5.3. Prediction by partial match (PPM)
Cleary & Witten, 1984

- **Problem with the previous method:**

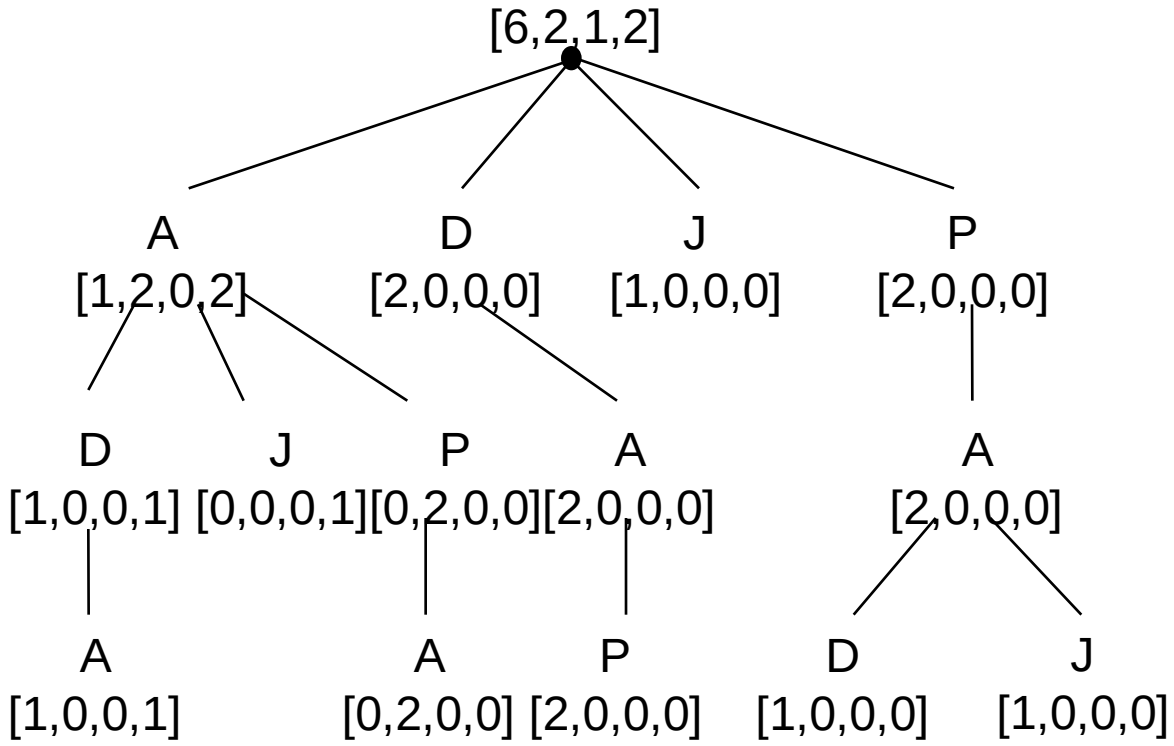
  If prediction fails for the maximal context, information from shorter contexts is not used.

The first idea:

- 'Blend' the predictions of different-length contexts:

$$P(x) = \sum_{o=-1}^{k} w_o P_o(x)$$

Example of backward trie with statistics



Blending examples:

"… ADA|P": $\frac{1}{5}\cdot\frac{1}{4}+\frac{1}{5}\cdot\frac{2}{11}+\frac{1}{5}\cdot\frac{2}{5}+\frac{1}{5}\cdot\frac{1}{2}+\frac{1}{5}\cdot\frac{1}{2}\approx 0.366$

"… PAD|A": $\frac{1}{5}\cdot\frac{1}{4}+\frac{1}{5}\cdot\frac{6}{11}+\frac{1}{5}\cdot\frac{2}{2}+\frac{1}{5}\cdot\frac{2}{2}+\frac{1}{5}\cdot\frac{2}{2}\approx 0.759$

Prediction by partial match (cont.)

- **Practical implementation:**
- Find the longest matching context and calculate the probability estimate:

$$P(x_i | x_{i-k} \ldots x_{i-1}) = \frac{Freq(x_{i-k} \ldots x_i)}{Freq(x_{i-k} \ldots x_{i-1})}$$

- If this is zero, then encode an *escape* symbol.
- Actually, the probability $P(x_i|\ldots)$ must be slightly decreased, in order to make room for the escape.
- Repeat escaping until prediction succeeds or order = –1.

Prediction by partial match (cont.)

- **Exclusion principle:**
- When calculating the probability, exclude from the alphabet the symbols predicted by longer contexts:

$$P_{ex}(x_i | x_{i-j} \ldots x_{i-1}) = \frac{Freq(x_{i-j} \ldots x_{i-1} x_i)}{\sum_{s \in S} Freq(x_{i-j} \ldots x_{i-1} s \,|\, Freq(x_{i-j-1} \ldots x_{i-1} s) = 0)}$$

- Similarly compute $Cum_{ex}(x_i \mid x_{i-j} \ldots x_{i-1})$.
- Again, the probability $P_{ex}(x_i|\ldots)$ must be slightly decreased, in order to make room for the escape

Example of exclusion

- Model basis: “JAPADAPADAA”
- Coding situation: “… APA|P”
- 1. attempt:
  - Context “APA”
  - Zero frequency for “APAP”
  - Encode *escape*; new alphabet: {A,D,J,P} – {D} = {A, J, P}
- 2. attempt:
  - Context “PA”
  - Zero frequency for “PAP”
  - Encode *escape*; new alphabet = {A,J,P} – {D} = {A, J, P}
- 3. attempt:
  - Context “A”
  - Freq(“AP”) = 2, out of 5
  - Encode(‘P’)

PPM: Determining the escape probability

PPMA: Escape has always frequency 1:

$$\text{Prob}_A(esc) = \frac{1}{Freq(x_{i-j}\ldots x_{i-1})+1}$$

$$\text{Prob}_A(x_i \mid x_{i-j}\ldots x_{i-1}) = \frac{Freq(x_{i-j}\ldots x_{i-1}x_i)}{Freq(x_{i-j}\ldots x_{i-1})+1}$$

PPM: Determining the escape probability (cont.)

PPMB: Escape probability is proportional to the observed escape frequency:

$$\text{Prob}_B(esc) = \frac{dif}{Freq(x_{i-j}...x_{i-1})}$$

where

$$dif = \#\{s \mid s \in Succ(x_{i-j}...x_{i-1})$$

A symbol is not predicted until it has occurred twice:

$$\text{Prob}_B(x_i \mid x_{i-j}...x_{i-1}) = \frac{Freq(x_{i-j}...x_{i-1}x_i) - 1}{Freq(x_{i-j}...x_{i-1})}$$

PPM: Determining the escape probability (cont.)

PPMC: Same as PPMB, but prediction can be used already after the first occurrence:

$$\text{Prob}_C(esc) = \frac{dif}{Freq(x_{i-j}...x_{i-1}) + dif}$$

$$\text{Prob}_C(x_i \mid x_{i-j}...x_{i-1}) = \frac{Freq(x_{i-j}...x_{i-1}x_i)}{Freq(x_{i-j}...x_{i-1}) + dif}$$

$$dif = \#\{s \mid s \in Succ(x_{i-j}...x_{i-1})\}$$

Other possible tailorings of PPMC

Update exclusion:

- Increase only the successor frequency for the context really used in prediction.
  This improves the compression efficiency and speed.

Restrict the frequencies:

- E.g. to at most 255 (8 bits), and rescale when needed.
- This makes the method more adaptive.

Lazy exclusion:

- Exclusion not applied at escapes.
- Decreases compression efficiency by about 5%.
- Increases speed by about 50%.
- A feature in *PPMC'*

PPMC: About the implementation

Data structures for contexts and frequencies:

- *Backward tries*, Successors attached as a list/vector to each context node.
- *Forward tries*: Successors are children of context nodes.

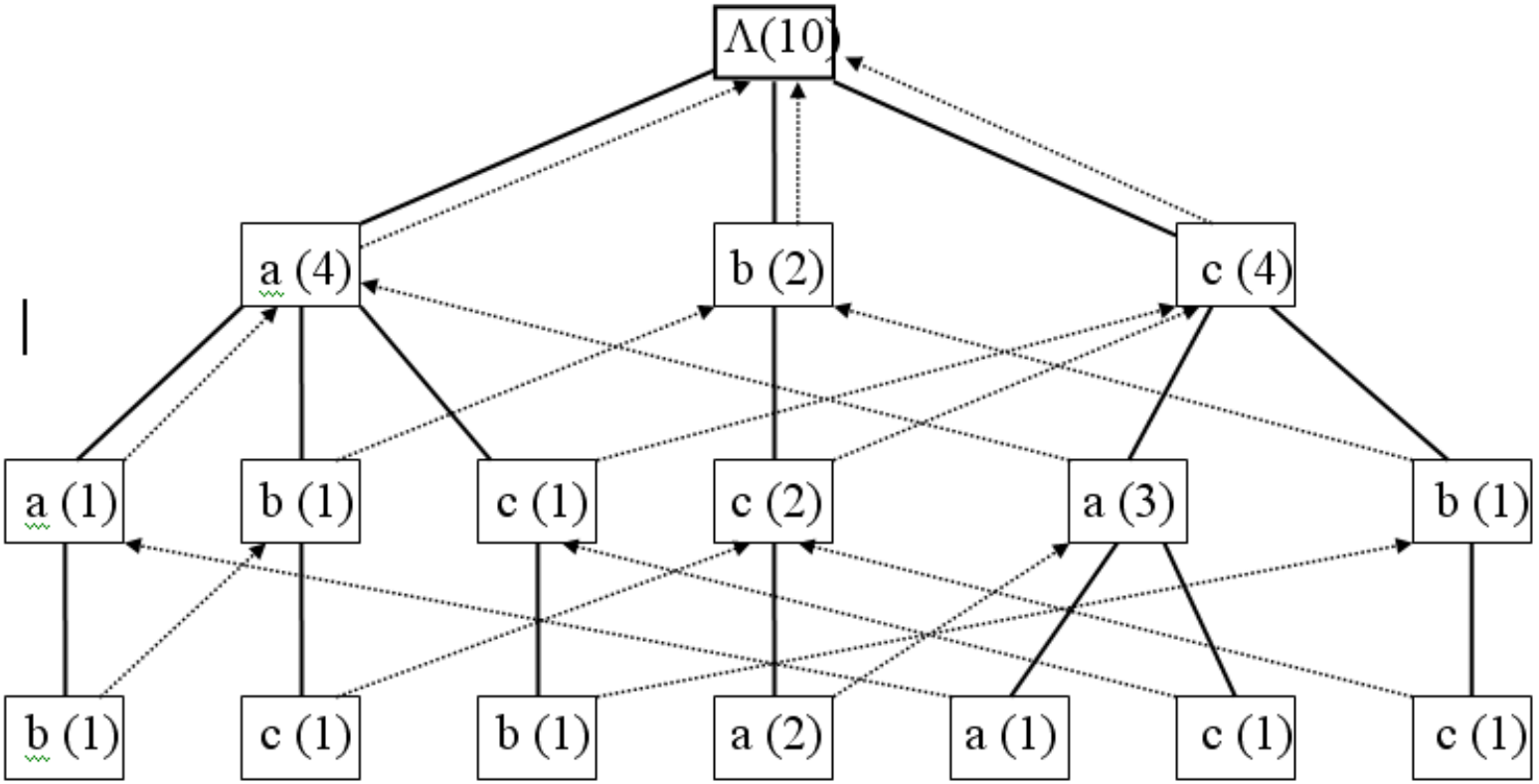
Vine (suffix) pointers:

- Make forward tries much more efficient.
- Escape to a shorter context: Follow the vine pointer.
- Moving to the next current context: Follow the vine pointer.

Efficiency of PPMC:

- One of the best methods ($\approx$ 2 bits/symbol for English)
- Relatively slow.

Schematic example of vine (suffix) pointers, for source message message “cacbcaabca”, max context 2



Encoding of successor ‘b’:

(1) Take the vine pointer from b-c-a to c-a.

(2) No successor ‘b’; escape and take vine pointer to ’a’

(3) Encode succ. ‘b’, having freq 1

Newer versions of PPM

PPMD (Howard, 1993):

- At escape, add ½ to escape and symbol probabilities
- A new formula for escape probability:

$$P(escape) = u / (2n)$$

where u = number of symbols in the context,

n = context frequency.

Newer versions of PPM (cont.)

PPM* (Cleary, Teahan, Witten, 1995):

- Unbounded context length
- Start prediction from the longest *deterministic* context, if such exists, otherwise from the longest stored context (deterministic = only one symbol has occurred as successor, but possibly several times)
- Data structure:
  - trie with deterministic paths to leaves,
  - pointers to corresponding places in the message,
  - linked list to currently active contexts.
- Escapes as in PPMC; no update exclusion.

Newer versions of PPM (cont.)

PPMZ (Bloom, 1998):

- Unbounded contexts only in deterministic cases.
- Prediction starts from the context having the biggest Prob(Most probable symbol)
- Escape probability derived from
  - statistics on context
  - context order
  - escape count
  - successful matches
- Complicated and slow; 'ultimate' PPM-method.

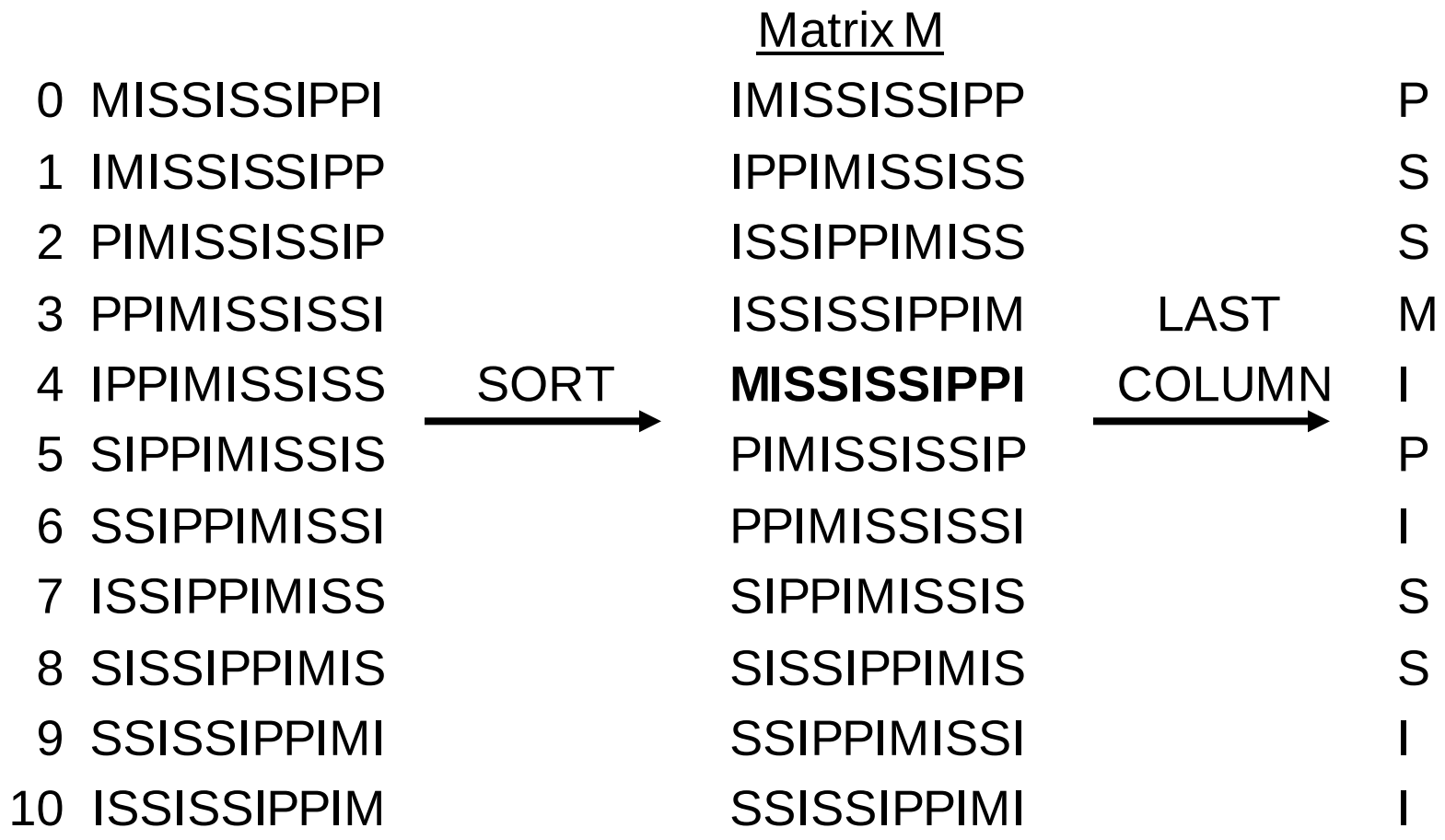
5.4. Burrows-Wheeler transform

- Totally different technique from others
- Effect is close to PPM*
- Correct functioning is not obvious
- Compression efficiency is not obvious
- Good speed is not obvious

Transform technique:

- *Sort* all *rotations* of the message
- Transform result = sequence *L* of the *last* symbols of the sorted rotations, plus the order number of the original message within the sorted order.

Burrows-Wheeler: Example sorting



Index of original message = 4

Burrows-Wheeler: Reverse transformation

- The sequence *F* of the first symbols of the sorted rotations can be recovered by sorting *L*.
  [Each column contains exactly the same characters in different orders.]
- The *L*[*i*]*F*[*i*] pairs can be linked so that the original message can be recovered in forward or backward order.
- In the example:

  MI, IS, SS, SI, IS, SS, SI, IP, PP, PI

- The task resembles connecting domino pieces, but ***how to find the correct order of linking?***

Burrows-Wheeler transform: Basic observations

Index	*L*	*F*
0	P	I …
1	S	I …
2	S	I …
3	M	I …
4	I	M …
5	P	P …
6	I	P …
7	S	S …
8	S	S …
9	I	S …
10	I	S …

Imagine that "…" corresponds to the rest of the sorted rotations.

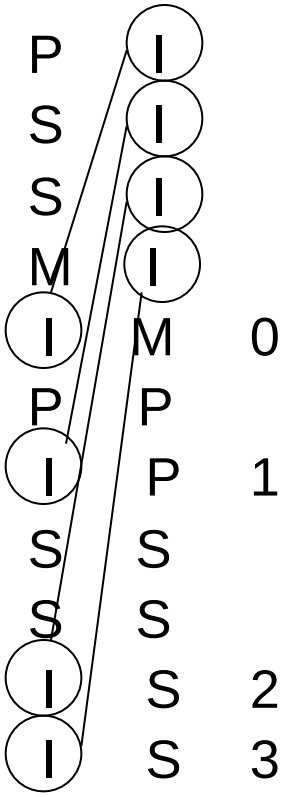
The first 4 rows correspond to sorted order of "I…", prefixed by the last symbols of the rotations = symbols preceding "I" in the original sequence.

The rows with *L* = 'I' are also in sorted order, because the first symbol is the same, and the tails are sorted.

The "I"-characters in columns *L* and *F* correspond to each other **in the same order**. This forms the connections between pairs.

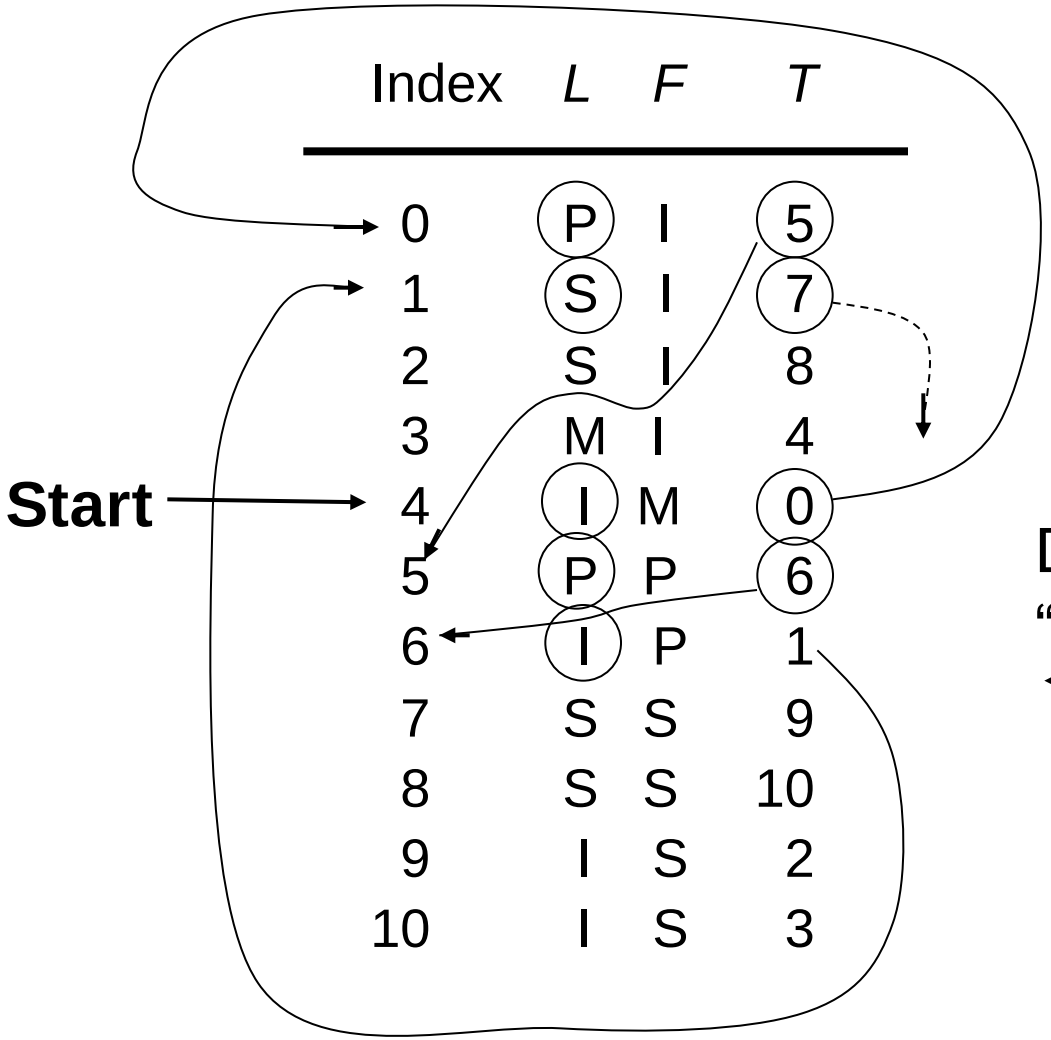
Numbering of corresponding symbols

Index	*L*	*F*	*T*
0	P	I	
1	S	I	
2	S	I	
3	M	I	
4	I	M	0
5	P	P	
6	I	P	1
7	S	S	
8	S	S	
9	I	S	2
10	I	S	3

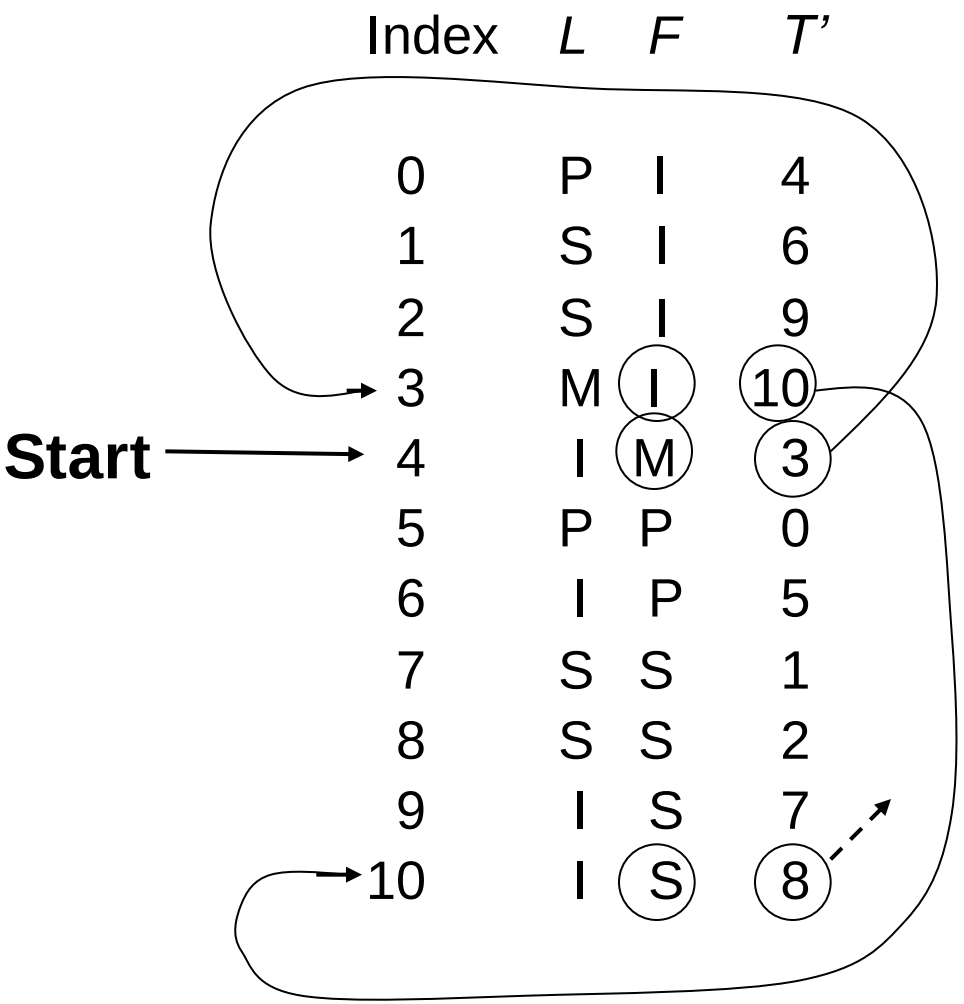


Note. The numbering in column *T* can be computed in linear time

Burrows-Wheeler: Reverse transformation



Alternative: forward linking

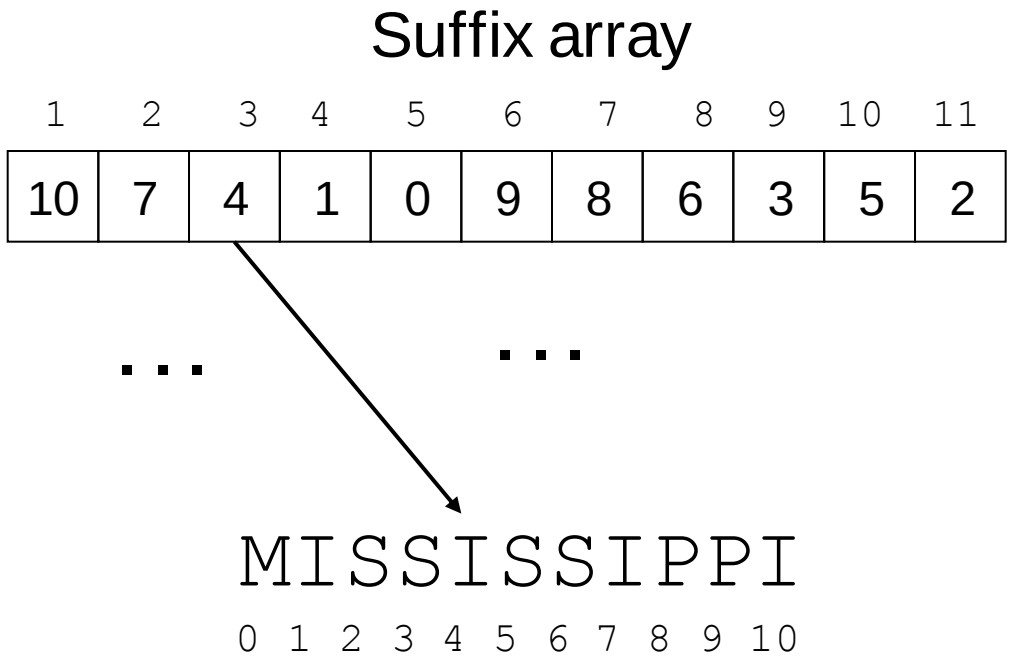


Decoding order:
"MIS…"

Burrows-Wheeler: Implementation of sorting

- Create a vector of pointers (sometimes called *suffix array*) to the message symbols.
- Sort the vector of pointers.
- *Comparison-based* methods: A pointer is ‘less’ than another, if the strings from the pointed indexes ahead are lexicographically in this order.
- Alternative: *radix sort* in forward order. First partition according to the first symbols of rotations, then sub-partition according to the second, etc. After a few levels, finalize with some simple technique.
- Another alternative: *suffix tree*.

Burrows-Wheeler: Data structure for sorting



The example pointer refers to rotation ”ISSIPPIMISS”, which is the third in alphabetic order of rotations.

Properties of the last column of sorted rotations

- Observation: The symbols are predecessors (in the original message) of sorted, i.e. very similar strings.
- *Contexts* of the symbols in the last column are the *whole* rest parts of the message, *n*-1 symbols each.
- Compared to PPM, contexts are longer and reverses (mirrors) of the PPM contexts.
- The last column typically consists of long sequences of the same symbol, enabling effective compression.

Example fragment of transformed English text

tssurssssssssssuussssssrsssuuutssrrmtsssssuussssssssstmmmsrrstssss
sssllllmrmpeeeeevvtttt
tttgmgtlmmtltrgttttmtttllgmmmrrrppwmeemttttttttttmttmiiutvtrvvvniicvrrir
uimmmcctucriittvvcvivtivntsvrrevtttirnnnntvvuunincrivtvttircmmvvmve
reirviv
ccccccccccccccccccccccchhhhHHHHHHHHHHHHHHHHHHHHHHHHHhhCC ccccccccuuuiimreiimm m r cfc ch cc
mGGGGccccmmm
cfiicucciunumeuuumitucmucninunncii vt
nvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvh
nrrerrrrerrrrrrrrrrrrrrrrrrrrrrs
sssssssscsssssssssssssssdnxxxr xsxxxxssxxxxxxxxxxxxsrrrrmmmh
ch chh hhm cIhjj h
ccccccccccccccmmmmmmmmmmmmmmmmmmmmhc hccc h
mc mcchchcmhhhhhhchhcCC

Encoding of the last column of sorted rotations

- Good approach by experience:
  - Maintain a list of symbols in the alphabet
  - For each symbol, encode the ***order number*** of the symbol in the list, and ***move the symbol to the front*** of the list.
- Most of the numbers (typically 60-70%) will be zeroes, and also the rest are quite small.
- Statistical methods, like arithmetic coding, can be applied.
- *Run-length coding* of zeroes improves the compression.
- The distribution of the order numbers is so skew that even *unary coding* gives good results.

Example of move-to-front technique for encoding the last column into small numbers

Index	**P**	**S**	**S**	**M**	**I**	**P**	**I**	**S**	**S**	**I**	**I**
0	I	P	S	S	M	I	P	I	S	S	I
1	M	I	P	P	S	M	I	P	I	I	S
2	P	M	I	I	P	S	M	M	P	P	P
3	S	S	M	M	I	P	S	S	M	M	M
Order no	**2**	**3**	**0**	**3**	**3**	**3**	**1**	**3**	**0**	**1**	**0**

Burrows-Wheeler: some issues

On-line compression?

- In its basic form, the method is *off-line* (multi-phase).
- By doing the compression blockwise, the method can be applied to on-line situations.

Performance:

- Better compression than with PPMC or gzip; worse than PPMZ.
- Order of magnitude faster than PPMZ; slightly slower than gzip.
- Practical implementation: ***bzip2*** (Unix / Linux)