2-INF-237 Vybrané partie z dátových štruktúr

2-INF-237 Selected Topics in Data Structures

- Instructor: Broňa Brejová
- E-mail: brejova@fmph.uniba.sk
- Office: M163

- Course webpage: http://compbio.fmph.uniba.sk/vyuka/vpds/

Burrows-Wheeler transform (BWT) 1994

T =banana$

Sort all rotations of the word lexicographically:

b	a	n	a	n	a	$
a	n	a	n	a	$	b
n	a	n	a	$	b	a
a	n	a	$	b	a	n
n	a	$	b	a	n	a
a	$	b	a	n	a	n
$	b	a	n	a	n	a

$	b	a	n	a	n	**a**
a	$	b	a	n	a	**n**
a	n	a	$	b	a	**n**
a	n	a	n	a	$	**b**
b	a	n	a	n	a	**$**
n	a	$	b	a	n	**a**
n	a	n	a	$	b	**a**

BWT = annb$aa

Can be computed using suffix array: BWT[i] = T[SA[i]-1] (or T[n] if SA[i]=0)

Reverse transformation

Get first column (F) by sorting the last (L):

F		L
\$	...	a
a	...	n
a	...	n
a	...	b
b	...	\$
n	...	a
n	...	a

Reverse transformation

Observations:

– $F[i]$ follows $L[i]$ in T

– jth occurrence of x in F is the same as jth occurrence of x in L

0:	**$**$_6$	b	a	n	a	n	**a**$_5$
1:	**a**$_5$	$	b	a	n	a	**n**$_4$
2:	**a**$_3$	n	a	$	b	a	**n**$_2$
3:	**a**$_1$	n	a	n	a	$	**b**$_0$
4:	**b**$_0$	a	n	a	n	a	**$**$_6$
5:	**n**$_4$	a	$	b	a	n	**a**$_3$
6:	**n**$_2$	a	n	a	$	b	**a**$_1$

Reverse transformation

– $F[i]$ follows $L[i]$ in T

– jth occurrence of x in F is the same as jth occurrence of x in L

**$**$_6$	b	a	n	a	n	**a**$_5$
a$_5$	$	b	a	n	a	**n**$_4$
a$_3$	n	a	$	b	a	**n**$_2$
a$_1$	n	a	n	a	$	**b**$_0$
b$_0$	a	n	a	n	a	**$**$_6$
n$_4$	a	$	b	a	n	**a**$_3$
n$_2$	a	n	a	$	b	**a**$_1$

Find $ in L, get $T[0]$ in F ($T[0] = b$, 1st)

Find 1st b in L, get $T[1]$ in F ($T[1] = a$, 3rd)

Find 3rd a in L, get $T[2]$ in F ($T[2] = n$, 2nd)

In general: If $T[i]$ in $L[j]$, get $T[i+1]$ in $F[j]$

Reverse transformation

0:	$\$_6$	a_5
1:	a_5	n_4
2:	a_3	n_2
3:	a_1	b_0
4:	b_0	$\$_6$
5:	n_4	a_3
6:	n_2	a_1

Sort L to get F, then build the following data structures:

Representation of F :

$: 0

a: 1

b: 4

n: 5

0: $
1: a
2: a
3: a
4: b
5: n
6: n

Representation of L:

$: 4

a: 0, 5, 6

b: 3

n: 1, 2

Use of Burrows-Wheeler transform in compression

T $=$`ema.ma.mamu.mama.ma.emu.ema.sa.ma$`

BWT $=$`auaaaauaammsmmmmmm$....ae.e..ea.mm`

In a given region of SA common prefixes

Preceded by similar letters

In our case `'.'` preceded by u,a

Regions with the same letter repeated or few letters mixed

Use of Burrows-Wheeler transform in compression

T =`ema.ma.mamu.mama.ma.emu.ema.sa.ma$`

BWT =`auaaaauaammsmmmmmm$....ae.e..ea.mm`

Regions with the same letter repeated or few letters mixed

Move-to-front recording: replace $T[i]$ by the number of distinct letters since last occurrence of $T[i]$ in $T[0..i-1]$

`$.aemsu|auaaaauaammsmmmmmm$....ae.e..ea.mm`

4, 1, 1, 0, 0, 0, 1, 1, 0, 3, 0, 3, 1, 0, 0, 0, 0, 0, 6, 6, 0, 0, 0, 4, 6, 2, 1, 1, 0, 1, 2, 2, 4, 0

Small numbers, many zeroes, in English text 50% zeroes

Use of Burrows-Wheeler transform in compression

Encode MTF of BWT by Huffman or arithmetic encoding

$T \longrightarrow BWT \longrightarrow MTF \longrightarrow$ Huffman/arithmetic encoding $\longrightarrow$ compressed T

e.g. bzip2 (with further details)

- Let S be a string in which $a \in \Sigma$ occurs n_a times
- Its **0-th order empirical entropy** is $H_0(S) = \sum_a \frac{n_a}{n} \lg \frac{n}{n_a}$
- In our example:
  $H_0(T) = 2.37$, $H_0(\text{MTF of BWT of } T) = 2.18$

Higher order entropy and BWT

- Let S be a string in which $a \in \Sigma$ occurs n_a times
- Its **0-th order empirical entropy** is $H_0(S) = \sum_a \frac{n_a}{n} \lg \frac{n}{n_a}$
- **k-th order empirical entropy** is
  $H_k(S) = (1/n) \sum_{w \in \Sigma^k} |w_S| H_0(w_S)$
  where w_S is concatenation of symbols following occurrences of w in S
- Corrected $H_k^*(S)$ ensures that $nH_k^*(S) \geq \lg n$
- H_0-based encoding of MTF of BWT uses at most
  $8nH_k(S) + (\mu + 2/25)n + O(\sigma^{k+1} \log \sigma)$ bits for any $k \geq 0$
  μ depends on encoding, $\mu = 1$ for Huffman, less for arithmetic
  [Manzini 2001]

A different reverse transformation (backwards)

$LF[i]$**:** row j in which $F[j]$ corresponds to $L[i]$

Example: $LF[2] = 6$

If i corresponds to $T[k..n]$, then $LF[i]$ corresponds to $T[k-1..n]$

0:	**\$**$_6$	b	a	n	a	n	**a**$_5$
1:	**a**$_5$	\$	b	a	n	a	**n**$_4$
2:	**a**$_3$	n	a	\$	b	a	**n**$_2$
3:	**a**$_1$	n	a	n	a	\$	**b**$_0$
4:	**b**$_0$	a	n	a	n	a	**\$**$_6$
5:	**n**$_4$	a	\$	b	a	n	**a**$_3$
6:	**n**$_2$	a	n	a	\$	b	**a**$_1$

```
T[n] = $; s = 0;
for(i=n-1; i>=0; i--) { T[i] = L[s]; s = LF[s]; }
```

A different reverse transformation (backwards)

$LF[i]$**:** row j in which $F[j]$ corresponds to $L[i]$

If i corresponds to $T[k..n]$, then $LF[i]$ corresponds to $T[k-1..n]$

$C[x]$**:** the index of first occurrence of x in F

$rank[x,i]$**:** the number of occurrences of x in $L[0..i]$

$LF[i] = C[L[i]] + rank[L[i], i-1]$

i	$F[i]$	$L[i]$	$r[\$, i]$	$r[a, i]$	$r[b, i]$	$r[n, i]$
0:	$\$_6$	a_5	0	1	0	0
1:	a_5	n_4	0	1	0	1
2:	a_3	n_2	0	1	0	2
3:	a_1	b_0	0	1	1	2
4:	b_0	$\$_6$	1	1	1	2
5:	n_4	a_3	1	2	1	2
6:	n_2	a_1	1	3	1	2

Note: less practical decompression (more memory, result in reverse order)

Use of Burrows-Wheeler transform for string matching

FM index [Ferragina and Manzini 2000]

Occurrences of pattern P in T form an interval in SA

Consider suffixes of P from shortest, update interval

Example: search for $\mathrm{P} = \mathtt{nan}$

$banan**a**	$banan**a**	$banan**a**	$banan**a**
a$bana**n**	a$bana**n**	a$bana**n**	a$bana**n**
ana$ba**n**	ana$ba**n**	**an**a$ba**n**	ana$ba**n**
anana$**b**	anana$**b**	**an**ana$**b**	anana$**b**
banana**$**	banana**$**	banana**$**	banana**$**
na$ban**a**	**n**a$ban**a**	na$ban**a**	na$ban**a**
nana$b**a**	**n**ana$b**a**	nana$b**a**	**nan**a$b**a**

FM index

$C[x]$: the index of first occurrence of x in F

$rank[x, i]$: the number of occurrences of x in $L[0..i]$

Example:

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
T[i]	e	m	a	.	m	a	.	m	a	m	u	.	m	a	m	a	.	m	a	.	e	m	u	$

SA[i]	23	19	16	3	11	6	18	15	2	5	13	8	0	20	17	14	1	4	12	7	21	9	22	10
L[i]	u	a	a	a	u	a	m	m	m	m	m	m	$	.	.	a	e	.	.	.	e	a	m	m

r$(i)	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
r.(i)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	3	4	5	5	5	5	5
ra(i)	0	1	2	3	3	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	6	6	6
re(i)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	2	2	2	2
rm(i)	0	0	0	0	0	0	1	2	3	4	5	6	6	6	6	6	6	6	6	6	6	6	7	8
ru(i)	1	1	1	1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2

x	$	.	a	e	m	u
C(x)	0	1	6	12	14	22

Counting occurrences of P using FM index

FM index:

$C[x]$: the index of first occurrence of x in F

$\mathrm{rank}[x, i]$: the number of occurrences of x in $L[0..i]$

```
l = 0; r = n;
for(i = m-1; i >= 0; i--) {
   a = P[i];
   l = C[a] + rank[a, l-1];
   r = C[a] + rank[a, r] - 1;
   if(l > r) return 0; // no occurrences
}
return r - l + 1;
```

To report positions, we also need suffix array SA

Counting occurrences of P using FM index

$C[x]$: the index of first occurrence of x in F

$\mathrm{rank}[x, i]$: the number of occurrences of x in $L[0..i]$

Update of ℓ: $\ell = C[a] + \mathrm{rank}[a, \ell - 1]$

Update of r: $r = C[a] + \mathrm{rank}[a, r] - 1$

$$
\begin{array}{c|ll|}
 & X\ldots & a \\
 & aX\ldots & \\
 & aY\ldots & \\
\ell' & aS\ldots & \\
r' & aS\ldots & \\
 & Y\ldots & a \\
\ell & S\ldots & \\
 & S\ldots & \\
r & S\ldots & \\
\end{array}
$$

Memory of FM index for counting occurrences of P in T

- Works in $O(m)$ time
- Requires arrays C ($\sigma \lg n$ bits) and $rank$ ($n\sigma \lg n$ bits), no need to store the text, its BWT, SA
- Wavelet trees store $rank$ in $n \lg \sigma + o(n \log \sigma) + O(\sigma \lg n)$ bits, time increases to $O(m \lg \sigma)$
- Alternatively use σ compressed binary vectors precomputed for ranks Memory $n \lg \sigma + O(n) + o(\sigma n)$ bits, time $O(m)$
- We need ranks over BWT of T, better compressible than T

Memory of FM index for printing occurrences of P in T

- Needs also to access $SA[\ell..r]$, SA needs n lg n bits
- Store only some values of SA, recompute the rest using LF

  t times less memory, t times more time to print each item

Store only suffixes of the form $T[it..n]$ for $i = 0, 1, \ldots$

```
get_SA(i)  {
  if (SA[i] is stored) {
      return SA[i];
  } else {
      return get_SA(LF[i])+1; // needs C, rank, L
  }
}
```

How exactly to store only some rows of SA?

Summary: Burrows-Wheeler transform

- BWT can be computed in $O(n)$ time via suffix array
- Reverse transformation also in $O(n)$
- Can be used in compression, groups the same letters together
- Can be used in FM index for string matching in $O(m+k)$
- String matching in BWT requires ranks: various solutions with succint data structures