

Suffix arrays

Space-efficient alternative to suffix trees

Lecture 4.1

Full-text indexes – continued

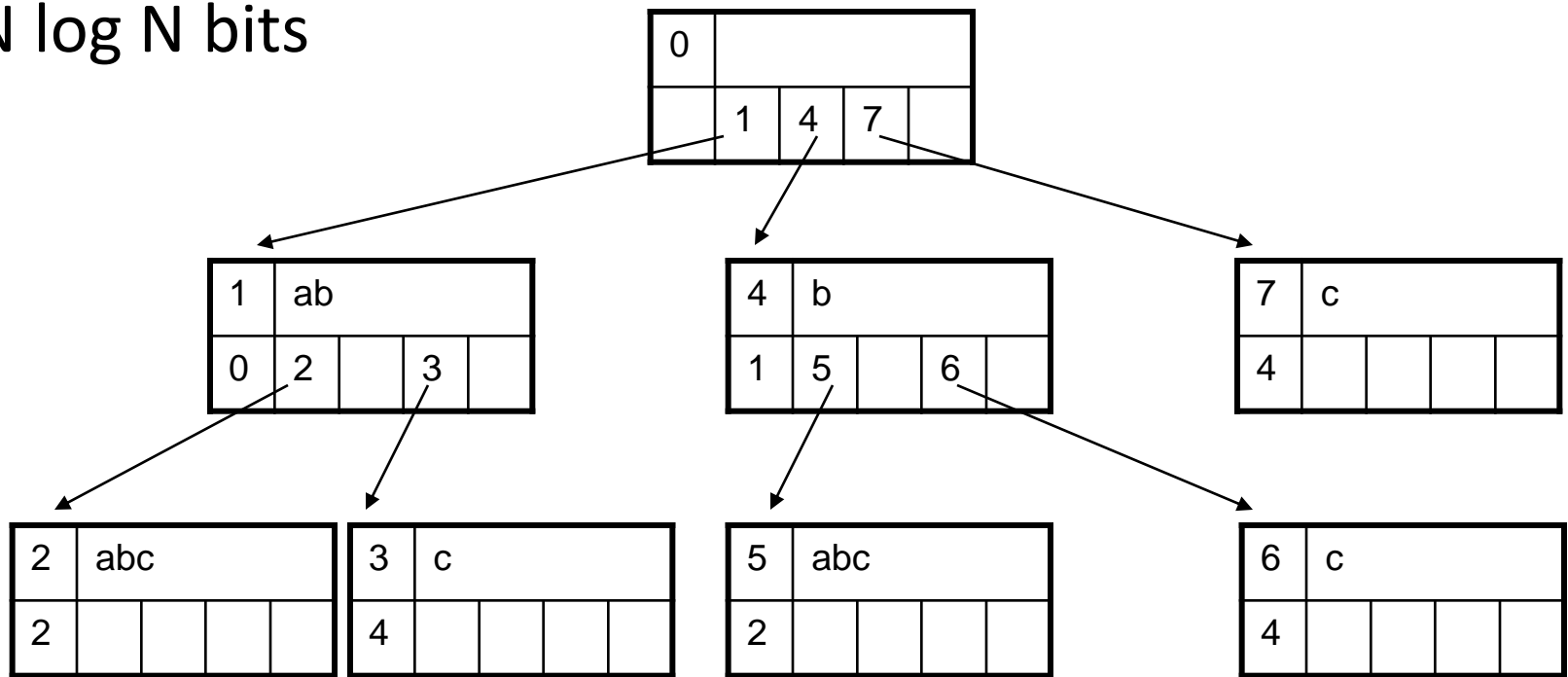
- In order to be able to find any substring in a **large static string** database we need to **index all possible substrings**
- How many possible substrings in text of length N ?
 N^2
- We would like all the substrings to be **sorted** – this will allow to search for any substring in time $O(\log N)$ using **binary search**

Brute-force – not feasible

- We need to *index* substrings **only if the text is large** – so large that the linear-time scanning of the text is out-of-question – takes too long
- But if we sort all possible substrings for a very large text of length N – there will be N^2 substrings to store
- And what is the size of each substring?
 $O(N)$
- Thus the brute-force approach will produce an index of **size $O(N^3)$**
- Where are we going to store N^3 values for large N ?
 $(3 \cdot 10^9)^3 = 27 \cdot 10^{27}$ (bytes) – 27 brontobytes or 1,125,899,906,842,624 terabytes

All different substrings can be exposed through the suffix tree of T

For each **byte**— 2 x 2 **numbers**. Each number $\log N$ bits
 $4N \log N$ bits



Linear space - but heavy constants

a **b** **a** **b** **c**
 00 01 00 01 10

Estimated size of Suffix Trees

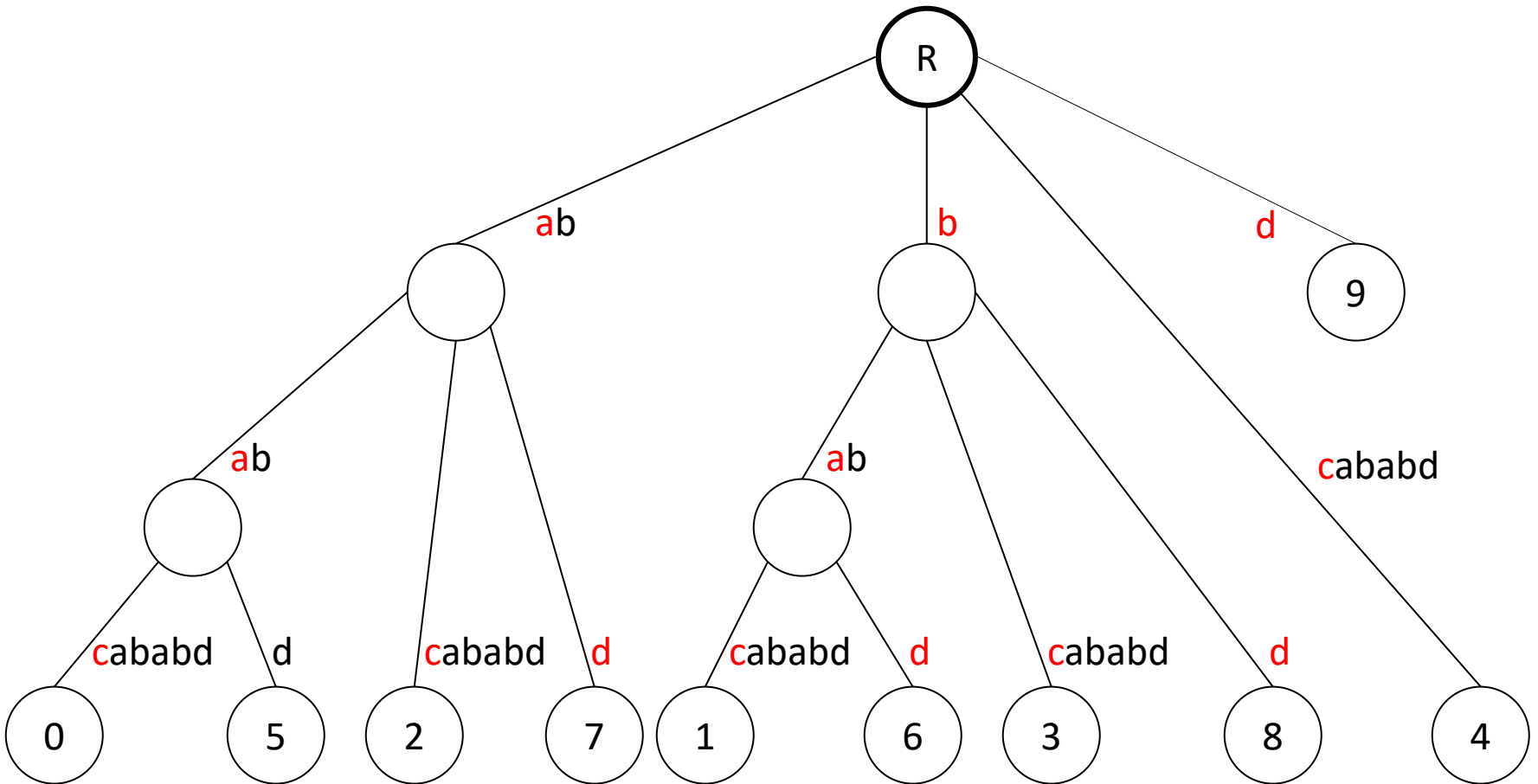
	input	suffix tree
• Human genome	3 GB	48 GB
• Corn genome	5 GB	58 GB
• All of GenBank	108 GB*)	1.5 TB
• Amoeba genome	670 GB	~30 TB
• 1000 genomes	20 TB ?	???

*)As of Jan 2010 <http://www.cbs.dtu.dk/databases/DOGS/GBgrowth.php>

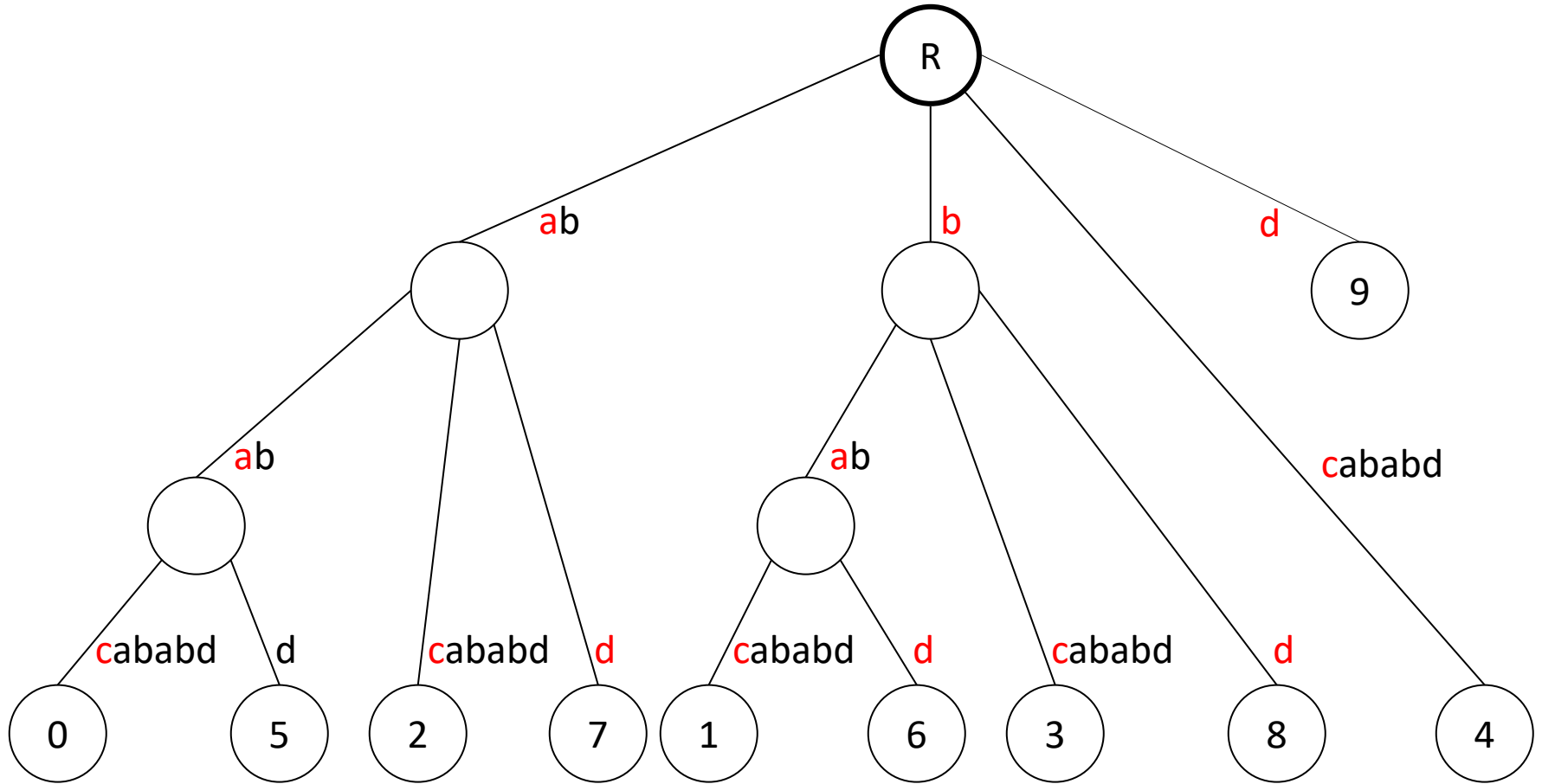
From suffix tree to suffix array

a b a b c a b a b d

0 1 2 3 4 5 6 7 8 9



<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>d</i>
0	1	2	3	4	5	6	7	8	9



If we traverse the suffix tree by processing children in lexicographical order of their edges, then we can collect all the suffixes in lexicographical order:

0,5,2,7,1,6,3,8,4,9

This sequence of suffix start positions is called a **Suffix Array**

Suffix array: example

S	a	b	a	b	a	a	a	b	b	c
	0	1	2	3	4	5	6	7	8	9

suffix start	4	5	2	0	6	3	1	7	8	9
	a	a	a	a	a	b	b	b	b	c
	a	a	b	b	b	a	a	a	b	
	a	b	a	a	b	a	b	b	c	
	b	b	a	b	c	a	a	b		
		
LCP	0	2	1	3	2	0	2	3	1	0

← SA
↓ Suffixes in lexicographical order

The *suffix array SA* of string S is defined to be **an array of integers** providing the starting positions of suffixes of S in lexicographical order.

Build SA for *cocoa*

<i>c</i>	<i>o</i>	<i>c</i>	<i>o</i>	<i>a</i>	N
0	1	2	3	4	

Alphabetically sort suffixes: **Suffix array**

4	2	0	3	1	O(N)
a	c	c	o	o	O(N ²)
	o	o	a	c	
	a	c		o	
		o		a	

Suffix array: space

- For a string of length N bytes
 - Total space is N numbers. Each number can be represented with $\log N$ bits
 - So the space is just $N \log N$ bits
- Also because it is sorted, it can be partitioned, distributed, and searched in parallel

Simple **binary search** using SA for S

S	a	b	a	b	a	a	a	b	b	c
	0	1	2	3	4	5	6	7	8	9

SA	4	5	2	0	6	3	1	7	8	9
----	---	---	---	---	---	---	---	---	---	---

Search for pattern: *bab*

$N = 10$

- String at pos SA $[N/2] = SA[5] = 3$: *baa* < *bab*

=> search to the right of $N/2$

- String at pos SA $[N/2+N/4] = SA [5+2]=7$: *bbc* > *bab*

=> Search between $N/2$ and $N/2+N/4$

- String at pos SA $[N/2 + N/8] = SA [6] = 1$: *bab* = *bab*