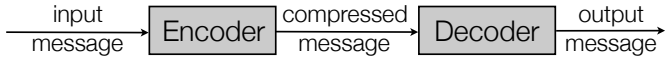
# Compression

02282, Inge Li Gørtz



## Encoding and decoding

- Set of messages S




- Lossless: Input message = output message
- Lossy: Input message ≈ output message



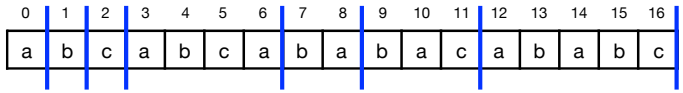
## Compression Quality

- Quality of compression usually measured by:
  - Time used to compress/decompress
  - Size of encoded message
  - Generality of the technique
  - lossy compression : also quality of reconstructed approximation



## Lempel-Ziv Algorithms

## Lempel-Ziv Compression




(0,0,a) (0,0,b) (0,0,c) (3,3,a) (3,1,a) (2,2,a) (6,5,c)



## LZ77 with self-referencing

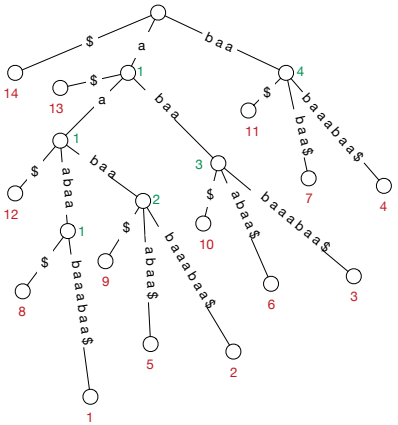

(0,0,a) (0,0,b) (2,1,a) (3,2,b) (5,5,b) (2,10, a)



## Computing LZ77 phrases/LZ77 factorization

- Suffix tree annotated with smallest leaf number below.
- aaabaabaaabaa$
- Factors/phrases: a|aab|aabaaa|baa$

(0,0,a)
(1,2,b)
(3,5,a)
(7,3,$)

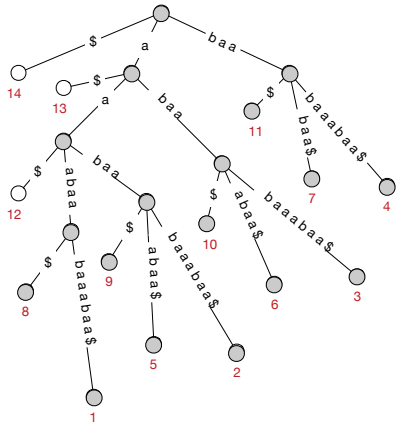





## Computing LZ77 phrases/LZ77 factorization

- Suffix tree + RMQ data structure.
- aaabaabaaabaa$
- Factors/phrases: a|aab|aabaaa|baa$

(0,0,a)
(1,2,b)
(3,5,a)
(7,3,$)

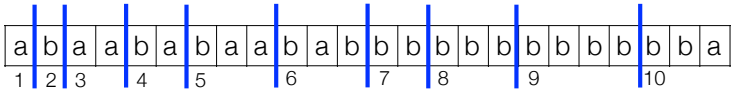


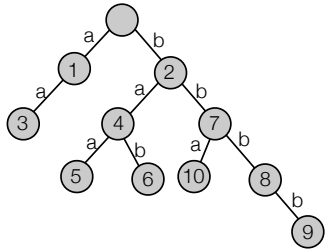
# LZ77

- Greedy parse left-to-right.
- Shortest substring that we have not seen before.
- Can use a suffix tree to compute the phrases.
- Usually used with a sliding window of size W: window = previous W characters
  - use longest match starting in window.
  - pointers use less bits.
  - saves space in encoding
- Alternative encoding: pairs (pointer to previous string, length)
- Instead of suffix tree: hashtable. Hash every substring of length 3. Compare with all substring with the same hash value and return longest.



# LZ78 Example



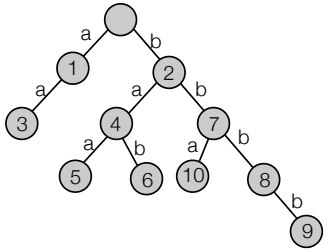




- Example: $(0,a)_1$, $(0,b)_2$, $(1,a)_3$, $(2,a)_4$, $(4,a)_5$, $(4,b)_6$, $(2,b)_7$, $(7,b)_8$, $(8,b)_9$, $(7,a)_{10}$



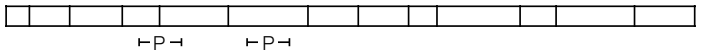
# Random access in LZ78

- Access(x): Return character at position x.
- Data structure:
  - Predecessor data structure over phrases indexed by starting position.
  - Level ancestor data structure on the trie.
- Access(x):
  - Find phrase p containing x.
  - l = x - phrase start + 1.
  - Use level ancestor to jump to the ancestor of p on level l.



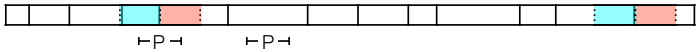



# Pattern matching in LZ78




- Find all occurrences of pattern P in T.
- 2 types of occurrences: overlapping and internal.
- Internal occurences: if P occurs in phrase x then either
  - P occurs in the phrase x refers to.
  - P has an occurrence ending in last position in the phrase.
- Overlapping:
  - Can be found by decompressing around borders: m-1 on each side.

# Pattern matching in LZ78




- Relevant suffix: m characters before end of phrase
- Relevant prefix: m-1 characters from start of phrase
- Find overlapping occurrences and occurrences ending in last position of phrase x:
  - Decompress relevant suffix and prefix of a phrase x
  - Run pattern matching algorithm (e.g. KMP)
- Rest of internal occurrences in x: check parent in trie.
- Time:  decompression of relevant suffixes and prefixes + KMP
  - Decompress: linear time in length: O(nm + occ), if n phrases.
  - KMP: O(nm + occ) in total.
- Best known algorithm obtains O(n+m + occ) time.