-LENGTH ENCODING \*

#### VELI MÄKINEN<sup>†</sup>

Dept. of Computer Science, University of Helsinki
Gustaf Hällströmin katu 2b. 00014 University of Helsinki Finland
whakinen@cs.helsinki fi

GONZAHO NAVARRO

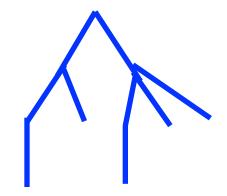
Dept. of Computer Science, University of Chil

Plance Freedad 2120 Santiaga Ghilem.
gnavarro@dcc.uchile.cl

# A B

### **ARBITRARY ORDERED TREES**

- Use parenthesis notation
- Represent the tree



- As the binary string (((())())(())())): traverse tree as "(" for node, then subtrees, then ")"
- 2 Bits per node

# **SPACE FOR TREES**

O

C

# STANDARD REPRESENTATION

# CAN WE IMPROVE THE SPACE BOUND?

o 2<sup>2</sup>

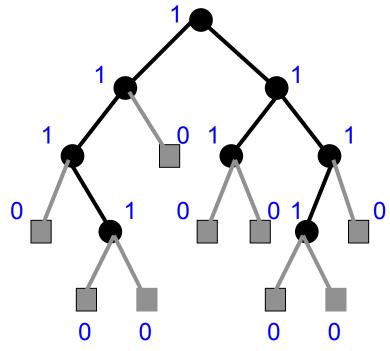
02

2

# HEAP-LIKE NOTATION FOR A BINARY TREE

. Ո

1111011010010000



Α

2 +1

### RANK/SELECT ON A BIT VECTOR

```
B
                                  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
                               B: 0 1 1 0 1 0 0 0 1 1 0 1 1 1 1
_{1}(\ )=\#\ 1
                                B
 1()=
                               В
                                                             _{1}(5) = 3
                                                              _{1}(4) = 9
                                                             _{0}(5) = 2
           ( )-
                                                               _{0}(4) = 7
```

Α

# **BINARY TREE REPRESENTATION**

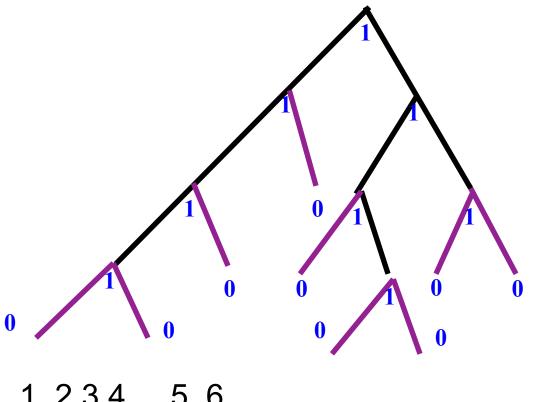
```
o A 2 + ()
```

- •
- •

# 

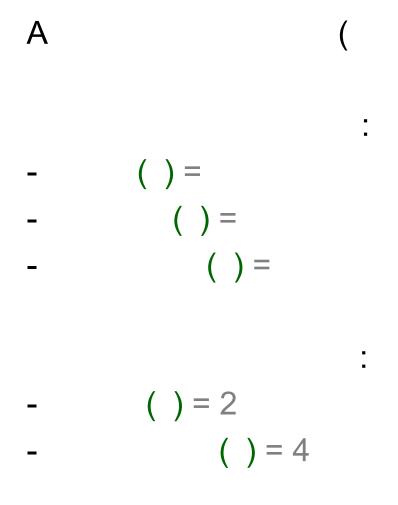
#### HEAP-LIKE NOTATION FOR A BINARY TREE

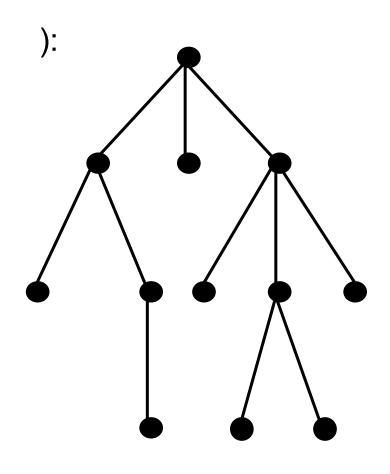




```
1 2 3 4 5 6
1 1 1 1 1 0 1 1 1 0 0 1 0 00000
1 2 3 4 5 6 7 8 9 0 1 2 34567
```

# **ORDERED TREES**



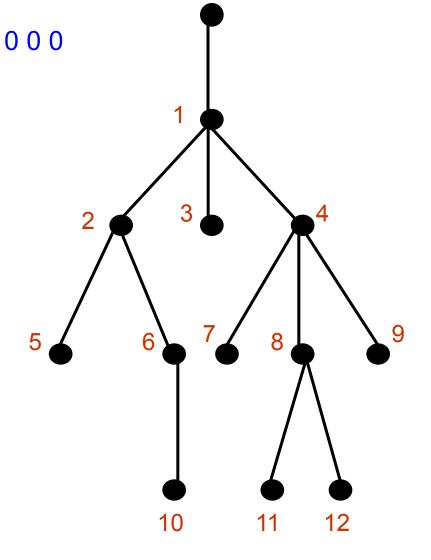


# **ORDERED TREES**

```
o A
                              2 + ()
 2 + ( )
```

# **SUPPORTING OPERATIONS**

```
1011101100111001001100000
 234 56 789
                10
    ( ) = # 0
```



# 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.
- oLet L be the last column and F be the first column.
- obwt(T)=L associated with the row number of T in the sorted M.

# **EXAMPLE**

12345

sa <sub>I</sub> M

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

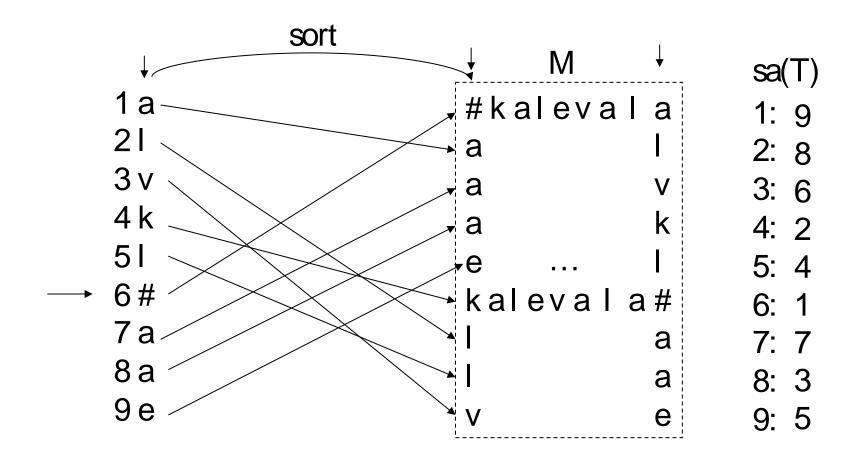
#

L = alvkl#aae, row 6

==>

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

#### T-1=#alavelak



i 123456789 LF[i] 27968 1345

## IMPLICIT LF[I]

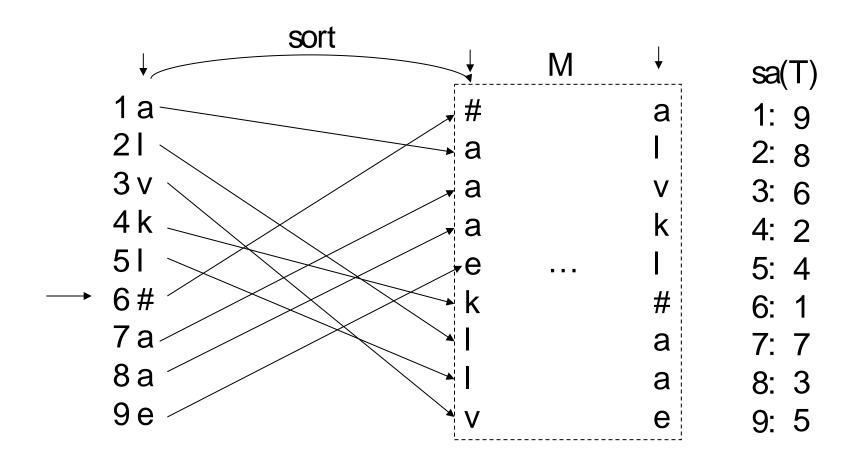
## RANK/SELECT

```
select<sub>1</sub>(L,j) 3 6 9 10 12

L 001001001101

rank<sub>1</sub>(L,i) 001112223445
```

#### T-1=#alavelak



i 123456789 LF[i] 27968 1345

$$LF[7]=C_T[a]+rank_a(L,7)$$
  
=1+2=3

# 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

```
i' := C_T[c] + rank_c(L,i-1) + 1,

j' := C_T[c] + rank_c(L,j).
```

# BACKWARD SEARCH ON BWT(T)...

- Array  $C_T[1,\sigma]$  takes  $O(\sigma \log |T|)$  bits.
- o L=Bwt(T) takes  $O(|T| \log \sigma)$  bits.
- Assuming rank<sub>c</sub>(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 FMindex 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  $C_S[1,c]$ ,
  - wavelet tree is build on S, and
  - some formulas are changed.

## RUN-LENGTH FM-INDEX...

L	В	S	L	F	B'
С	1	С	C	∫a	1
С	0	a	C	<b>a</b>	0
С	0	g	С	{a	1
a	1	a	a	C	1
a	0	t	a	d       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

## CHANGES TO FORMULAS

- Recall that we need to compute  $C_T[c]$ +rank<sub>c</sub>(L,i) in the backward search.
- Theorem: C[c]+rank<sub>c</sub>(L,i) is equivalent to
  - select<sub>1</sub>(B',C<sub>S</sub>[c]+1+rank<sub>c</sub>(S,rank<sub>1</sub>(B,i)))-1, when L[i]  $\neq$  c,
  - select<sub>1</sub>(B',C<sub>S</sub>[c]+rank<sub>c</sub>(S,rank<sub>1</sub>(B,i)))+
     i-select<sub>1</sub>(B,rank<sub>1</sub>(B,i)), otherwise

## EXAMPLE, L[I]=C

```
LF[8] = select_1(B', C_s[a] + rank_a(S, rank_1(B, 8))) +
             B S B'
                                   8-select<sub>1</sub>(B,rank<sub>1</sub>(B,8))
       a 1 c 1
                                  = select<sub>1</sub>(B',0+rank<sub>a</sub>(S,4))+8-select<sub>1</sub>(B,4)
       a 0 a 0
                                  = select<sub>1</sub>(B',0+2)+8-8
      a 0 g 1
                                  = 3
       c 1 a 1
a
a
g
g
       g
a
```

o For more detail, read the original paper

## EXERCISE

- o ipsm\$pisi
- 111011111010

## WHAT IS B'

$\boldsymbol{i}$	$\mathbf{B}$	<u>S</u>
1	<u>B</u>	<u>S</u>
<ol> <li>1</li> <li>2</li> <li>3</li> <li>4</li> <li>5</li> <li>6</li> <li>7</li> <li>8</li> <li>9</li> <li>10</li> </ol>	1	p s
3	1	S
4	0	
5	1	m
6	1	\$
7	1	\$ p i
8	1	i
9	1	S
10	0	
11	0 1 0	i
12	0	

## USUALLY B' IS GIVEN TO SAVE COMPUTATIONS

i	В	<u>S</u>	<u>B'</u>
1	<u>B</u>	<u>S</u> i	
2	1		1 1
3	1	p s	1
4	0		
5	1	m	0
6	1	m \$ p i	1
7	1	р	1 1
8	1	i	1
9	1	S	1
10	0		1 0 1
11	1	i	1
<u>i</u> 1234567891011	0		0

### REVERSE BWT FROM ROW 6

i	В	<u>S</u>	<u>B</u> '
1	<u>B</u>	<u>S</u>	
2	1	р	1
3	1	p s	1
4	0		1
5	1	m	0
6	1	m \$ p i	1
7	1	р	1
8	1	i	1
9	1	S	1
10	0		0
11	1	i	1 1 1 0 1 1 0 1 0
<i>i</i> 1 2 3 4 5 6 7 8 9 10 11 12	0		0

i	$\mathbf{\underline{B}}$	<u>S</u>	<u>B'</u>
1	<u>1</u>	<u>S</u>	1
2	1	р	1
3	1	p s	1
4	0		1
5	1	m	0
6	1	m \$	1
7	1	p	1
8	1	i	1
9	1	S	1
10	0		0
11	1	i	1
<i>i</i> 1 2 3 4 5 6 7 8 9 10 11 12	0		0

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

```
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)))
5 1
       m
6 1 $
                   = select_1(B', 0 + rank_s(S, 5)) + 6 - select_1(B 5)
                   = 1 + 6 - 6 = 1
8 1
10
12
```

```
S[rank_1(B, 1)] = i
            \frac{\mathbf{S}}{\mathsf{i}}
                             LF[1]
                             = select<sub>1</sub>(B', C<sub>S</sub>[i] + rank<sub>i</sub>(S, rank<sub>1</sub>(B, 1))) + 1
4 0
5 1
                             -\operatorname{select}_1(B, \operatorname{rank}_1(B, 1)))
            m
6
     1 $
                             = select_1(B', 1 + rank_i(S, 1)) + 1 - select_1(B, 1)
8 1
                             = 2 + 1 - 1 = 2
10
12
```

```
S[rank_1(B, 1)] = i
                             LF[1]
                             = select<sub>1</sub>(B', C<sub>S</sub>[i] + rank<sub>i</sub>(S, rank<sub>1</sub>(B, 1))) + 1
5
                             -\operatorname{select}_1(B, \operatorname{rank}_1(B, 1)))
6
                             = select_1(B', 1 + rank_i(S, 1)) + 1 - select_1(B, 1)
8
10
12
```

#### **BACKWARD SEARCH**

```
Suppose search for si:
                  c = i, First = 2, Last = 5
                  C = S
4 0
5 1
                  First = C[c] + Occ(c, First - 1) + 1
        m
6 1 $
                  Last = C[c] + Occ(c, Last)
10 0
12
```

#### **BACKWARD SEARCH**

```
c = i, First = 2, Last = 5
          \frac{S}{i} \frac{B'}{1}
                         C = S
                         First = select<sub>1</sub>(B', C_s[s]+1+rank_s(S, rank_1(B,
                         (2-1))) -1 + 1
5 1 m 0
                         =select<sub>1</sub>(B',7+1+rank<sub>s</sub>(S,1))
6 1 $
                         = select<sub>1</sub>(B', 8) = 9
                         Last = select<sub>1</sub>(B', C_s[s]+1+rank_s(S, rank_1(B,5)))
10 0
                         = select<sub>1</sub>(B',7+1+rank<sub>s</sub>(S,4)) - 1
12
                         =select<sub>1</sub>(B', 9) -1 = 11 - 1 = 10
```