

Compression

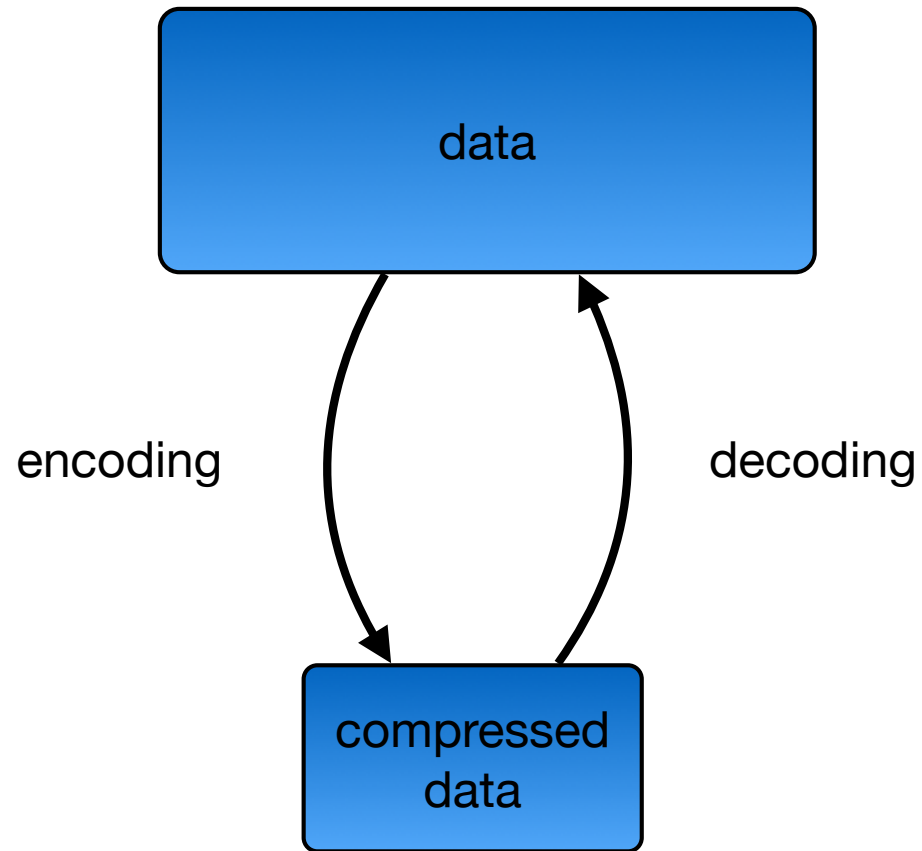
- Compression
- Lempel-Ziv
- Re-Pair and Grammars

Compression

- Compression
- Lempel-Ziv
- Re-Pair and Grammars

Compression

- Encoding and decoding.
- Lossless and lossy
- Compressed computation.



Compression

- [Statistical compression.](#)
 - Huffman, arithmetic encoding, Burrows-Wheeler, PPM, ...
- [Dictionary compression.](#)
 - Lempel-Ziv 77, Lempel-Ziv 78, Lempel-Ziv-Welch, ...
- [Grammar based schemes.](#)
 - Re-Pair, sequitur, greedy, bisection, ...
- [Kolmogorov compression.](#)
 - Ultimate compression scheme.
- [Transformation techniques.](#)
 - Differencing, Burrows-Wheeler, run-length encoding, Fourier transform, ...

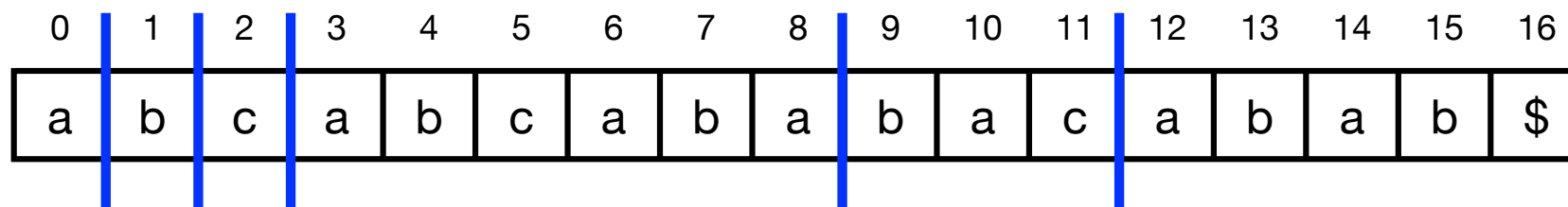
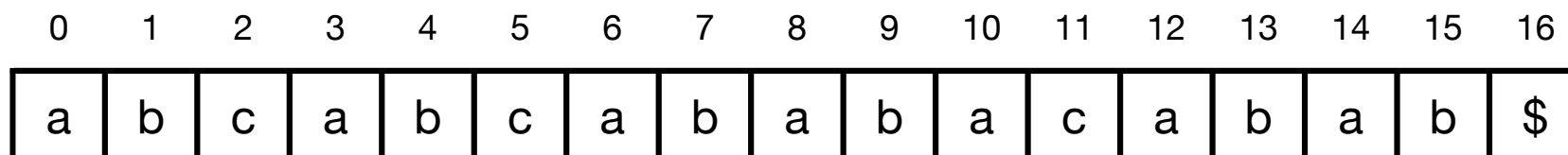
Compression

- Compression
- Lempel-Ziv
- Re-Pair and Grammars

Lempel-Ziv 77

- Encoding.

- Parse from left-to-right into **phrases**.
- Select longest matching substring starting before current position + 1 character.
- Encode phrases by (previous occ dist, length, extra character) or single character.



a b c

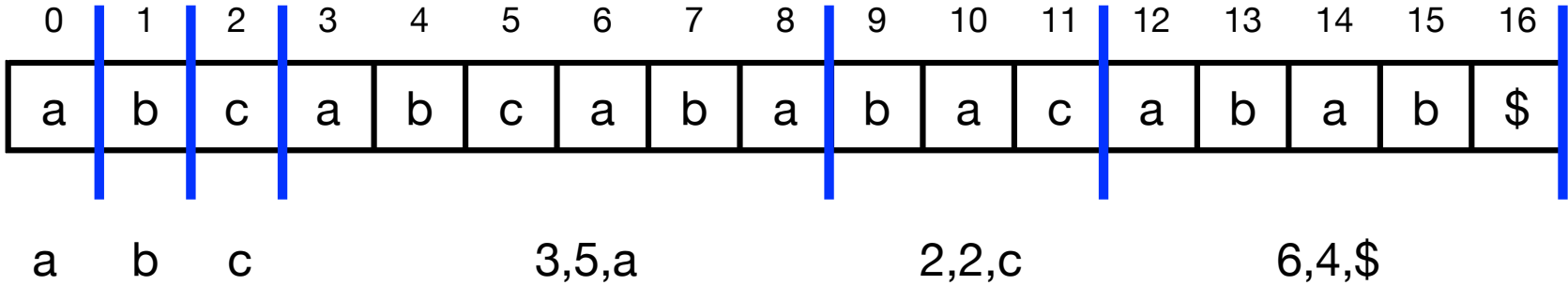
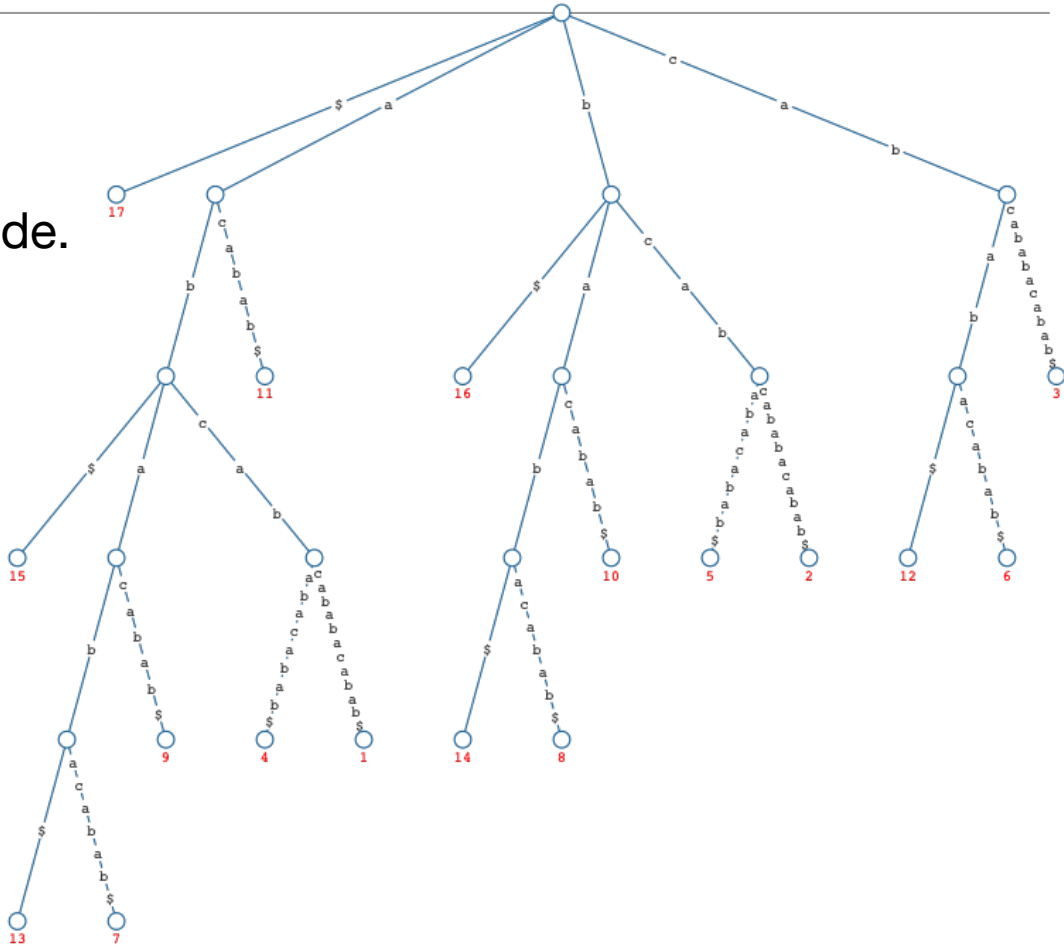
3,5,a

2,2,c

6,4,\$

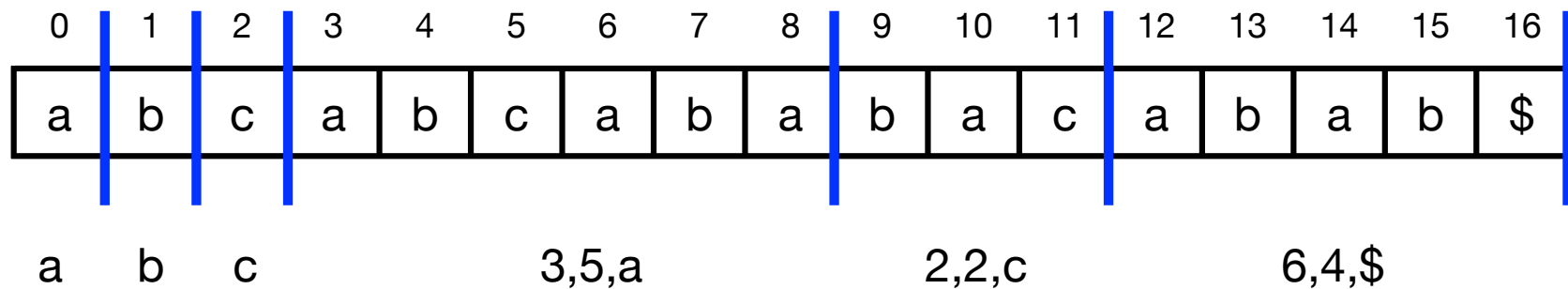
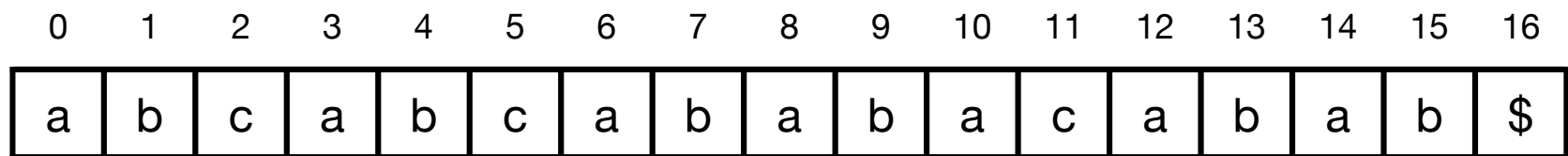
Lempel-Ziv 77

- **Encoding.**
 - Build suffix tree
 - Store smallest leaf below each node.
 - Greedy left-to-right parse.
- **Time.** $O(\text{sort}(n, |\Sigma|))$.



Lempel-Ziv 77

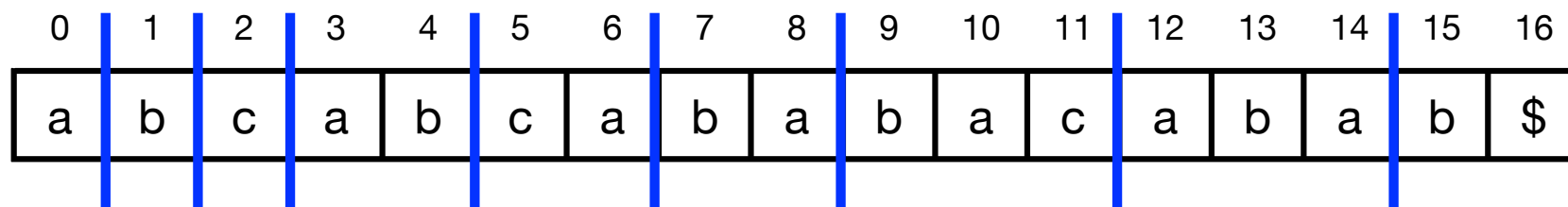
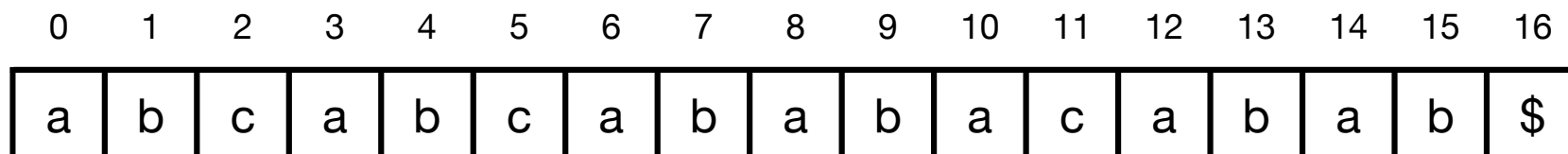
- **Decoding.** Read and decode left-to-right.
- **Time.** $O(n)$



Lempel-Ziv 78

- Encoding.

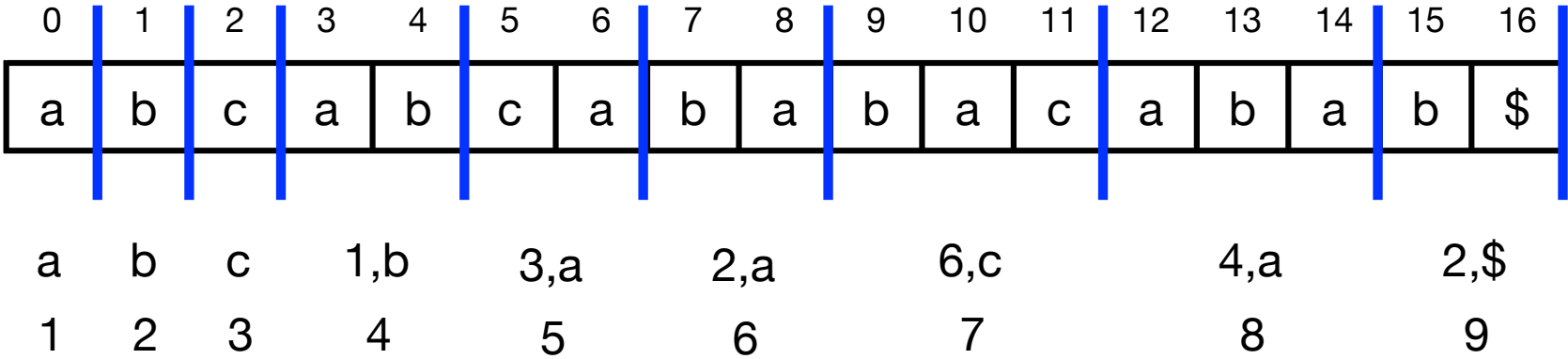
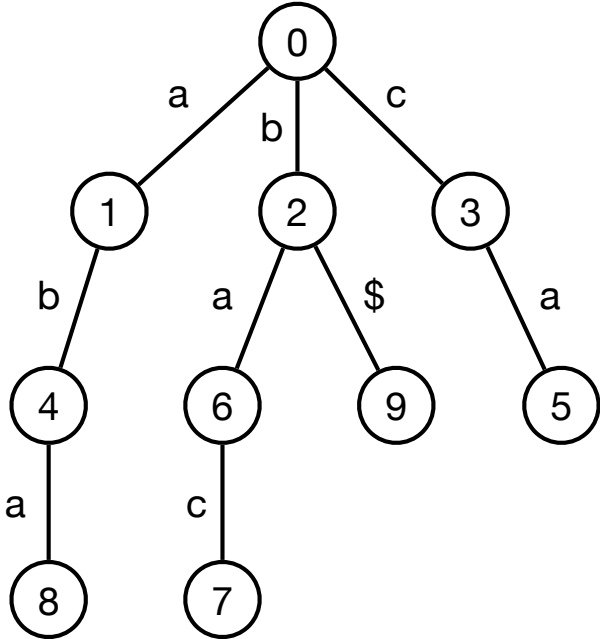
- Parse from left-to-right into phrases.
- Select longest **phrase** seen before + a single character.
- Encode phrases (previous phrase, character) or single phrase



a	b	c	1,b	3,a	2,a	6,c	4,a	2,\$
1	2	3	4	5	6	7	8	9

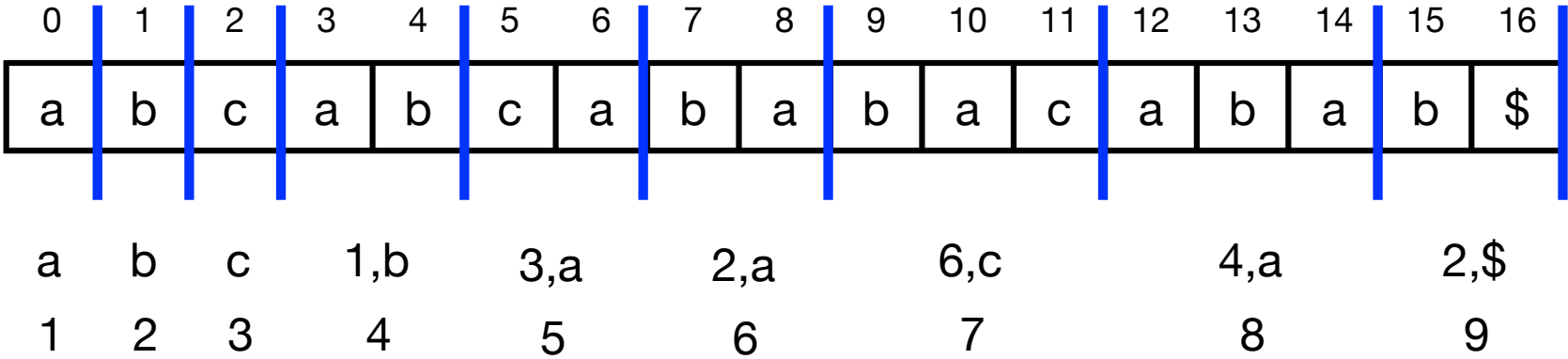
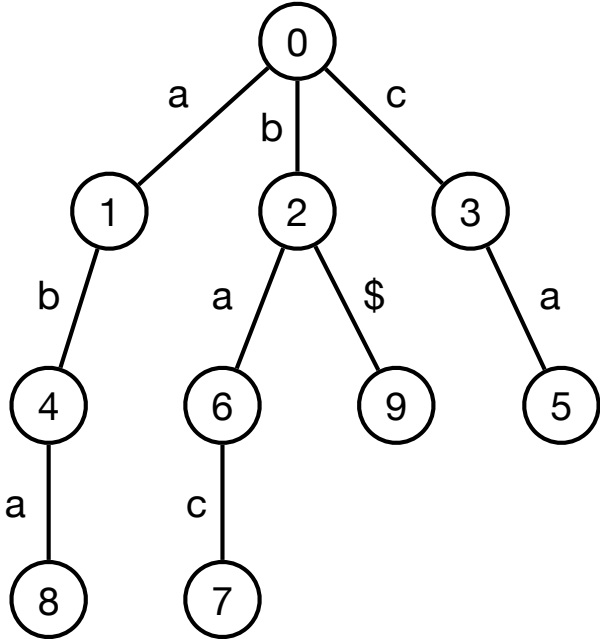
Lempel-Ziv 78

- **Encoding.**
 - Dynamically build and traverse the **LZ78 trie**.
- **Time.** $O(n)$ expected



Lempel-Ziv 78

- **Decoding.** Read and decode left-to-right.
- **Time.** $O(n)$



Compression

- Compression
- Lempel-Ziv
- Re-Pair and Grammars

Re-Pair Compression

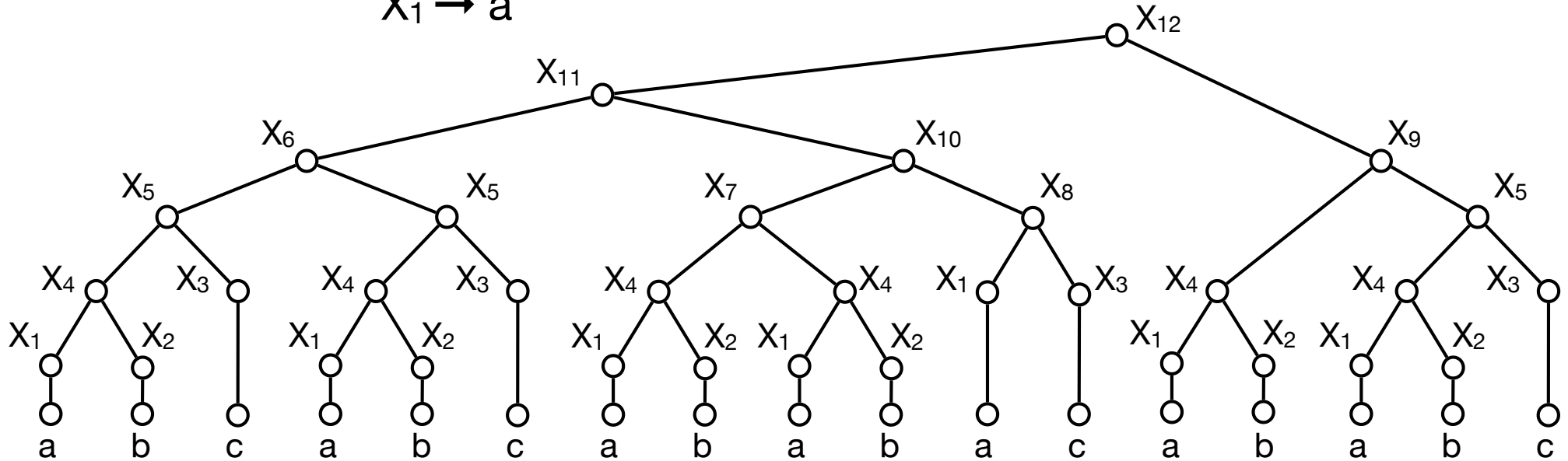
- [Recursive-pairing compression \[Larsson and Moffat 2000\]](#).
 - Start with string S.
 - Replace a most frequent pair ab by new character X_i . Output rule $X_i \rightarrow ab$.
 - Repeat until we have a single pair.
- [Decoding](#). Unfold rules top-down.

X_9	
X_8X_6	$X_9 \rightarrow X_8X_6$
$X_3X_7X_6$	$X_8 \rightarrow X_3X_7$
$X_3X_4X_5X_6$	$X_7 \rightarrow X_4X_5$
$X_3X_4X_5X_1X_2$	$X_6 \rightarrow X_1X_2$
$X_3X_4acX_1X_2$	$X_5 \rightarrow ac$
$X_3X_1X_1acX_1X_2$	$X_4 \rightarrow X_1X_1$
$X_2X_2X_1X_1acX_1X_2$	$X_3 \rightarrow X_2X_2$
$X_1cX_1cX_1X_1acX_1X_1c$	$X_2 \rightarrow X_1c$
abcabcababacababc	$X_1 \rightarrow ab$

Grammar Compression

- **Grammar compression.** Encode string S as an **grammar** G that generates S.
- **Parse tree.** Unfolded set of rules.

$X_{12} \rightarrow X_{11}X_9$	$X_6 \rightarrow X_5X_5$
$X_{11} \rightarrow X_6X_{10}$	$X_5 \rightarrow X_4X_3$
$X_{10} \rightarrow X_7X_8$	$X_4 \rightarrow X_1X_2$
$X_9 \rightarrow X_4X_5$	$X_3 \rightarrow c$
$X_8 \rightarrow X_1X_3$	$X_2 \rightarrow b$
	$X_1 \rightarrow a$



Compression

- Compression
- Lempel-Ziv
- Re-Pair and Grammars