

Grammar-Based Compression

Slides by Carl Kingsford

Context-free Grammars

Def. A context-free grammar is a collection of rules of the form:

$$A \rightarrow x_1x_2x_3\dots x_k$$

where $x_1x_2x_3\dots x_k$ are either terminal symbols (letters in the alphabet Σ) or symbols that appear on the left-hand side of some rule.

$$S \rightarrow ABC$$

$$A \rightarrow xxDz$$

$$B \rightarrow ww$$

$$C \rightarrow aAa$$

$$D \rightarrow zB$$

$$S \rightarrow xxDzwwaAa$$

$$S \rightarrow xxzBzwwaxxDza$$

$$S \rightarrow xxzwwzwwaxxzBza$$

$$S \rightarrow xxzwwzwwaxxzwwza$$

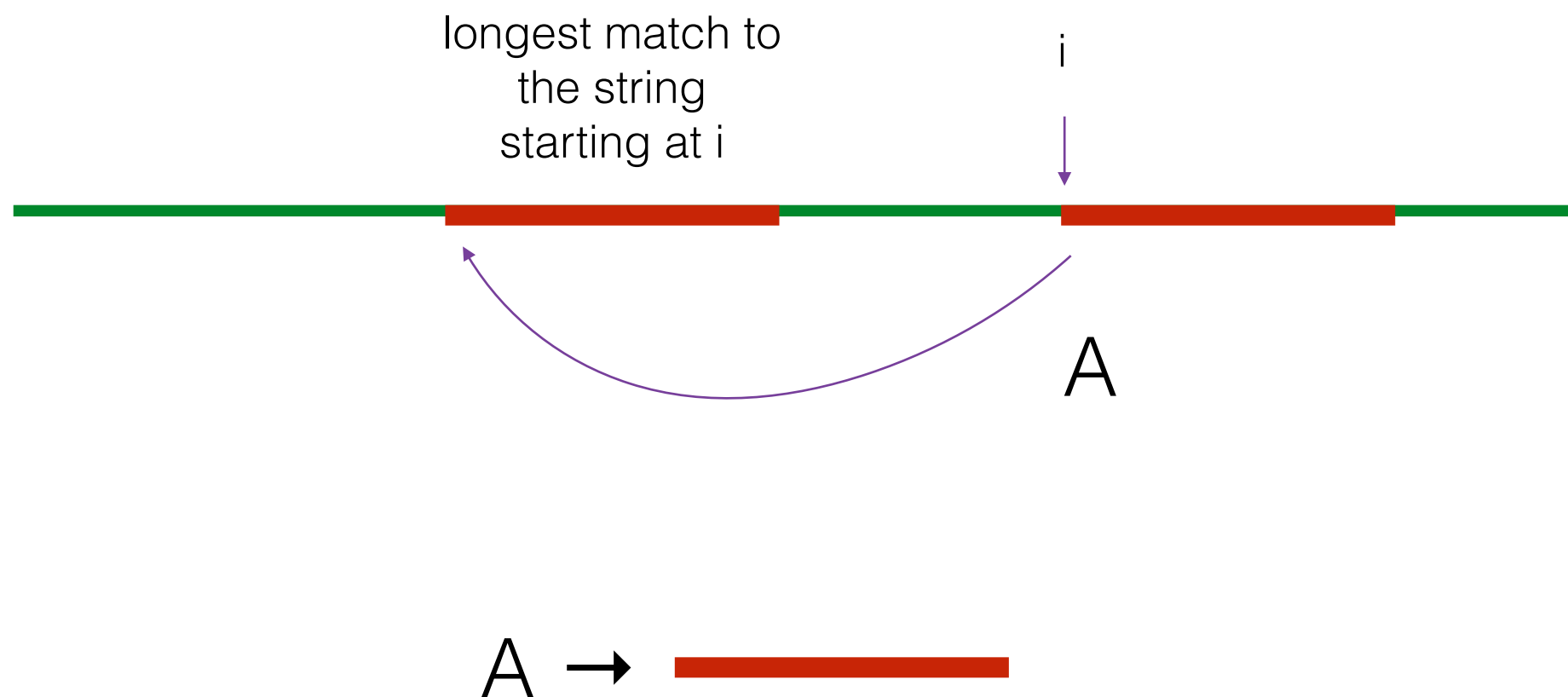
Ziv-Lempel as a Grammar

longest match to
the string
starting at i

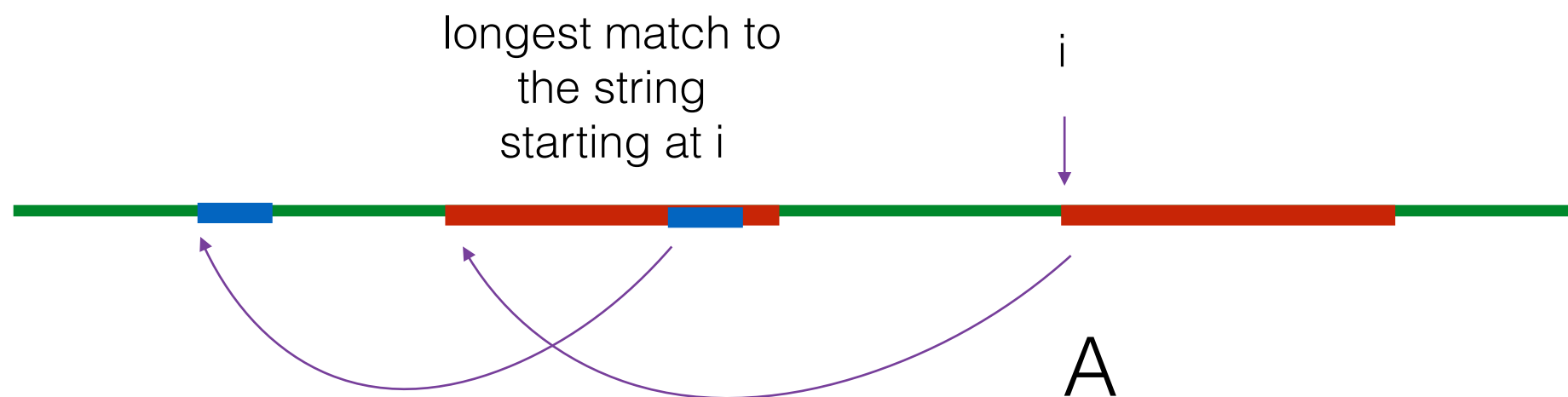
i




Ziv-Lempel as a Grammar



Ziv-Lempel as a Grammar



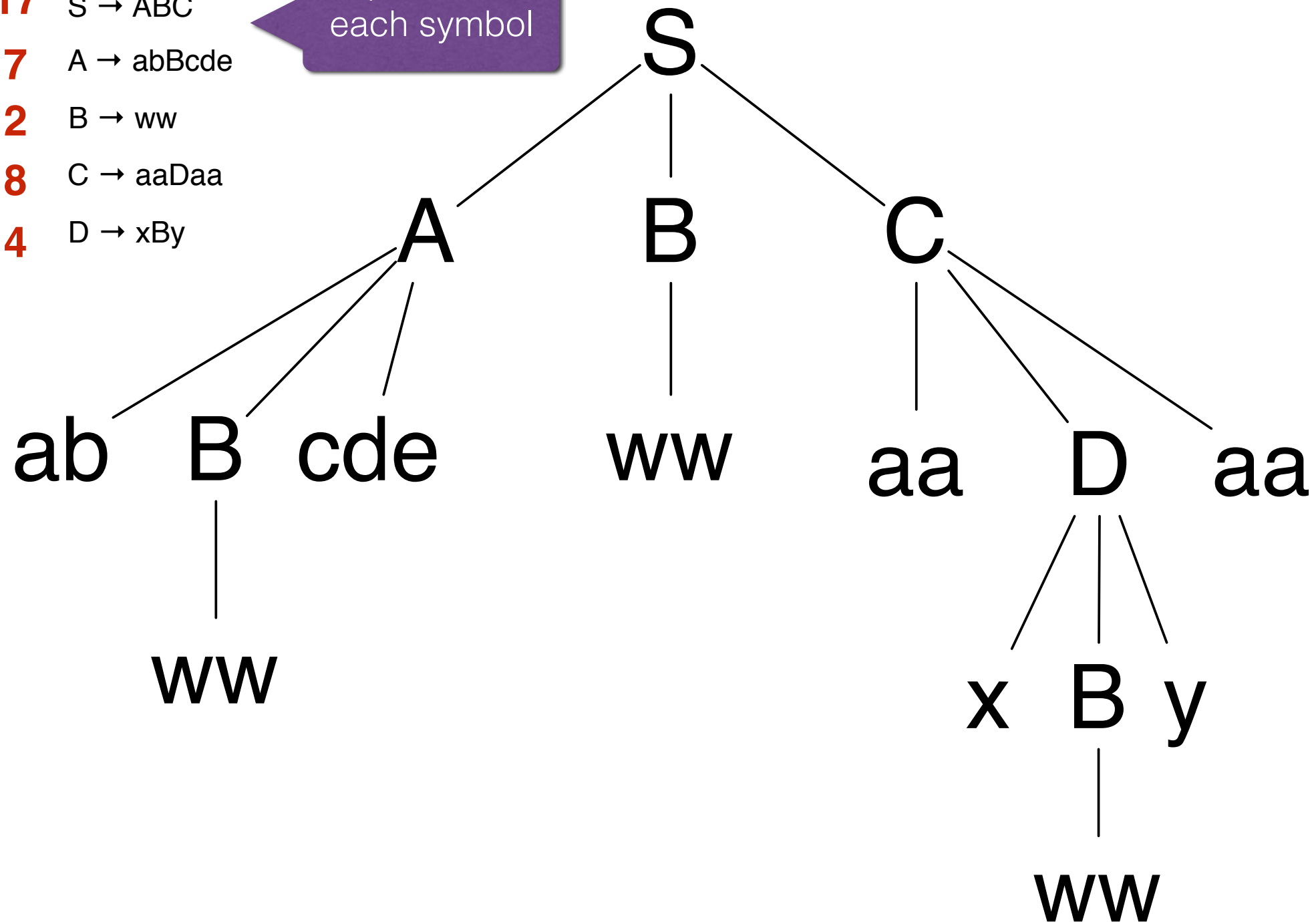
A →   

B → 

Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

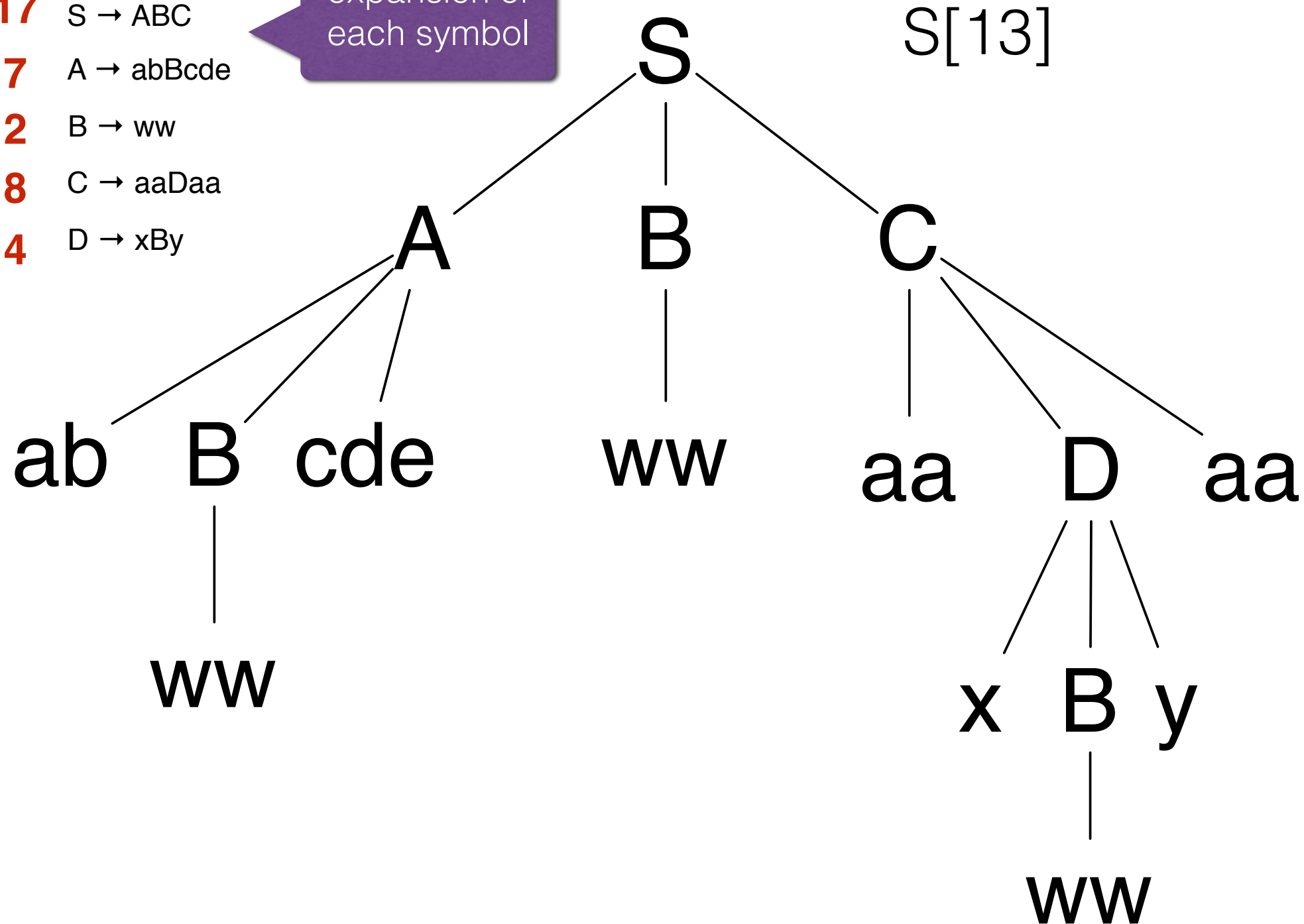
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

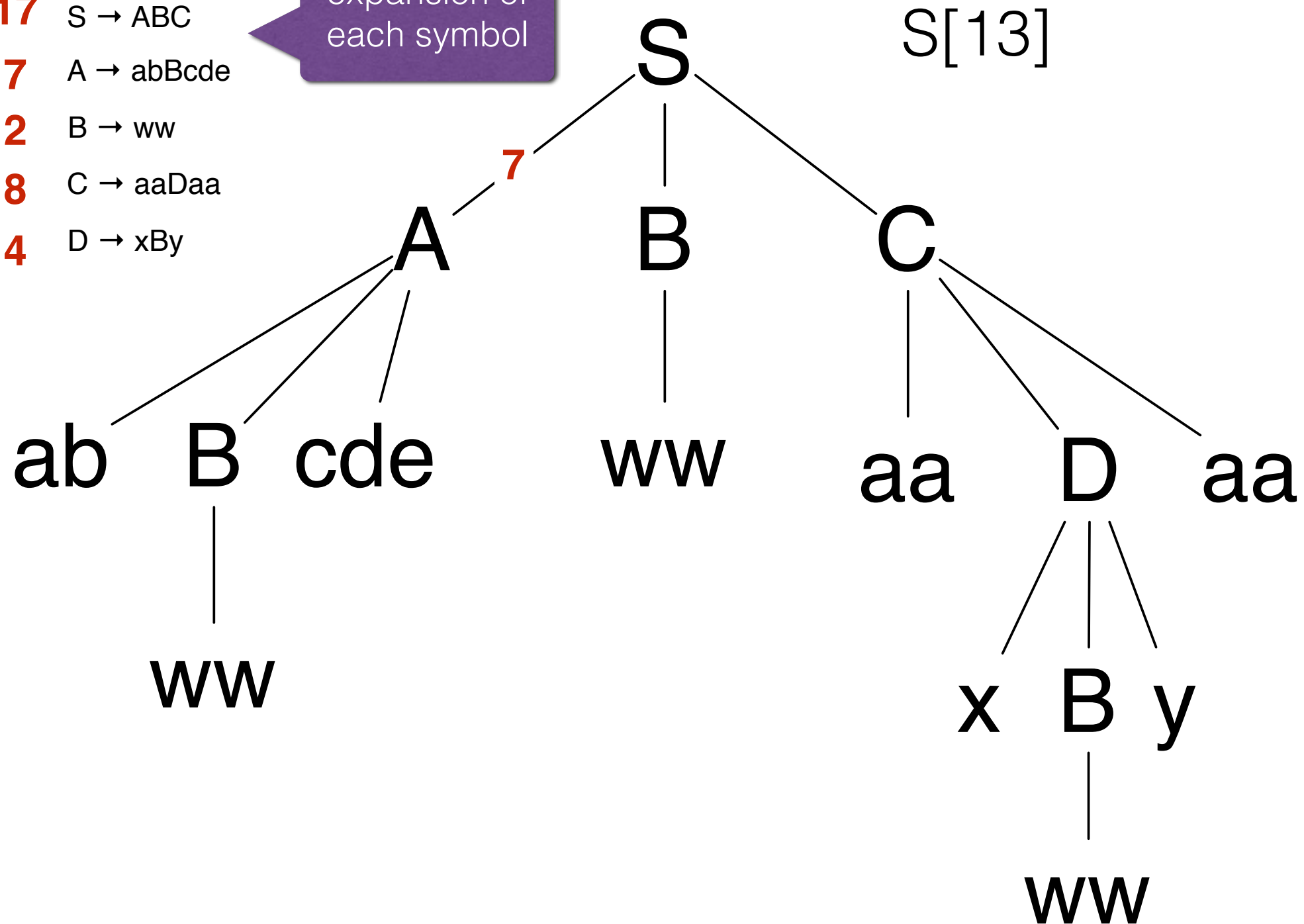
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

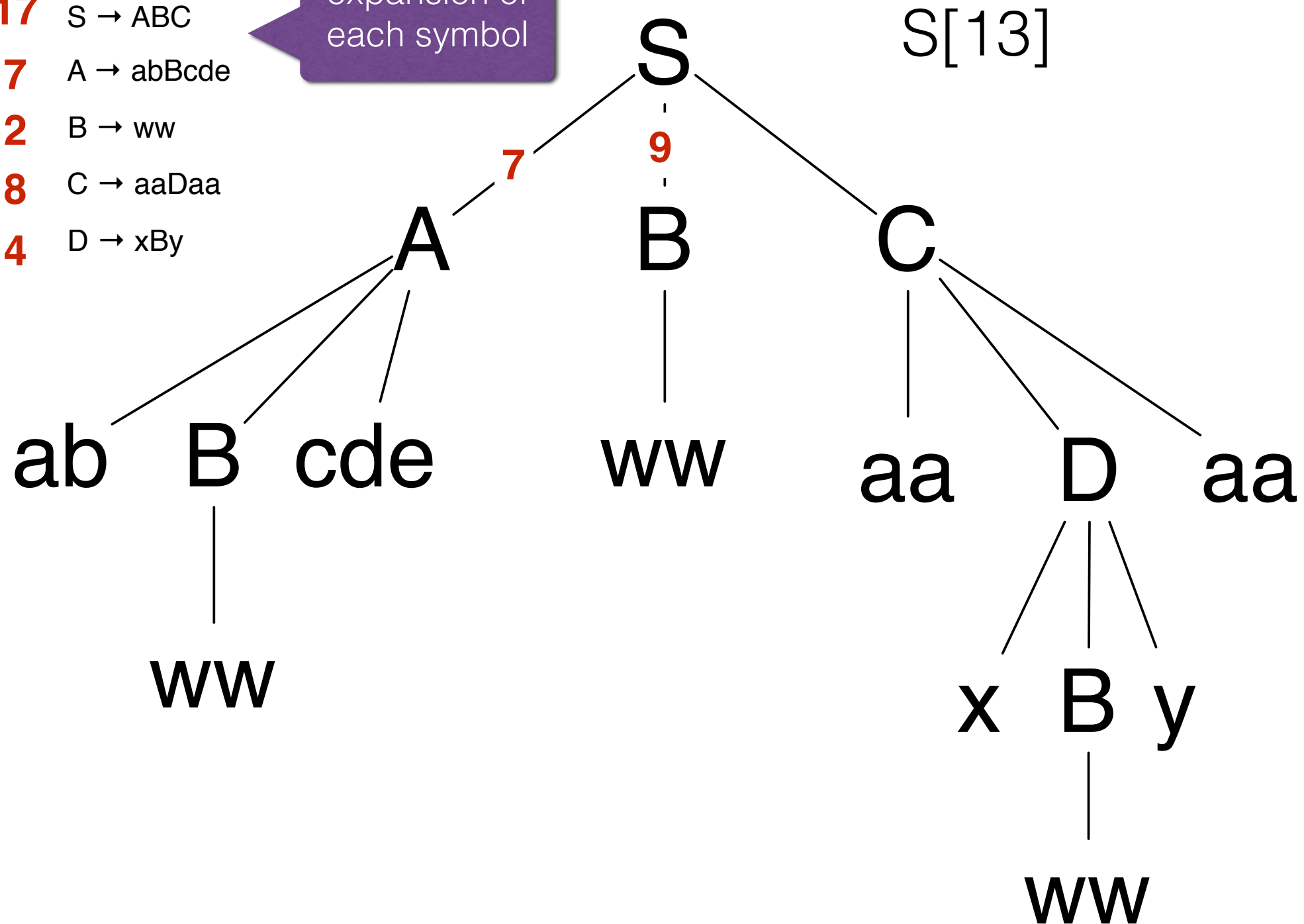
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

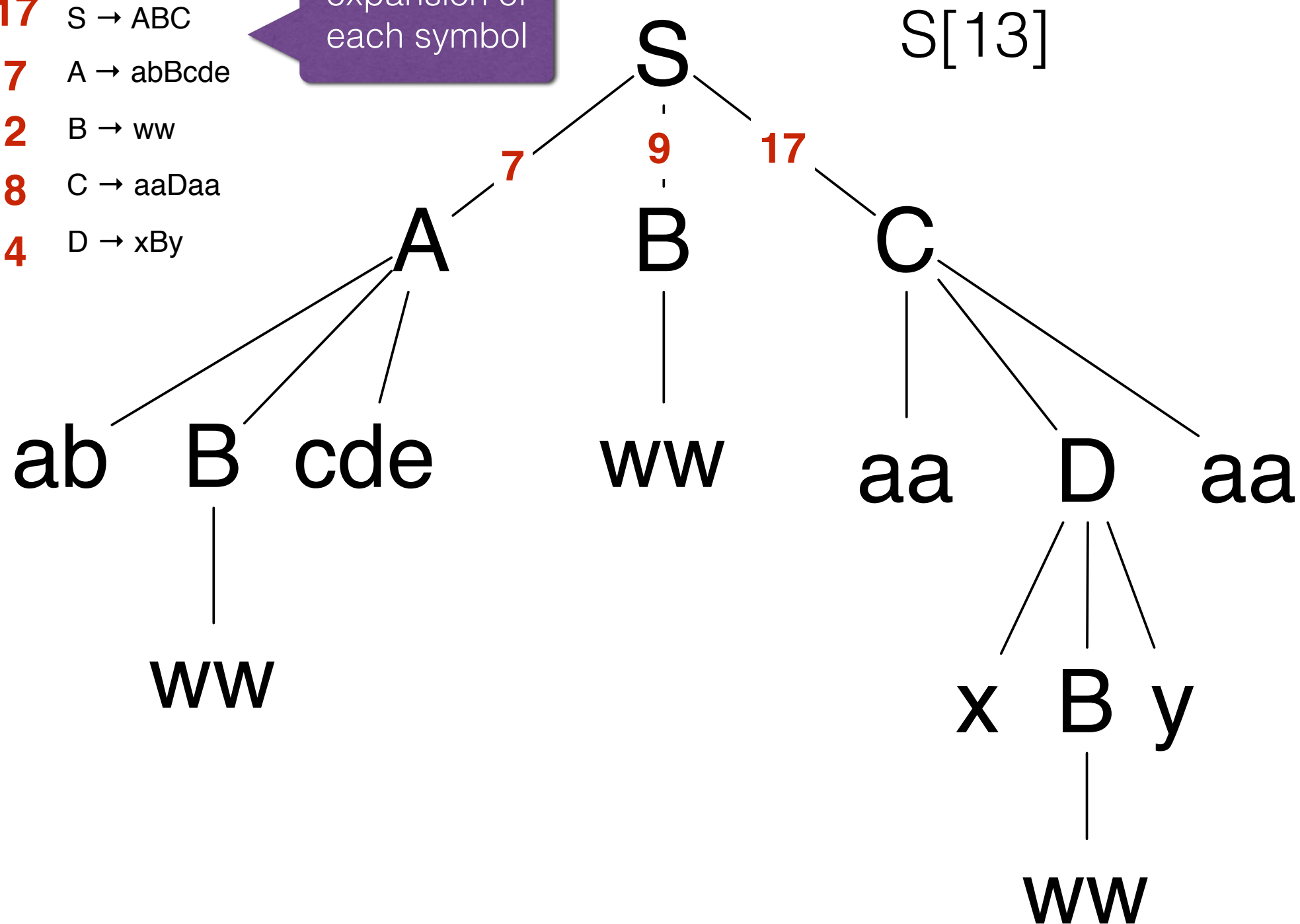
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

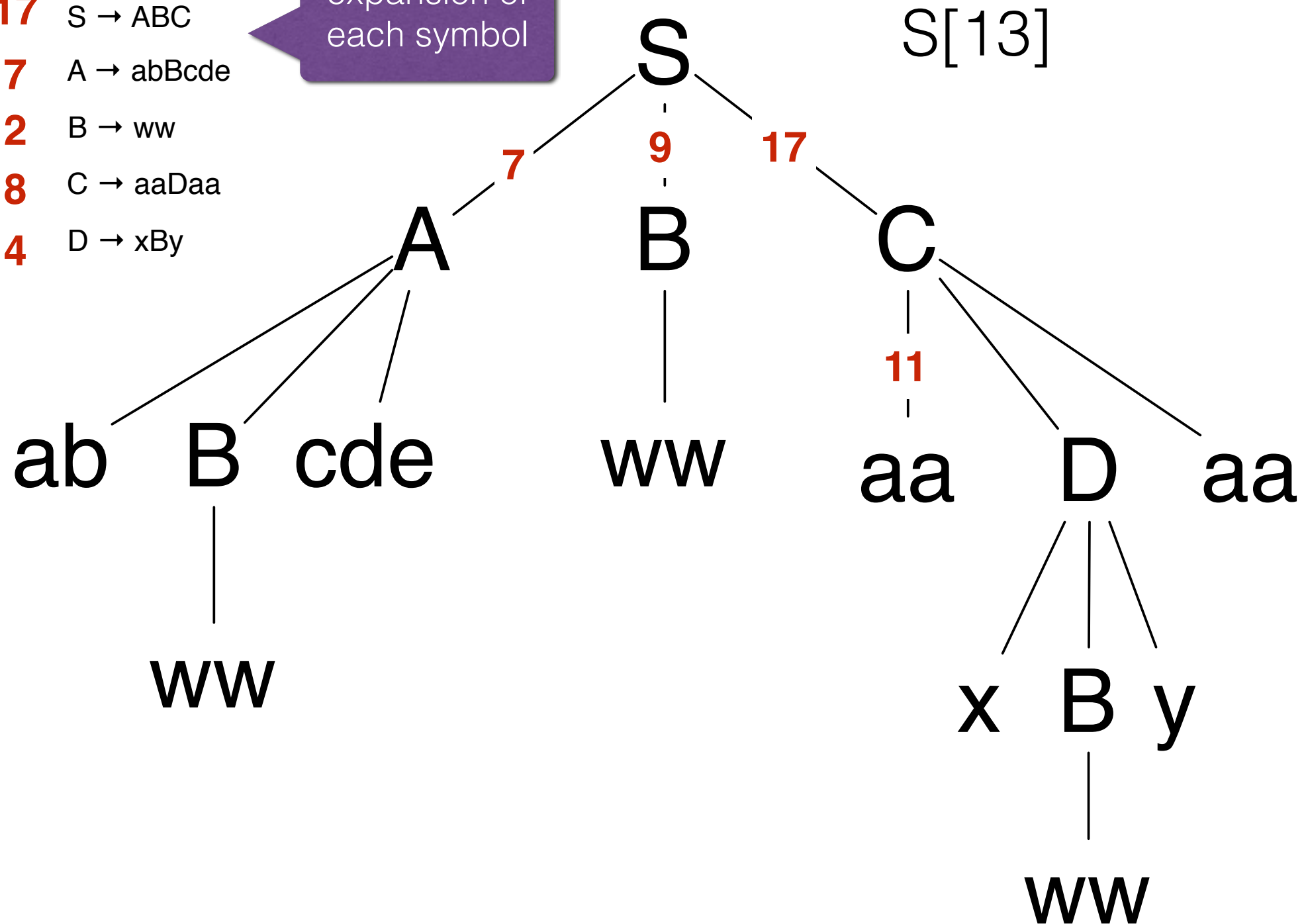
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

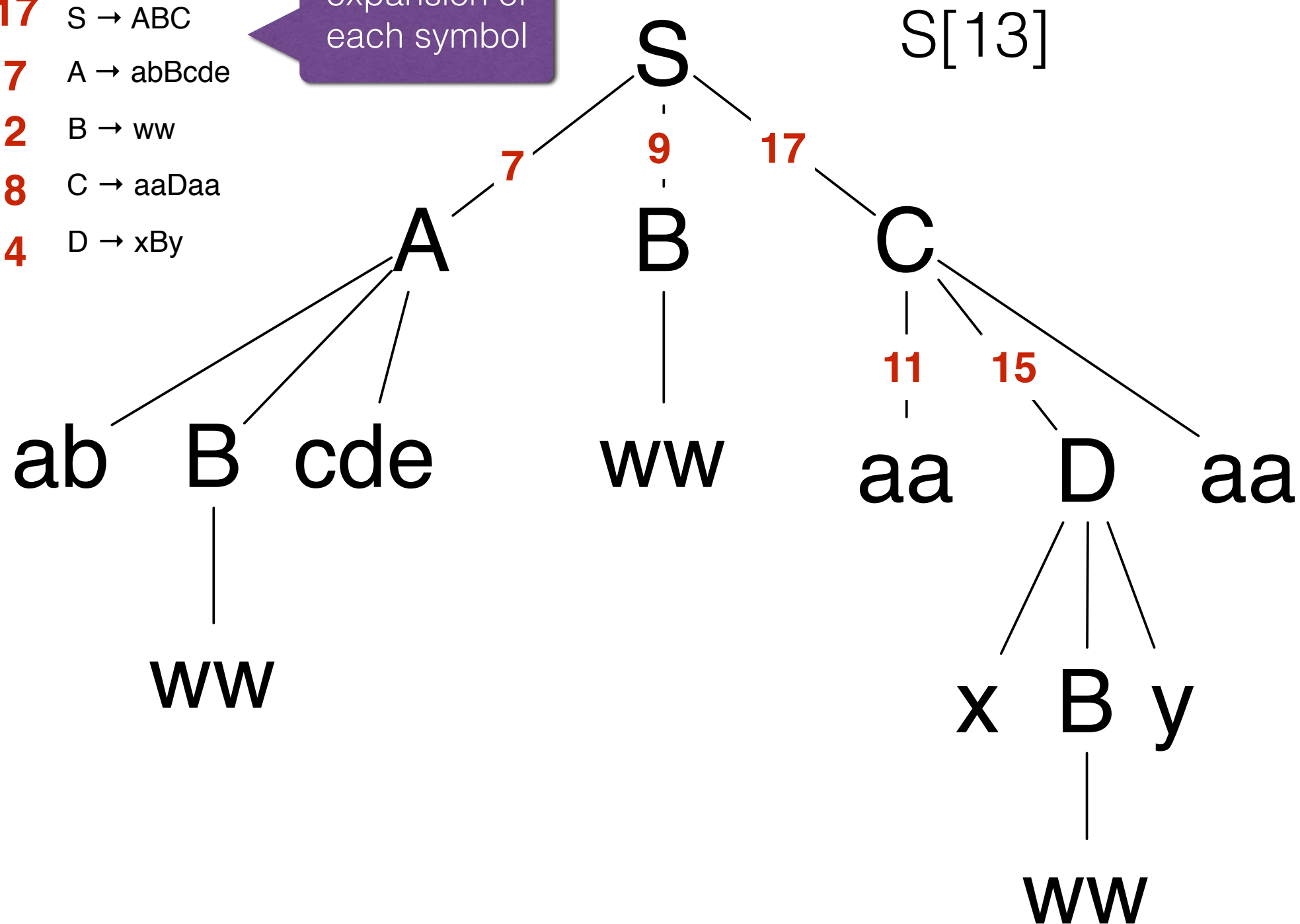
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

17 $S \rightarrow ABC$

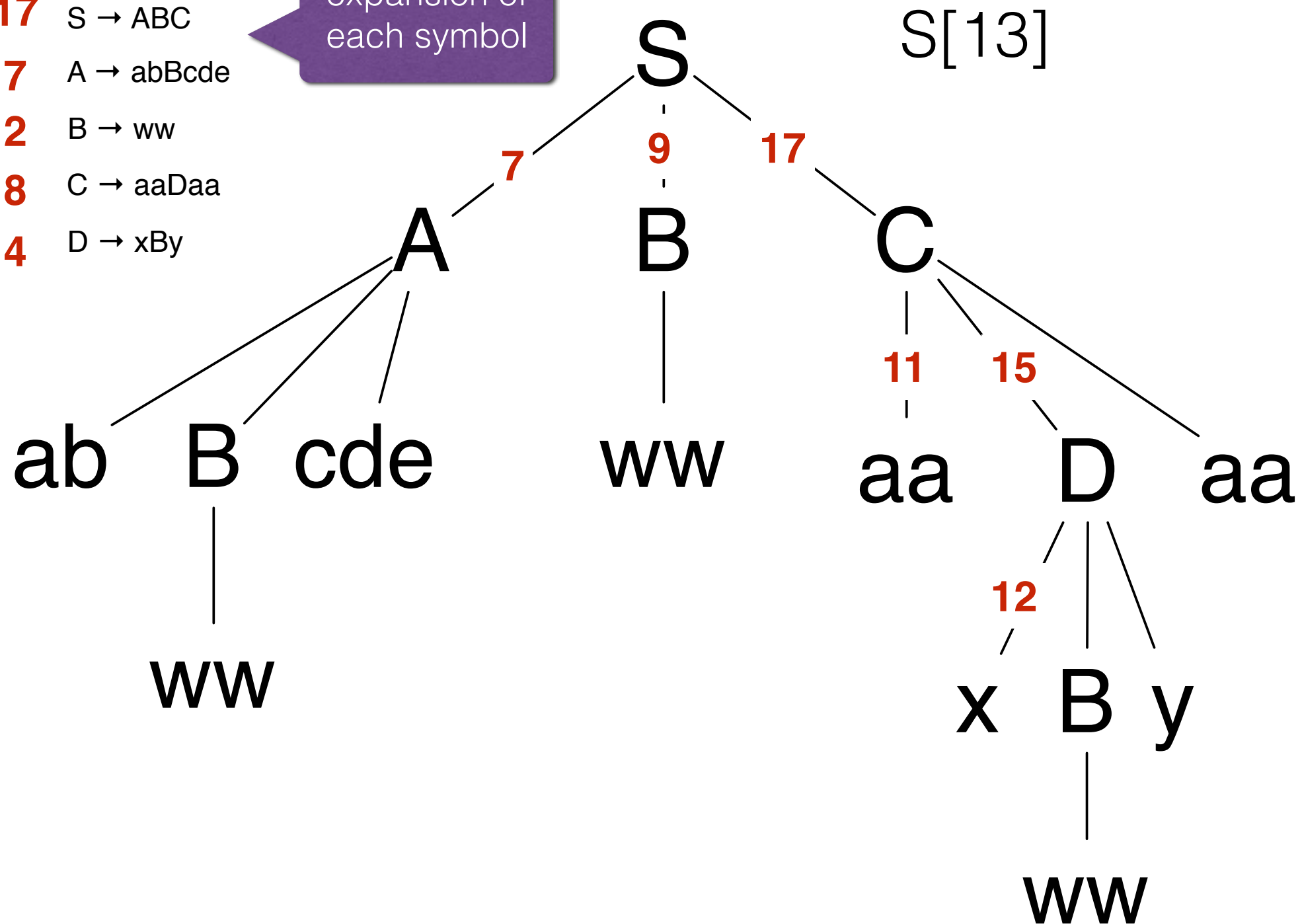
7 $A \rightarrow abBcde$

2 $B \rightarrow ww$

8 $C \rightarrow aaDaa$

4 $D \rightarrow xBy$

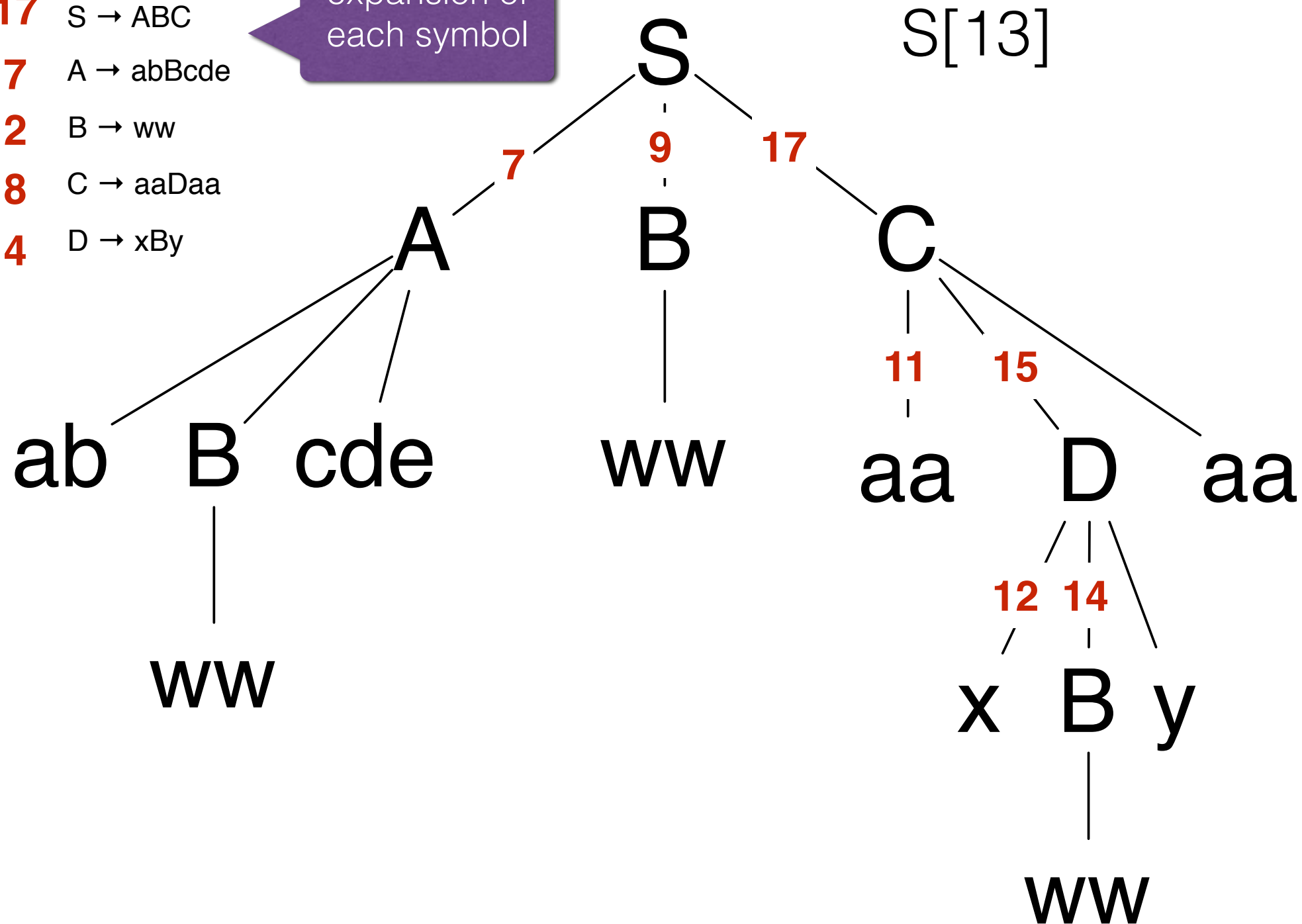
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17 $S \rightarrow ABC$
- 7 $A \rightarrow abBcde$
- 2 $B \rightarrow ww$
- 8 $C \rightarrow aaDaa$
- 4 $D \rightarrow xBy$

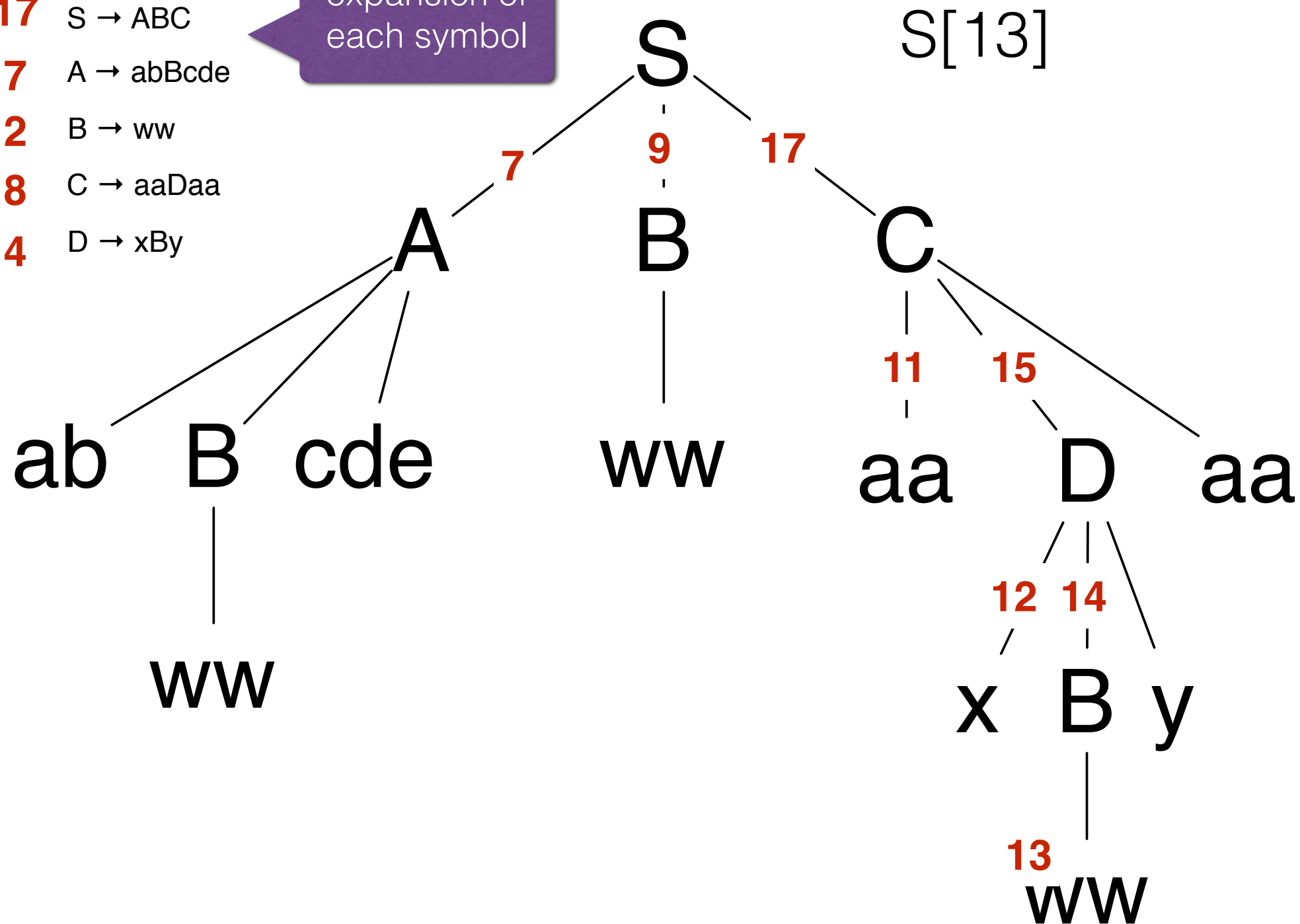
Store length of expansion of each symbol



Accessing $S[i]$ in Compressed Grammar

- 17** $S \rightarrow ABC$
- 7** $A \rightarrow abBcde$
- 2** $B \rightarrow ww$
- 8** $C \rightarrow aaDaa$
- 4** $D \rightarrow xBy$

Store length of expansion of each symbol



Re-PAIR Off-line Compression Algorithm

Larsson and Moffat, Off-Line Dictionary-Based Compression,
Proceedings of the IEEE, 88(11):1722-1732 (2000).

Re-Pair Algorithm Schema

1. Find the pair ab that occurs most frequently in the current message.

2. Replace all occurrences of ab with a new symbol A

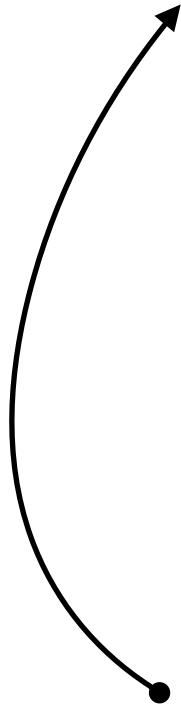
$A \rightarrow ab:$ $ababcab \rightarrow AAcA$
 $A \rightarrow aa:$ $aaaacaa \rightarrow AAcA$

3. Add the rule $A \rightarrow ab$ to the grammar.

4. Repeat until no pair occurs > 1 time.

5. Zero-order compress (e.g. Huffman) the resulting string

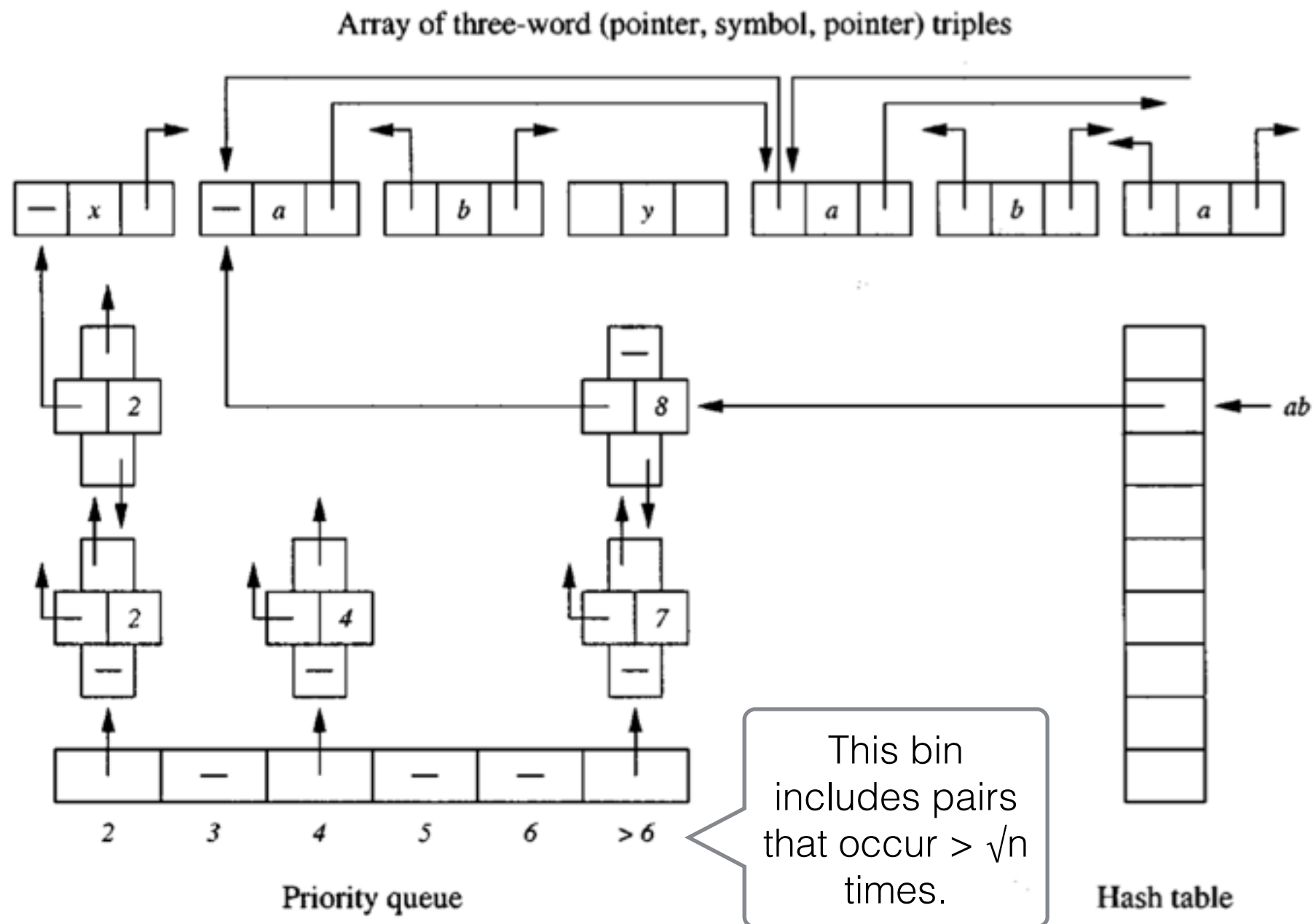
6. Encode and transmit the grammar



Example

Pair	String
	singing.do.wah.diddy.diddy.dum.diddy.do
A → .d	singingAo.wahAidddyAidddyAumAidddyAo
B → dd	singingAo.wahAiByAiByAumAiByAo
C → Ai	singingAo.wahCByCByAumCByAo
D → By	singingAo.wahCDCDAumCDAo
E → CD	singingAo.wahEEAumEAo
F → in	sFgFgAo.wahEEAumEAo
G → Ao	sFgFgG.wahEEAumEG
H → Fg	sHHG.wahEEAumEG

Implementation Details



(Larsson & Moffat)

Replace(ab, A):

1. Find next occurrence of $xaby$ (using hash and linked list of symbols)
2. Replace ab with A
3. Decrement counts of xa and by (moving entry lower in queue)
4. Increment counts of xA and Ay (moving entry higher in queue, creating them the first time)

Running Time

- **Finding the most frequent pair:**
 - walk down the last list in the priority queue in time $O(\sqrt{n})$ and find the most frequent pair. (Why is it $O(\sqrt{n})$ time to read the last list?)
 - that pair will result in at least $O(\sqrt{n})$ replacements. Why?
- Each operation of **Replace(ab,A)** takes $O(1)$ time, so each replace happens in constant time.
- For a sequence of length n there can be at most $O(n)$ replacements. Why?
- Total time to build the grammar = $O(n)$.

Running Time

Every item on the list occurs $\geq \sqrt{n}$ times, so there can be at most \sqrt{n} such times.

- **Finding the most frequent pair:**
 - walk down the last list in the priority queue in time $O(\sqrt{n})$ and find the most frequent pair. (Why is it $O(\sqrt{n})$ time to read the last list?)
 - that pair will result in at least $O(\sqrt{n})$ replacements. Why?
- Each operation of **Replace(ab,A)** takes $O(1)$ time, so each replace happens in constant time.
- For a sequence of length n there can be at most $O(n)$ replacements. Why?
- Total time to build the grammar = $O(n)$.

Running Time

Every item on the list occurs $\geq \sqrt{n}$ times, so there can be at most \sqrt{n} such times.

- **Finding the most frequent pair:**
 - walk down the last list in the priority queue in time $O(\sqrt{n})$ and find the most frequent pair. (Why is it $O(\sqrt{n})$ time to read the last list?)
 - that pair will result in at least $O(\sqrt{n})$ replacements. Why?
 - Each operation of **Replace(ab,A)** takes $O(1)$ time, so each replace happens in constant time.
 - For a sequence of length n there can be at most $O(n)$ replacements. Why?
- Each replacement reduces the length of the sequence by 1.
- Total time to build the grammar = $O(n)$.

Encoding the Grammar (Dictionary)

Divide symbols into *generations*:

0:  (input alphabet)

1: 

2: 

3: 

A symbol $A \rightarrow XY$ is in the lowest generation such that X and Y are in previous generations

$k_i := \#$ of symbols in generation $\leq i$

Can equate a symbol in generation i with a number between k_{i-1} and k_i-1

Encoding Dictionary, Idea

Need to output a sequence of pairs $(a_1, b_1), (a_2, b_2), \dots$

Note:

1. In generation j , the maximum value of any a_i or b_i is $\leq k_j$.
2. In generation j , if $a_i \leq k_{j-2}$ then $b_j \geq k_{j-2}$. Why?
3. We can order the pairs in a generation in lexicographical order

Encoding idea:

Changes slowly, so
use Δ -like encoding

m	a
m	b
m	g
n	m
n	n
o	c

Use rules 1 and 2 above to figure out the range of the second coordinate and use the minimum # of bits for that range.

Re-Pair Timing

<i>method</i>	<i>encoding</i>	<i>decoding</i>
<i>GZip -9</i>	26.0	1.5
<i>PPMD</i>	41.2	41.7
<i>Re-Pair</i>	135.7	3.1

(Larsson & Moffat)

Decompression time very fast: not quite as good as gzip, but much better than a context-based encoder.

Re-Pair Compression Performance

in bits per character

<u>file</u>	<u>chi.</u>	<u>stat.</u>	<u>tot.</u>	<u>GZip</u>	<u>PPMD</u>
<i>E.coli</i>	0.11	1.98	2.09	2.24	1.99
<i>bible.txt</i>	0.29	1.47	1.76	2.33	1.58
<i>world192.txt</i>	0.31	1.31	<u>1.62</u>	<u>2.33</u>	<u>1.52</u>
<i>average</i>			1.83	2.30	1.70
<i>WSJ20</i>	0.29	1.68	1.98	2.91	1.72

dictionary

string

(Larsson & Moffat)

Re-Pair Compression Performance

in bits per character Bigger than the simple 2-bit encoding!

<u>file</u>	<u>chi.</u>	<u>stat.</u>	<u>tot.</u>	<u>GZip</u>	<u>PPMD</u>
<i>E.coli</i>	0.11	1.98	2.09	2.24	1.99
<i>bible.txt</i>	0.29	1.47	1.76	2.33	1.58
<i>world192.txt</i>	0.31	1.31	1.62	2.33	1.52
<i>average</i>			1.83	2.30	1.70
<i>WSJ20</i>	0.29	1.68	1.98	2.91	1.72

dictionary ↗ ↖ string

(Larsson & Moffat)

Sequitur

Nevill-Manning & Witten, *The Computer Journal* 40 (1997),
103-116.

Sequitur Invariants

- Online algorithm: reads string from left to right, constructing a grammar, maintaining the following invariants at each step:
 - *Digram uniqueness*: no adjacent pair of symbols appears > 1 time in the grammar.
 - *Rule utility*: Every rule must be used at least twice.

Sequitur Algorithm

```
for i = 1...|S|:  
  append S[i] to rule S
```

Repeatedly replace digrams that occur $>$ once with their symbol

Repeatedly remove rules that occur only once.

Sequitur Example

(From Cherniavsky & Ladner, 2004)

$S = \text{acgtcgcgacgt}$

$S \rightarrow \underline{\text{acgtcgcg}}$ \rightarrow $S \rightarrow \text{aAtA}$
 $A \rightarrow \text{cg}$ \rightarrow $S \rightarrow \text{aAtAa}\underline{\text{cg}}$ \rightarrow $S \rightarrow \text{aAtAaA}$
 $A \rightarrow \text{cg}$

$S \rightarrow \underline{\text{aAtAaA}}$ \rightarrow $S \rightarrow \underline{\text{BtAB}}$
 $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$ \rightarrow $S \rightarrow \underline{\text{BtABt}}$ \rightarrow $S \rightarrow \text{CAC}$
 $B \rightarrow \text{aA}$ \rightarrow $B \rightarrow \text{aA}$ \rightarrow $A \rightarrow \text{cg}$
 $C \rightarrow \text{Bt}$

$S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$
 $C \rightarrow \text{aAt}$

Sequitur Example

(From Cherniavsky & Ladner, 2004)

$S = \text{acgtcgacgt}$

“cg” appears twice

$S \rightarrow \underline{\text{acgtcg}}$ \rightarrow $S \rightarrow \text{aAtA}$
 $A \rightarrow \text{cg}$ \rightarrow $S \rightarrow \text{aAtAa}\underline{\text{cg}}$ \rightarrow $S \rightarrow \text{aAtAaA}$
 $A \rightarrow \text{cg}$

$S \rightarrow \underline{\text{aAtAaA}}$ \rightarrow $S \rightarrow \underline{\text{BtAB}}$ \rightarrow $S \rightarrow \underline{\text{BtABt}}$ \rightarrow $S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$
 $B \rightarrow \text{aA}$ \rightarrow $B \rightarrow \text{aA}$ \rightarrow $B \rightarrow \text{aA}$
 $C \rightarrow \text{Bt}$

$S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$
 $C \rightarrow \text{aAt}$

Sequitur Example

(From Cherniavsky & Ladner, 2004)

$S = \text{acgtcgacgt}$

“cg” appears twice

$S \rightarrow \underline{\text{acgtcg}}$ \rightarrow $S \rightarrow \text{aAtA}$
 $A \rightarrow \text{cg}$ \rightarrow $S \rightarrow \text{aAtAa}\underline{\text{cg}}$ \rightarrow $S \rightarrow \text{aAtAaA}$
 $A \rightarrow \text{cg}$

“aA” appears twice

$S \rightarrow \underline{\text{aAtAaA}}$ \rightarrow $S \rightarrow \underline{\text{BtAB}}$ \rightarrow $S \rightarrow \underline{\text{BtABt}}$ \rightarrow $S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$ \rightarrow $A \rightarrow \text{cg}$
 $B \rightarrow \text{aA}$ \rightarrow $B \rightarrow \text{aA}$ \rightarrow $B \rightarrow \text{aA}$
 $C \rightarrow \text{Bt}$

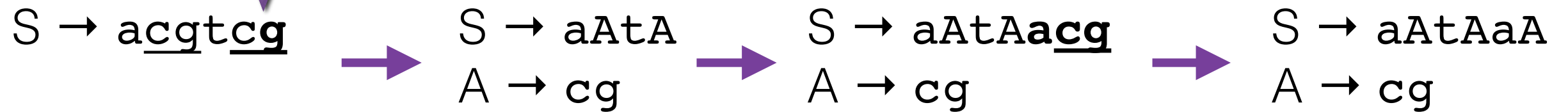
$S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$
 $C \rightarrow \text{aAt}$

Sequitur Example

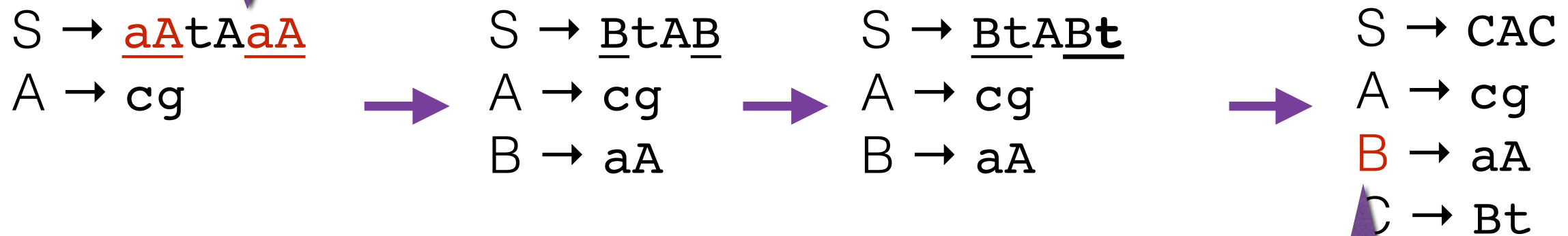
(From Cherniavsky & Ladner, 2004)

$S = \text{acgtcgacgt}$

"cg" appears twice



"aA" appears twice



$S \rightarrow \text{CAC}$
 $A \rightarrow \text{cg}$
 $C \rightarrow \text{aAt}$

B appears only once on right hand sides

Encoding the Grammar

Rule S is transmitted left to right, with the following rules to handle non-terminals (NT):

- The **first** time a NT is encountered, it's right-hand side is transmitted.

Its RHS is transmitted using these same rules

- The **second** time a NT is encountered, the pair (i, len) is transmitted that gives an index into S and length that form the righthand side of the NT.
 - At this point, the decoder stores $j \rightarrow S[i...i+len]$ as a rule
 - j is the next NT number.
- The **third** time a NT is encountered, a single number (j) is transmitted referring to the rule created before.

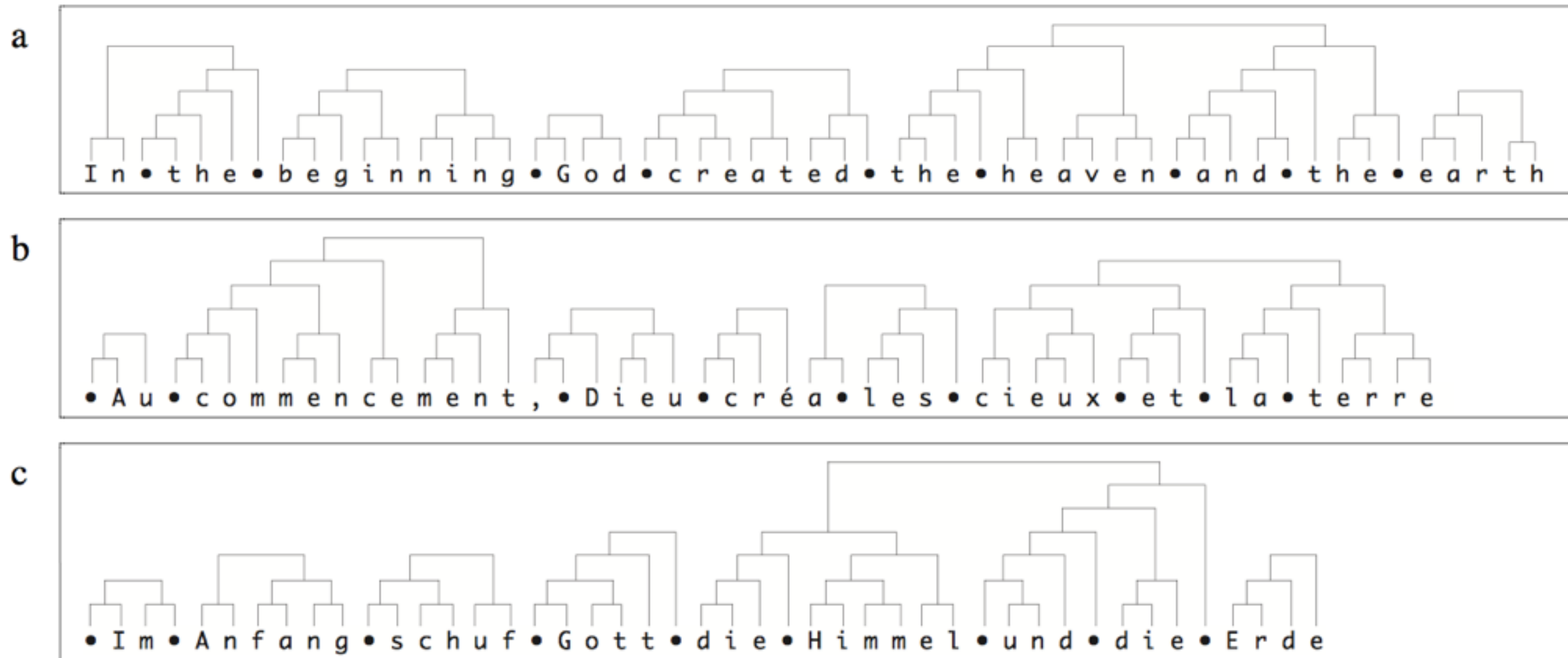
Compression Performance

name	description	size	compress	gzip	SEQUITUR	PPMC
bib	bibliography	111261	3.35	2.51	2.48	2.12
book1	fiction book	768771	3.46	3.25	2.82	2.52
book2	non-fiction book	610856	3.28	2.70	2.46	2.28
geo	geophysical data	102400	6.08	5.34	4.74	5.01
news	USENET	377109	3.86	3.06	2.85	2.77
obj1	object code	21504	5.23	3.84	3.88	3.68
obj2	object code	246814	4.17	2.63	2.68	2.59
paper1	technical paper	53161	3.77	2.79	2.89	2.48
paper2	technical paper	82199	3.52	2.89	2.87	2.46
pic	bilevel image	513216	0.97	0.82	0.90	0.98
progc	C program	39611	3.87	2.68	2.83	2.49
progl	Lisp program	71646	3.03	1.80	1.95	1.87
progp	Pascal program	49379	3.11	1.81	1.87	1.82
trans	shell transcript	93695	3.27	1.61	1.69	1.75
average			3.64	2.69	2.64	2.49
L-systems		908670	0.38	0.07	0.01	0.32
amino acids		1586289	4.52	4.08	3.28	3.54
Bible	King James version	4047392	2.77	2.32	1.84	1.92

Table 1 Performance of various compression schemes (bits per character)

(Nevill-Manning & Witten)

Grammars Useful for More Than Compression



(Nevill-Manning & Witten)

Figure 4 Hierarchies for Genesis 1:1 in (a) English, (b) French, and (c) German

DNASequitur

Cherniavsky and Ladner, Grammar-based Compression of DNA Sequences, UW CSE Tech Report, 2004

Applying Sequitur to DNA

- Reverse complements accounted for: when xy seen, **RC**(xy) is implicitly seen.
- Several other ideas implemented as well.

Table 3: Comparison of symbol streams (bits/symbol), best in bold

Sequence	Sequitur	Marker	LZ77-style	bzip2	arith	DNACompress
HUMDYSTROP	2.34	2.20	2.20	2.18	1.95	1.91
HUMGHCSA	1.86	1.74	1.77	1.73	2.00	1.03
MIPACGA	2.16	2.10	2.10	2.12	1.88	1.86
MPOCPCG	2.13	2.07	2.07	2.12	1.87	1.67
VACCG	2.11	2.06	2.06	2.09	1.92	1.76


DNA Sequitur

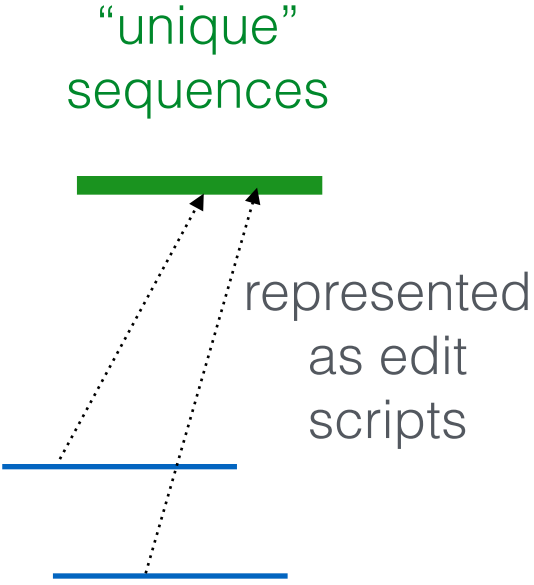
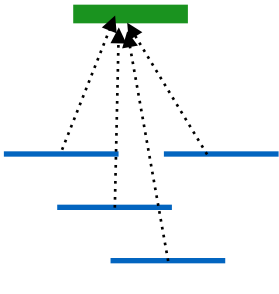
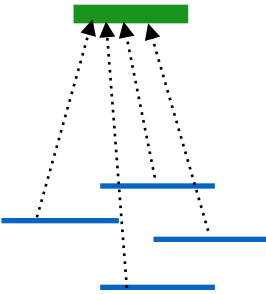
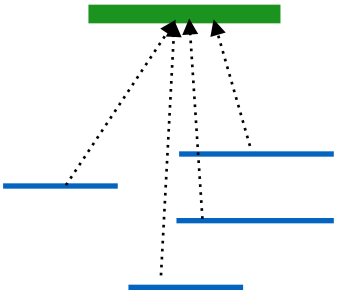
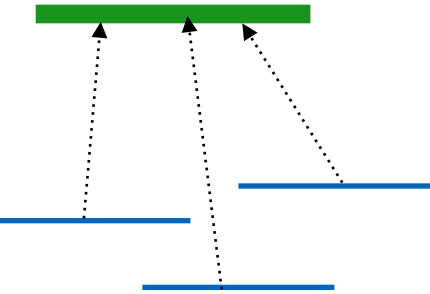
- Baseline = 2 bits / symbol
- Grammar-based methods do not compress the file in these tests.

"Compressive Genomics"

Po-Ru Loh, Michael Baym, Bonnie Berger. Compressive genomics allows computational analysis to keep pace with genomic data.

Nature Biotechnology 30(7):627-630, 2012.

CaBLAST

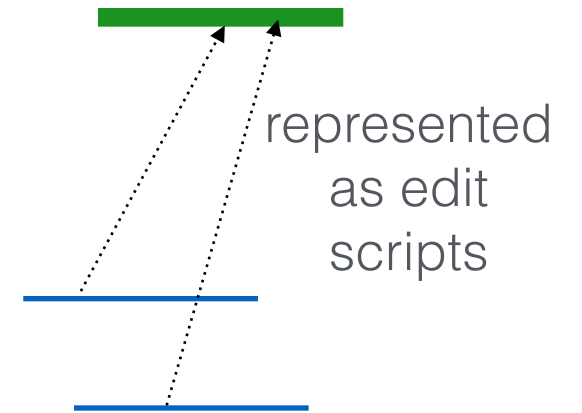
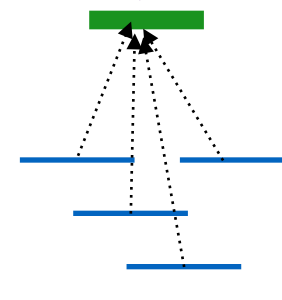
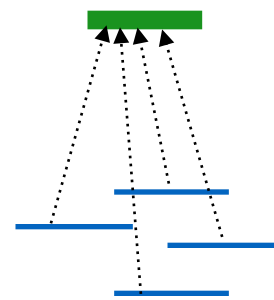
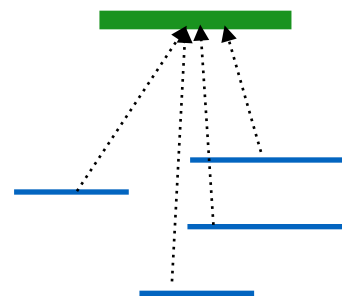
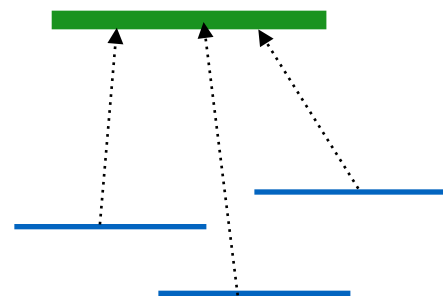


sequences similar to the unique sequence

CaBLAST

10-mer table to seed alignments

10-mer	Posn
	• •
	•



sequences similar to the unique sequence

“unique” sequences

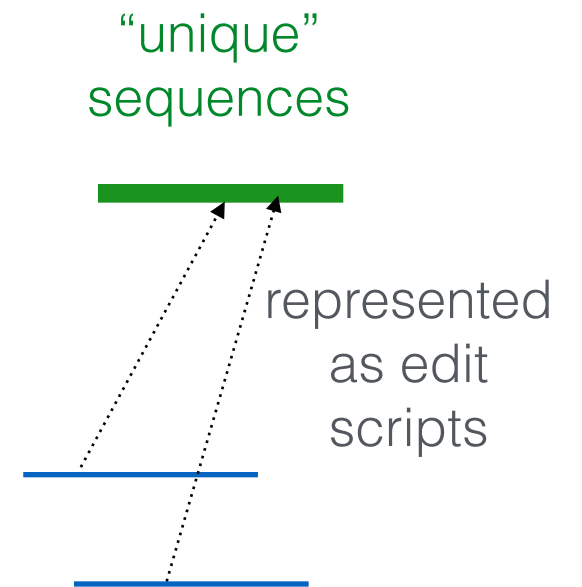
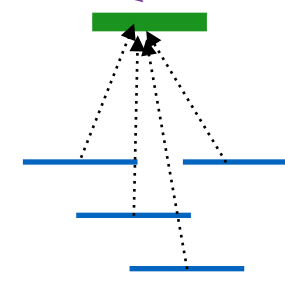
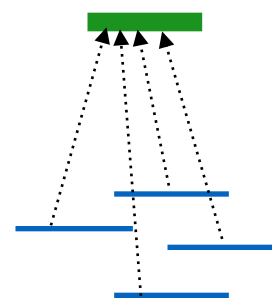
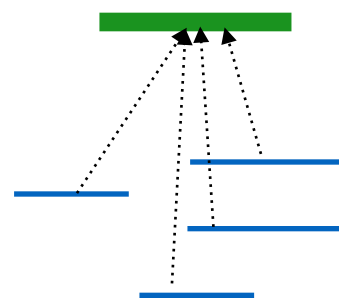
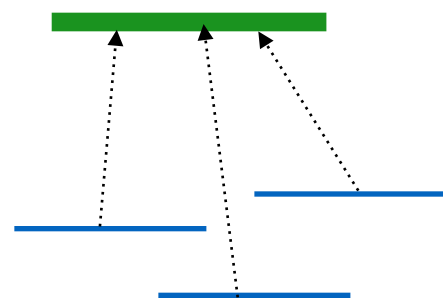
CaBLAST

Search: use BLAST to search **query** against **unique** sequences (use a liberal cutoff for a “match”)

for every hit of sufficient quality, expand the sequences contained in its bin and search them.

10-mer table to seed alignments

10-mer	Posn
	• •
	•

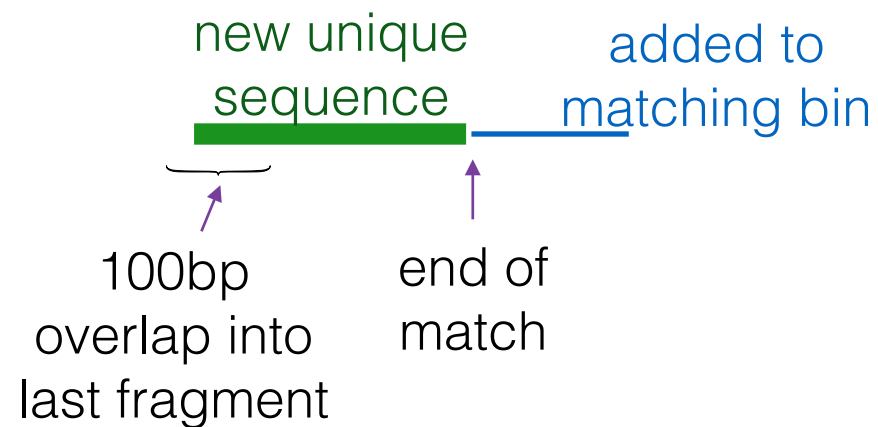


sequences similar to the unique sequence

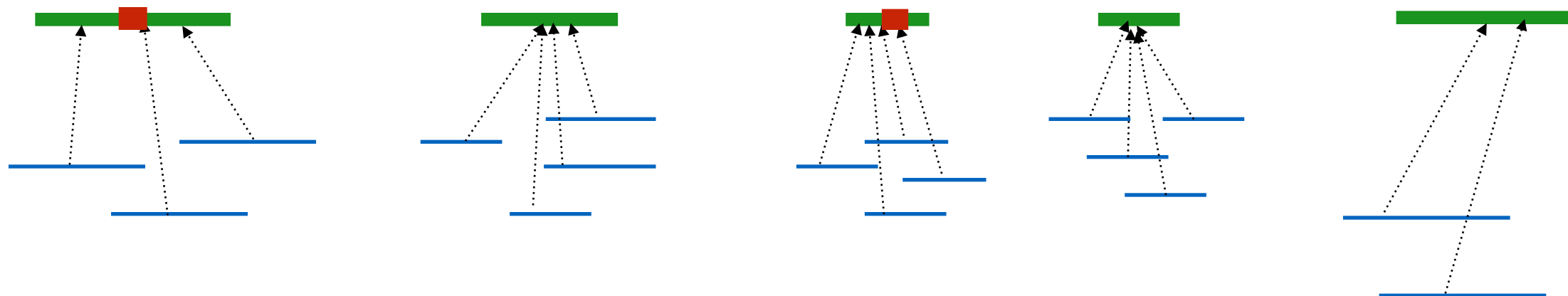
Building the CaBLAST database



Use **10-mer** at current position to find unique sequences to search
If any unique sequence contains a match of ≥ 300 ,
add the sequence between the two pointers to the database as follows:



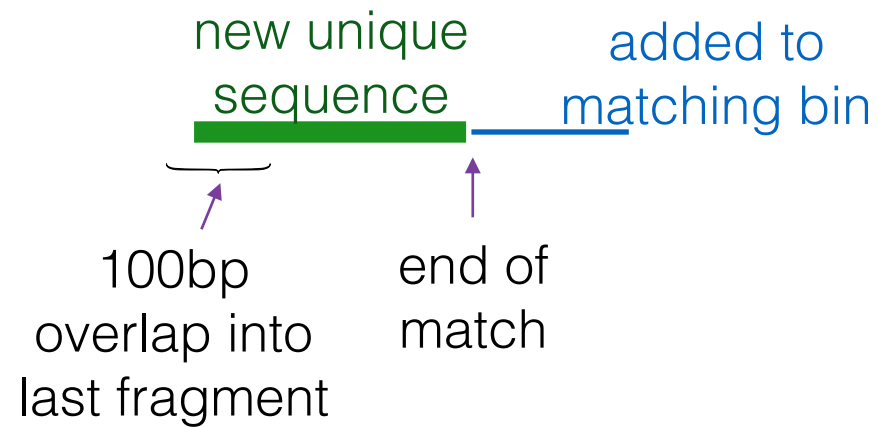
If 10,000 bases go by with no match, create a new 10,000 base unique sequence bin.



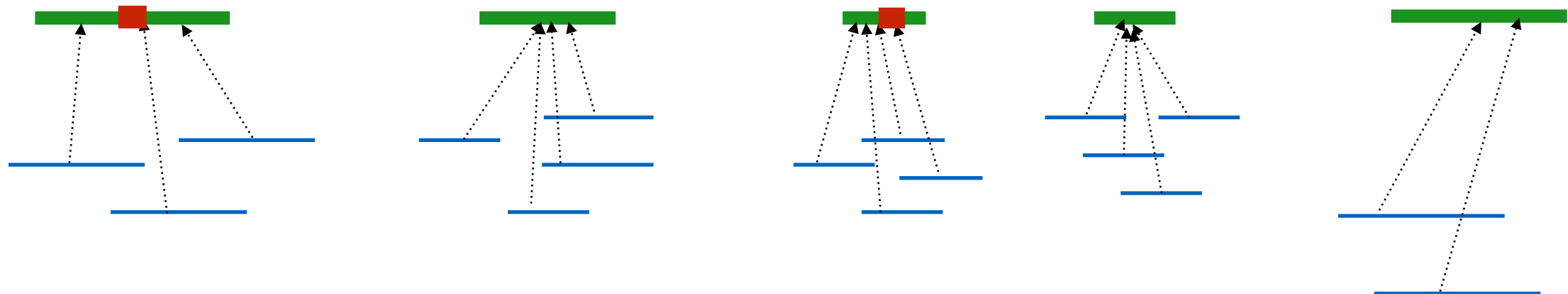
Building the CaBLAST database



Use **10-mer** at current position to find unique sequences to search
If any unique sequence contains a match of ≥ 300 ,
add the sequence between the two pointers to the database as follows:



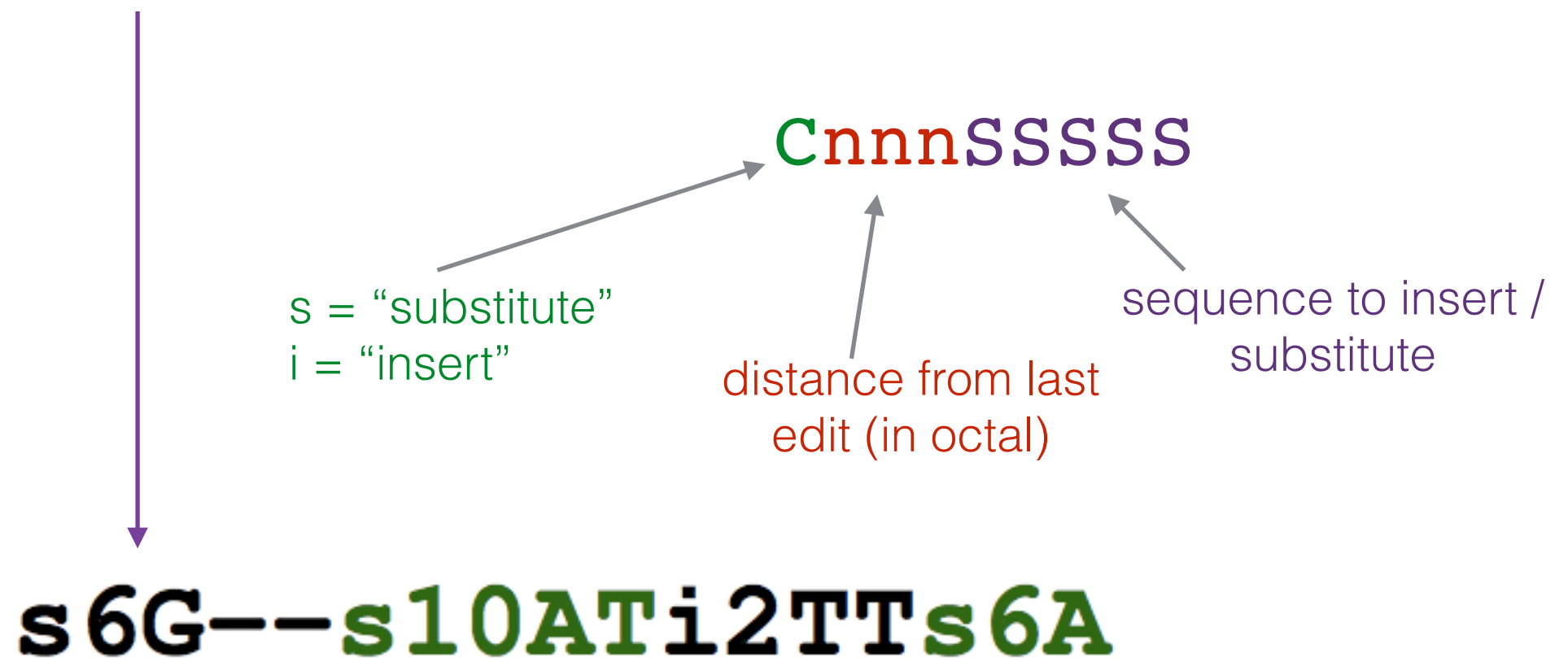
If 10,000 bases go by with no match, create a new 10,000 base unique sequence bin.



Storing Edit Scripts

GTTCACTTATGTATTC--ATATGATTTTGGCAA

GTTCACG--TGTATATTATATAATTTTGGCAA



Deletions are substitutions with "-".

There are 16 possible characters: s,i,A,C,G,T,N,-,0-7 → 4-bit encoding

CaBLAST Compression

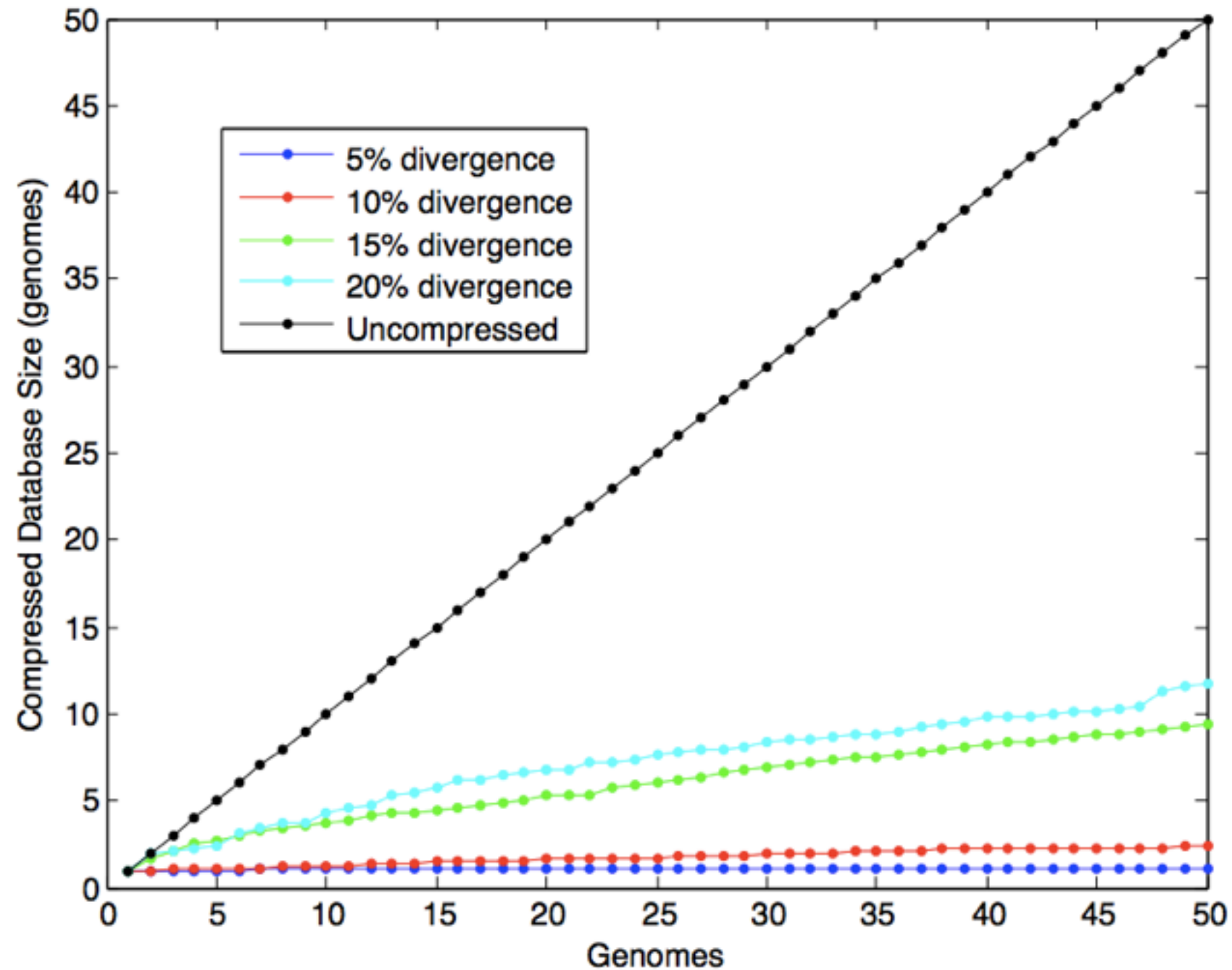


Figure S4: Performance of the Compressive BLAST preprocessing phase on simulated genera. Databases consist of sets of 50 simulated genomes (at 5%, 10%, 15%, and 20% divergence) generated with INDELible v1.03 [3].

(Loh et al, 2012)

CaBLAST Search Time

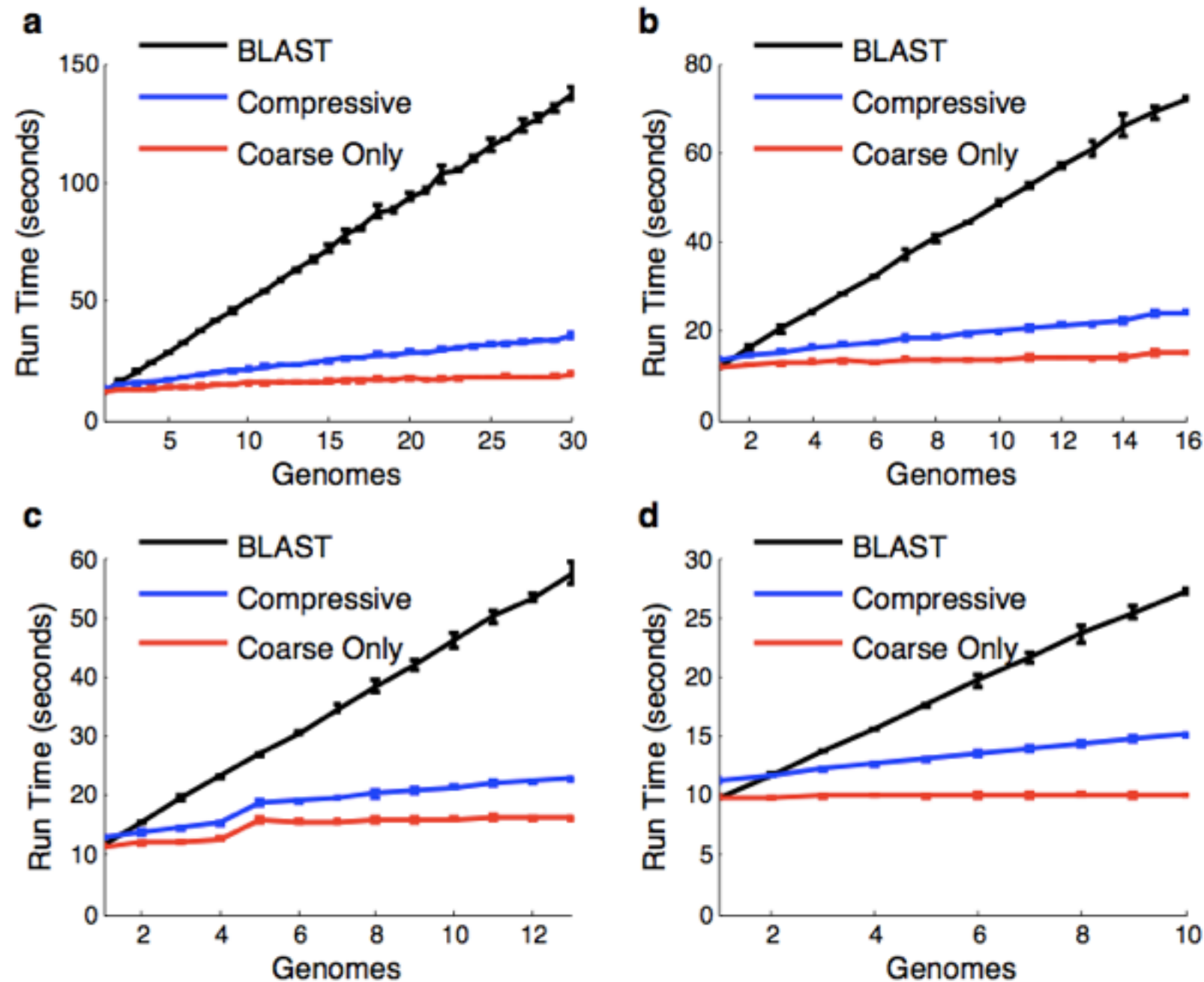
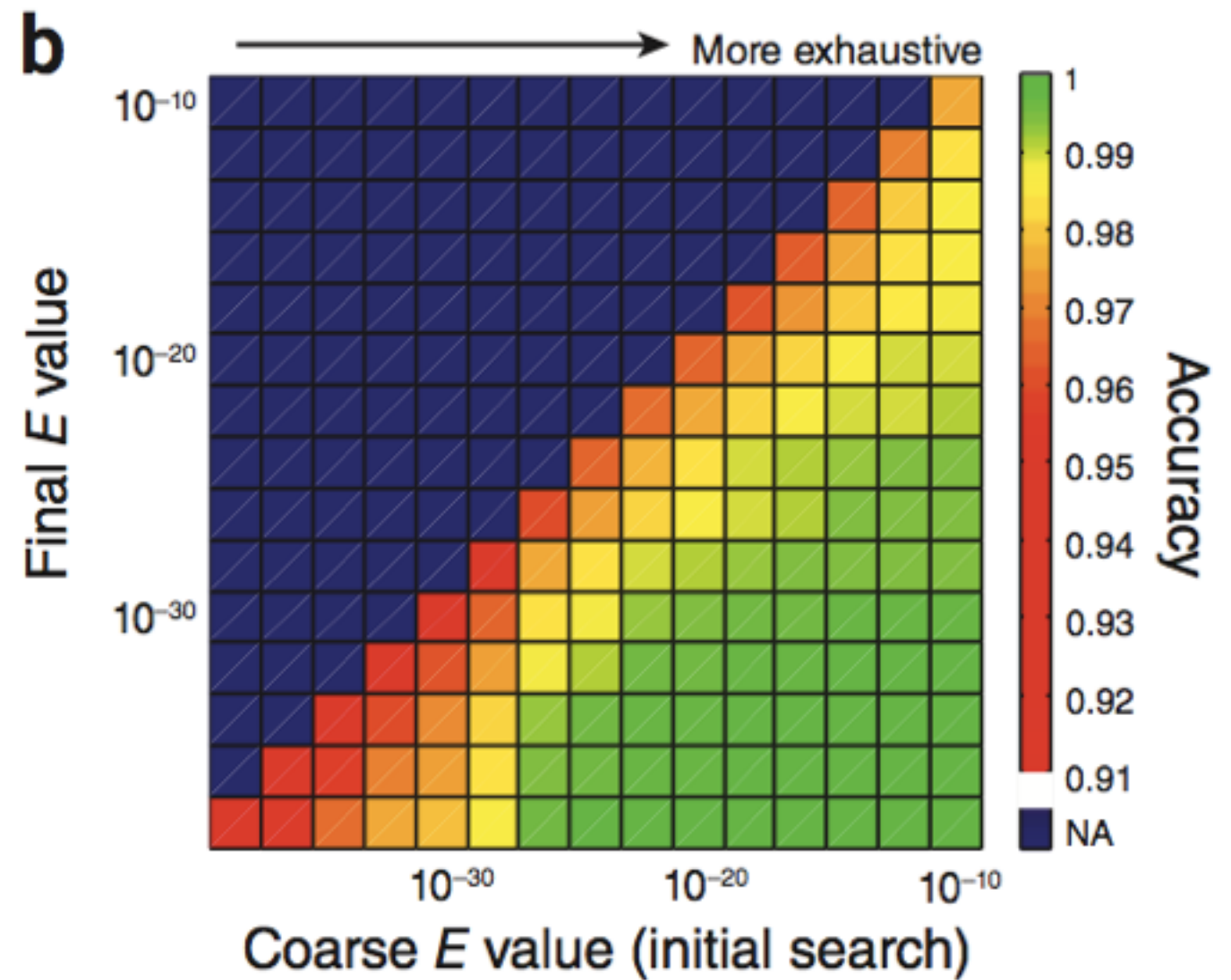
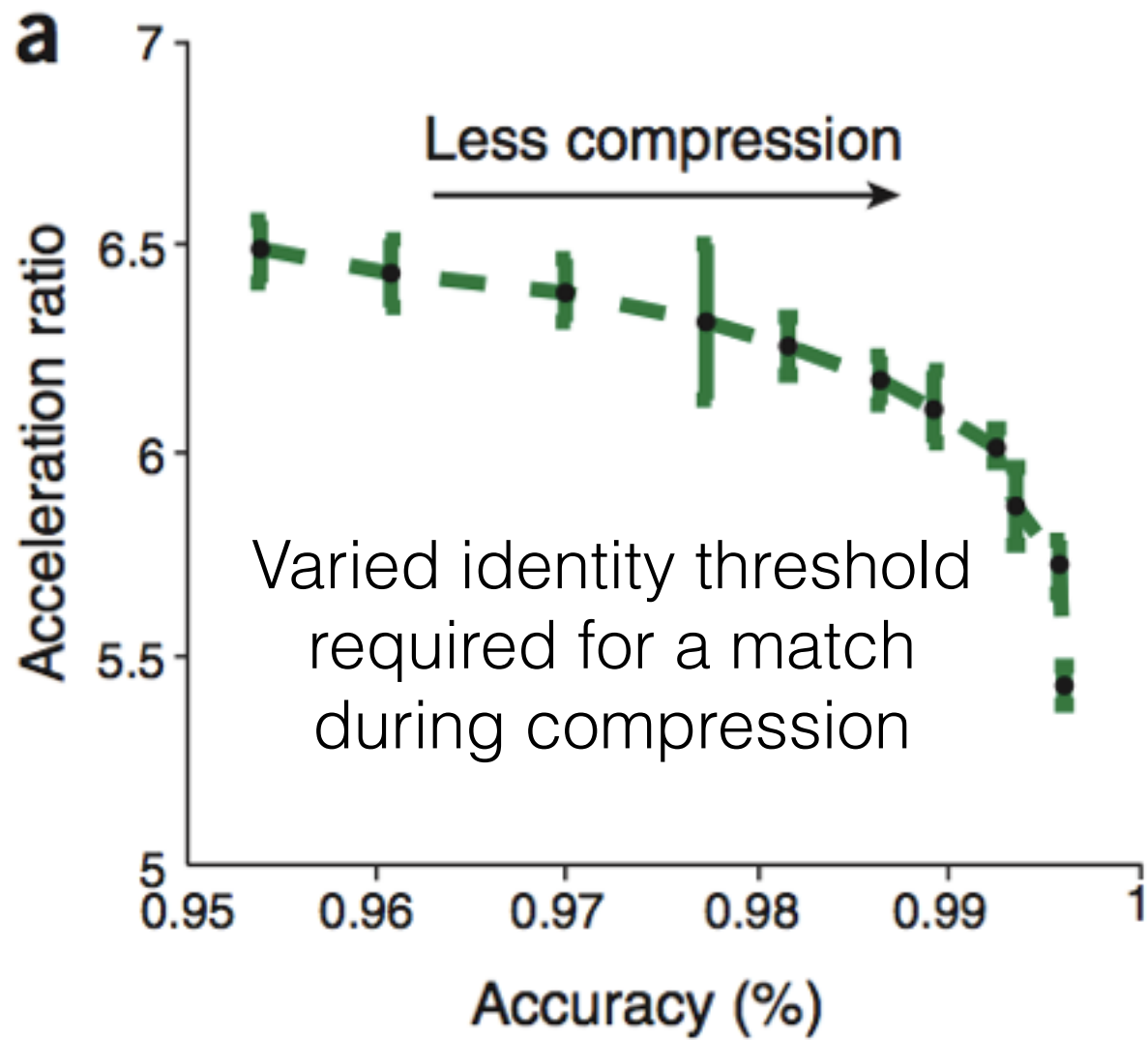


Figure S2: Performance of Compressive BLAST on databases of four bacterial genera using a single search set derived from the combined library of bacterial and yeast sequences. Parameters are the same (default) as in the primary manuscript. (a) Escherichia; (b) Salmonella; (c) Yersinia; (d) Brucella.

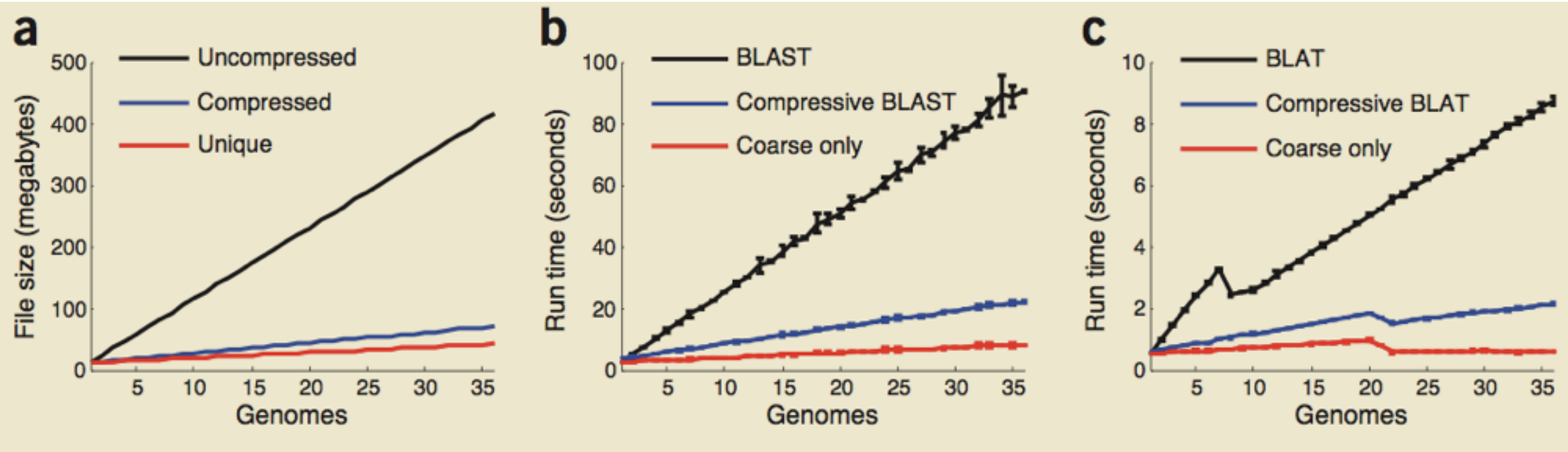
(Loh et al, 2012)

CaBLAST Accuracy



(Loh et al, 2012)

More CaBLAST Performance



Database of 36 yeast genomes.

(Loh et al, 2012)