BWT - Burrows Wheeler Transform

A block-sorting lossless data compression algorithm

- Introduction
- Ciphering phase
- Deciphering phase
- Deciphering phase – improvements
- Relevant compression techniques
- Effectiveness of BWT
- Comparison with other compression programmes
- References

Alberto Chiusole, s249223
Advanced Programming and Algorithmic Design @ uniTS – A.Y. 2017-2018

Introduction to BWT

- Used to preprocess strings before actual compression
- Produces output with high Index of Coincidence (IC)

  ==

  Identical characters are often in groups
- Compression is easy, i.e. with *run-length* encoding

Ciphering phase (1)

- Encode the string `mississippi`, length `N`
- Add separator symbol (End Of Line) at the end;
  it must not be present in the string
- `S = 'mississippi' + $`

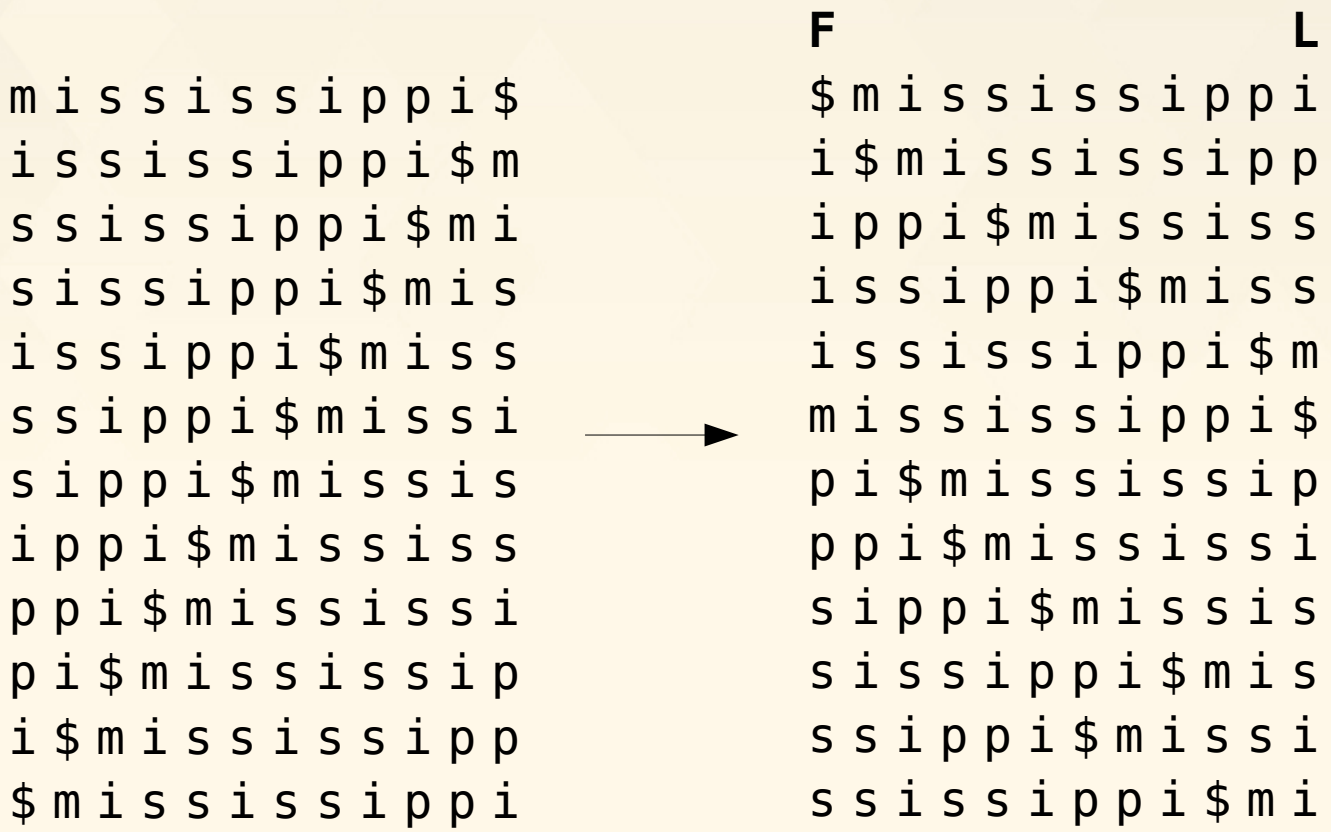
Ciphering phase (2)

- Create all the cyclic rotations/circular shifts of the string `S`
- Single character permutations; move the first character after the last

```
m i s s i s s i p p i $
i s s i s s i p p i $ m
s s i s s i p p i $ m i
s i s s i p p i $ m i s
i s s i p p i $ m i s s
s s i p p i $ m i s s i
s i p p i $ m i s s i s
i p p i $ m i s s i s s
p p i $ m i s s i s s i
p i $ m i s s i s s i p
i $ m i s s i s s i p p
$ m i s s i s s i p p i
```

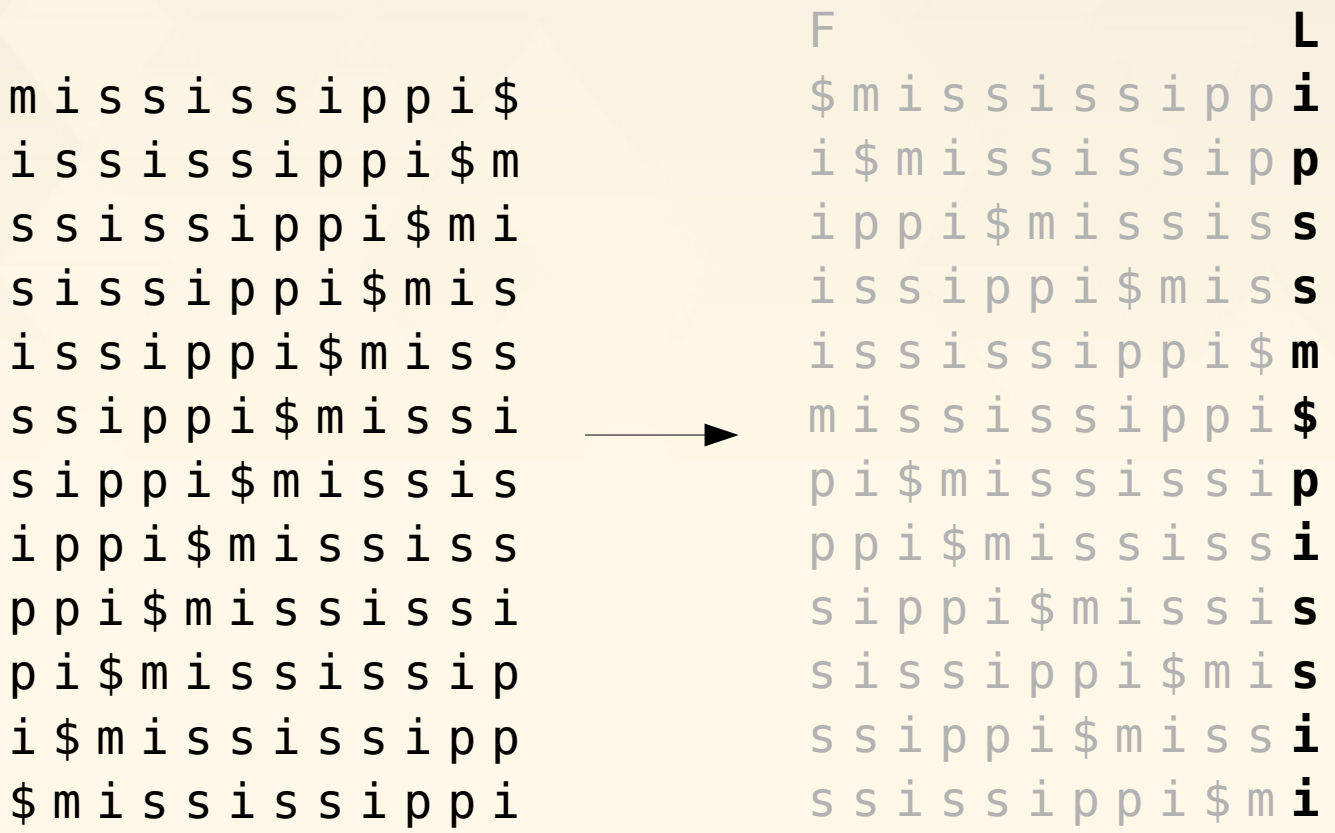
Ciphering phase (3)

- Sort all the permutations
- F and L are the First and Last columns of the matrix



Ciphering phase (4)

- Take only the last column `L = ipssm$pissii`
- Encoding done!



Deciphering phase (1)

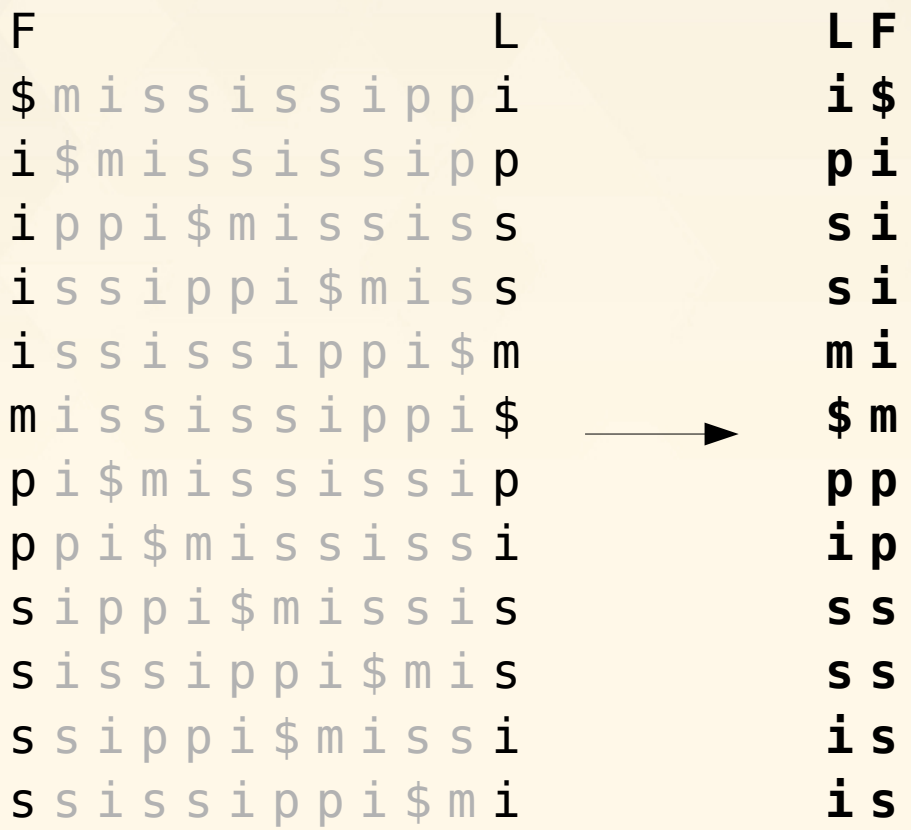
- The decipher only sees the encoded string `L` but,
- sort `L`, obtain `F = sort(L) = $iiiimppssss`

F										L
$	m	i	s	s	i	s	s	i	p	p	i
i	$	m	i	s	s	i	s	s	i	p	p
i	p	p	i	$	m	i	s	s	i	s	s
i	s	s	i	p	p	i	$	m	i	s	s
i	s	s	i	s	s	i	p	p	i	$	m
m	i	s	s	i	s	s	i	p	p	i	$
p	i	$	m	i	s	s	i	s	s	i	p
p	p	i	$	m	i	s	s	i	s	s	i
s	i	p	p	i	$	m	i	s	s	i	s
s	i	s	s	i	p	p	i	$	m	i	s
s	s	i	p	p	i	$	m	i	s	s	i
s	s	i	s	s	i	p	p	i	$	m	i

Deciphering phase (2)

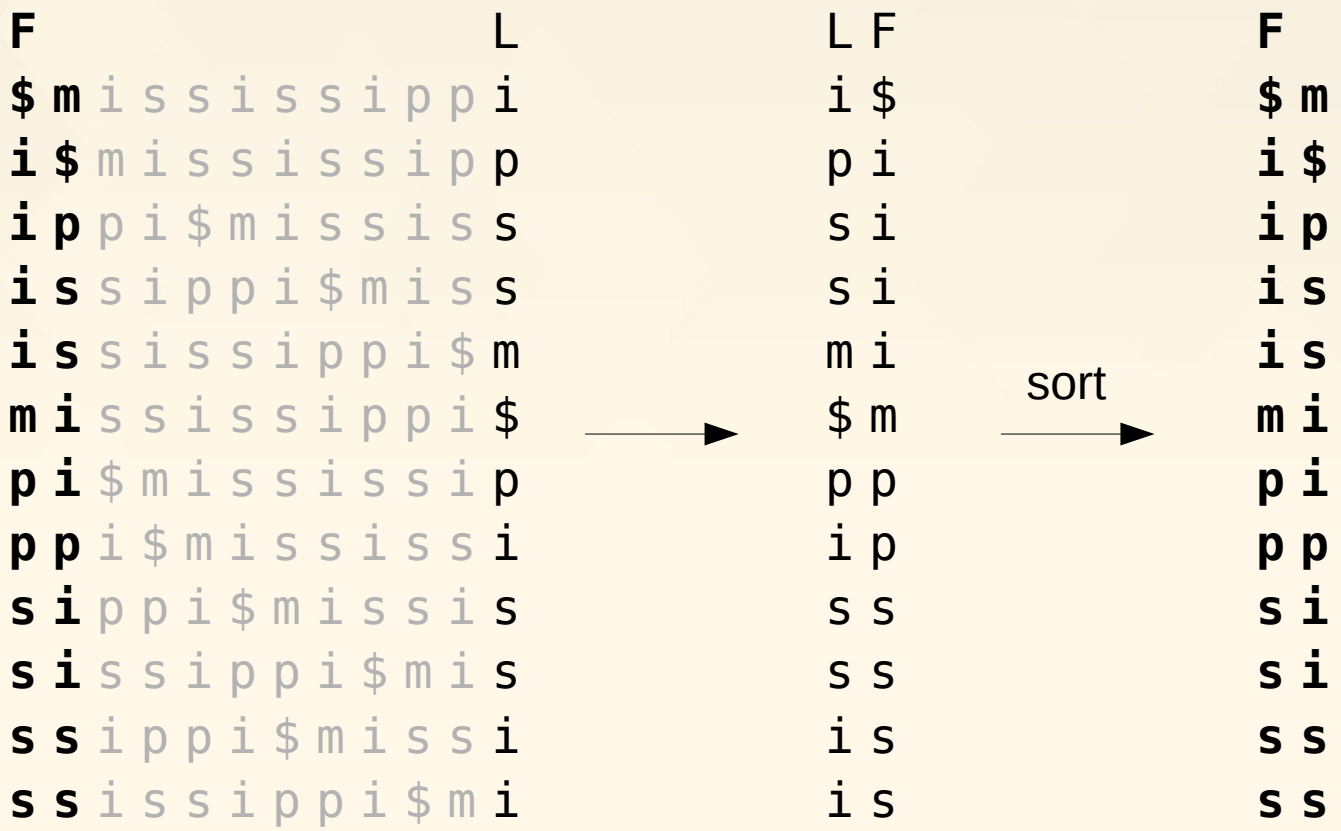
- To rebuild the initial matrix:

  stick together the columns `L  F`



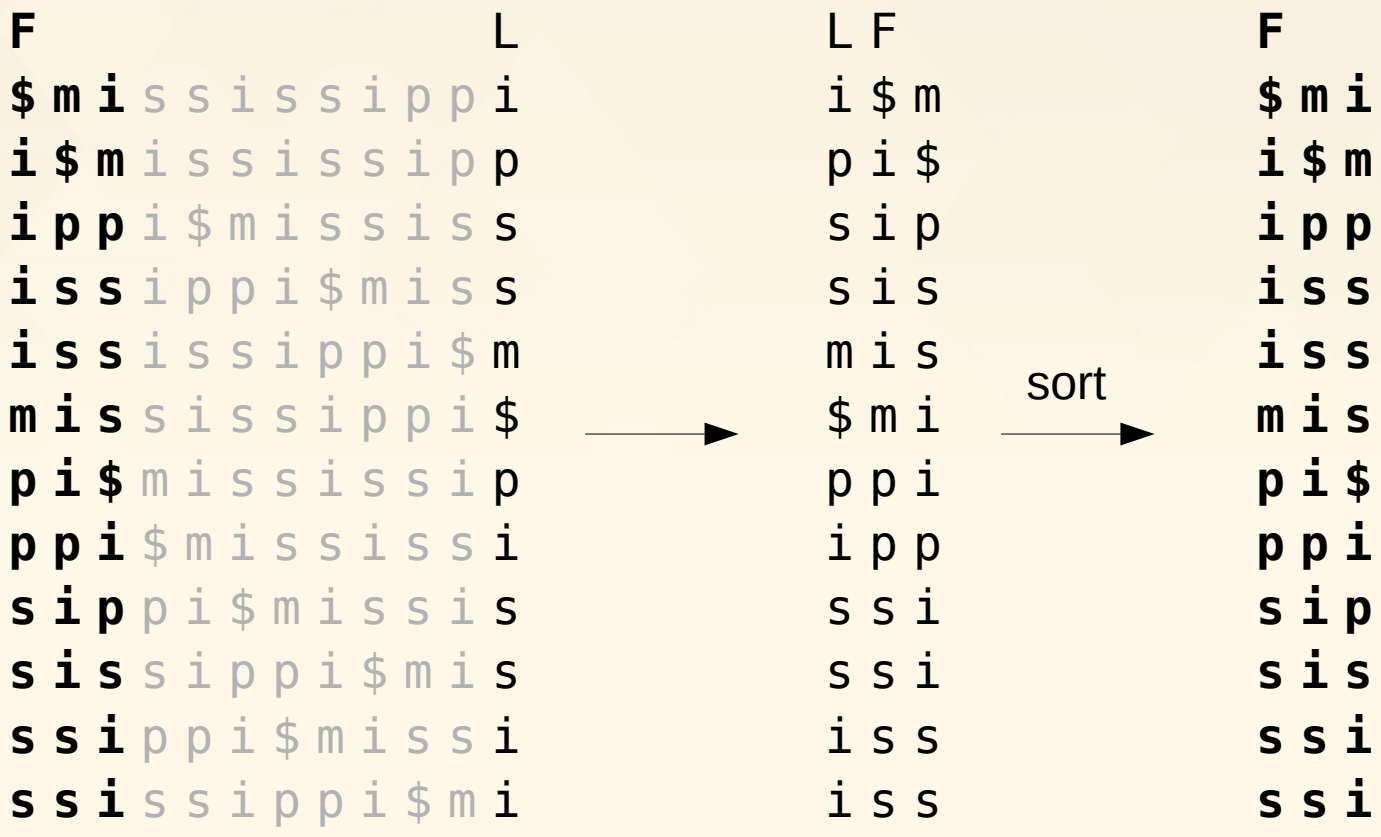
Deciphering phase (3)

- The 2-mer `L  F` is made of substrings of `S`
- Sort lexicographically, first 2 columns original matrix!



Deciphering phase (4)

- Take the 3-mer made of the first 2 columns + `L`
- Sort lexicographically, first 3 columns original matrix!



Deciphering phase (5)

- Repeat until the original matrix is complete
- The string `S` is the one ending with `$`

- Simple approach, but poor performances:
  - $O(N^2)$ *space* consumption for storing the N^2 matrix
  - $O(N^2 \log N)$ computations; N *k*-mers sorting operations with $O(\log N)$ sorting algorithm

Deciphering phase - improvements (1)

- Curious *first-last* property:
- The *relative* order in the group of same letters is preserved: observe, e.g., the relative order of the `i`'s

F										L
$	m	i	s	s	i	s	s	i	p	p	**$_1$**
i$_1$	$	m	i	s	s	i	s	s	i	p	p$_1$
i$_2$	p	p	i	$	m	i	s	s	i	s	s$_1$
i$_3$	s	s	i	p	p	i	$	m	i	s	s$_2$
i$_4$	s	s	i	s	s	i	p	p	i	$	m$_1$
m$_1$	i	s	s	i	s	s	i	p	p	i	$
p$_1$	i	$	m	i	s	s	i	s	s	i	p$_2$
p$_2$	p	i	$	m	i	s	s	i	s	s	**i$_2$**
s$_1$	i	p	p	i	$	m	i	s	s	i	s$_3$
s$_2$	i	s	s	i	p	p	i	$	m	i	s$_4$
s$_3$	s	i	p	p	i	$	m	i	s	s	**i$_3$**
s$_4$	s	i	s	s	i	p	p	i	$	m	**i$_4$**

Proof:

- consider rows starting with a specific letter
- delete the first letter, the columns are still sorted
- apply those first letter after `L`, columns are still sorted
- these new rows are available inside the matrix, and they are still sorted relatively to one another

Deciphering phase - improvements (2)

- Start from the EOL symbol `$` in `F`, go backwards into `L`: find `$i_1$`
- Search for `$i_1$` in `F`, go backwards in `L` and find `$p_1$`

F										L
$	m	i	s	s	i	s	s	i	p	p	**i_1**
i_1	$	m	i	s	s	i	s	s	i	p	**p_1**
i_2	p	p	i	$	m	i	s	s	i	s	s_1
i_3	s	s	i	p	p	i	$	m	i	s	s_2
i_4	s	s	i	s	s	i	p	p	i	$	m_1
m_1	i	s	s	i	s	s	i	p	p	i	$
p_1	i	$	m	i	s	s	i	s	s	i	p_2
p_2	p	i	$	m	i	s	s	i	s	s	i_2
s_1	i	p	p	i	$	m	i	s	s	i	s_3
s_2	i	s	s	i	p	p	i	$	m	i	s_4
s_3	s	i	p	p	i	$	m	i	s	s	i_3
s_4	s	i	s	s	i	p	p	i	$	m	i_4

S = ...**pi**

Deciphering phase - improvements (3)

- Search for `$p_1$` in `F`, go backwards in `L` and find `$p_2$`
- Search for `$p_2$` in `F`, go backwards in `L` and find `$i_2$`

F										L
$	m	i	s	s	i	s	s	i	p	p	i_1
i_1	$	m	i	s	s	i	s	s	i	p	p_1
i_2	p	p	i	$	m	i	s	s	i	s	s_1
i_3	s	s	i	p	p	i	$	m	i	s	s_2
i_4	s	s	i	s	s	i	p	p	i	$	m_1
m_1	i	s	s	i	s	s	i	p	p	i	$
p_1	i	$	m	i	s	s	i	s	s	i	**p_2**
p_2	p	i	$	m	i	s	s	i	s	s	**i_2**
s_1	i	p	p	i	$	m	i	s	s	i	s_3
s_2	i	s	s	i	p	p	i	$	m	i	s_4
s_3	s	i	p	p	i	$	m	i	s	s	i_3
s_4	s	i	s	s	i	p	p	i	$	m	i_4

S = ...**ippi**

Deciphering phase - improvements (4)

- No need to rebuild the original matrix
- Linear memory consumption: $2\ N \rightarrow O(N)$
- Linear number of computations to build the 2 first-last arrays

Relevant compression techniques (1)

- *Move-to-front transform* encoding can be applied after BWT and can improve Index Coincidence
- using the alphabet `a..z`, take a string (e.g., `` `S = ipssm$pissii` ``), note the index of each element

  but also modify the alphabet by moving to the front the last char of the alphabet used
- There will be lots of low index values

MTF(S, “a..z+$”) = [8, 15, 18, 0, 14, 26, 3, 4, 4, 0, 1, 0]

Relevant compression techniques (2)

- An encoded `L` string can be compressed with, e.g., *run-length encoding*.
- For long strings, it reduces the memory needed, especially with short alphabets (e.g., DNA strings)

```
RLE('ipssm$pissii') = i1 p1 s2 m1 $1 p1 s2 i2
```

Comparison with other compression progr.

Program name	CPU Time (s) Compression	Decompression	Average bit/char
compress	9.6	5.2	3.63
gzip	42.6	4.9	2.71
BWT + Huffman coder	*51.1*	*9.4*	*2.43*
comp-2	603.2	614.1	2.47

References

- Burrows, Michael; Wheeler, David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation