

A Block-Sorting Lossless Data Compression Algorithm

Finn Stutzenstein, Levin Nemesch, Joshua Sangmeister

January 29, 2021

Algorithm Engineering - Übung 5

- Idee: Möglichst kurze Kodierung einer Information (z.B. Text)
- Bsp. *Run-length encoding*: aaaabbbbcaaa \longrightarrow 4a3bc3a
- Problem: Finde gutes Encoding. Viele Ideen:
 - 17% e \leftrightarrow 0,02% q, ...
 - Ein Buchstabe wird besonders Häufig von einem anderen gefolgt
- Hier: Ordne String entsprechend neu an, sodass ähnliche Buchstaben zusammen stehen. Ermöglicht einfachere Kodierungen.

Gegeben: Eingabealphabet Σ , String $S \in \Sigma^*$ der Länge N

1. Transformation: $S \rightarrow (L, I)$, L ist Permutation von S , $I \in \{0, \dots, N - 1\}$.
2. Kompression von (L, I) (Kein spezifisches Verfahren erforderlich)
3. Dekompression von (L, I)
4. Rücktransformation $(L, I) \rightarrow S$

Transformation mit Beispiel

Beispieltext $S = \text{'abraca'}$ ($N = 6$, $\Sigma = \{\text{'a'}, \text{'b'}, \text{'c'}, \text{'r'}\}$)

Erstelle $N - 1$ Rotationen (Leftshifts). Die erste Zeile ist S :

a	b	r	a	c	a	
b	r	a	c	a		a
r	a	c	a	a		b
a	c	a	a	b		r
c	a	a	b	r		a
a	a	b	r	a		c

Transformation

Sortiere Rotationen lexikographisch in neue Matrix M :

	L					
0	a	a	b	r	a	c
1	a	b	r	a	c	a
2	a	c	a	a	b	r
3	b	r	a	c	a	a
4	c	a	a	b	r	a
5	r	a	c	a	a	b

l ist der Zeilenindex mit dem Originaltext.

Ausgabe: (L, l) (Im Beispiel ('caraab', 1))

Wieso Preprocessing?

L ist sehr wahrscheinlich gut zu komprimieren. Mehrinformation durch I vernachlässigbar.

- z.B. *the* kommt sehr oft vor (englische Texte)
- in M dann viele Zeilen mit '*he...t*' hintereinander
- somit lange '*t*'-Folgen in L

→ L effizient zu komprimieren

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $SL[i]$ to be the
i	n turn, set $SR[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector SR
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with sch appear in the {\em same order
i	n with sch . In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell\cite{bell}.

Rücktransformation

Input: (L, I) (hier: $(\text{'caraab'}, 1)$, insbesondere ist nicht die ganze Matrix verfügbar!)

	F					L
0	a	a	b	r	a	c
1	a	b	r	a	c	a
2	a	c	a	a	b	r
3	b	r	a	c	a	a
4	c	a	a	b	r	a
5	r	a	c	a	a	b

- Beobachtung: Jede Spalte ist Permutation von S
- Konstruiere F durch Sortieren von L : $F = \text{'aaabcr'}$

Rücktransformation

M :

	F						L
0	a	a	b	r	a	c	
1	a	b	r	a	c	a	
2	a	c	a	a	b	r	
3	b	r	a	c	a	a	
4	c	a	a	b	r	a	
5	r	a	c	a	a	b	

M' (einen Schritt nach rechts rotiert):

	L	F				
0	c	a	a	b	r	a
1	a	a	b	r	a	c
2	r	a	c	a	a	b
3	a	b	r	a	c	a
4	a	c	a	a	b	r
5	b	r	a	c	a	a

Rücktransformation

M :

	F						L
0	a	a	b	r	a	c	
1	a	b	r	a	c	a	
2	a	c	a	a	b	r	
3	b	r	a	c	a	a	
4	c	a	a	b	r	a	
5	r	a	c	a	a	b	

M' (einen Schritt nach rechts rotiert):

	L	F				
0	c	a	a	b	r	a
1	a	a	b	r	a	c
2	r	a	c	a	a	b
3	a	b	r	a	c	a
4	a	c	a	a	b	r
5	b	r	a	c	a	a

M' ist lexikografisch nach dem *zweiten* Zeichen sortiert

→ Alle Zeilen, die mit dem selben Zeichen starten, sind in der selben Reihenfolge wie in M

Rücktransformation

M :

	F						L
0	a	a	b	r	a	c	
1	a	b	r	a	c	a	
2	a	c	a	a	b	r	
3	b	r	a	c	a	a	
4	c	a	a	b	r	a	
5	r	a	c	a	a	b	

M' (einen Schritt nach rechts rotiert):

	L	F				
0	c	a	a	b	r	a
1	a	a	b	r	a	c
2	r	a	c	a	a	b
3	a	b	r	a	c	a
4	a	c	a	a	b	r
5	b	r	a	c	a	a

M' ist lexikografisch nach dem *zweiten* Zeichen sortiert

→ Alle Zeilen, die mit dem selben Zeichen starten, sind in der selben Reihenfolge wie in M

→ Zuweisung der Einträge von L auf F (Zeilen von M' auf M) mithilfe vom Vektor T

Rücktransformation

M :

	F						L
0	a	a	b	r	a	c	
1	a	b	r	a	c	a	
2	a	c	a	a	b	r	
3	b	r	a	c	a	a	
4	c	a	a	b	r	a	
5	r	a	c	a	a	b	

M' (einen Schritt nach rechts rotiert):

	L	F				
0	c	a	a	b	r	a
1	a	a	b	r	a	c
2	r	a	c	a	a	b
3	a	b	r	a	c	a
4	a	c	a	a	b	r
5	b	r	a	c	a	a

M' ist lexikografisch nach dem *zweiten* Zeichen sortiert

→ Alle Zeilen, die mit dem selben Zeichen starten, sind in der selben Reihenfolge wie in M

→ Zuweisung der Einträge von L auf F (Zeilen von M' auf M) mithilfe vom Vektor T

→ $T = (4, 0, 5, 1, 2, 3)$

Rücktransformation

$L = ('c', 'a', 'r', 'a', 'a', 'b')$

$l = 1$

$T = (4, 0, 5, 1, 2, 3)$

M :

	F					L
0	a	a	b	r	a	c
1	a	b	r	a	c	a
2	a	c	a	a	b	r
3	b	r	a	c	a	a
4	c	a	a	b	r	a
5	r	a	c	a	a	b

Implementationsdetails:

- Rücktransformation ist einfach umsetzbar ($\mathcal{O}(N \log N)$, Sortierung von L)
- Transformation: Wie kann $\mathcal{O}(N^2 \log N)$ vermieden werden?

Suffix Arrays

Implementationsdetails:

- Rücktransformation ist einfach umsetzbar ($\mathcal{O}(N \log N)$, Sortierung von L)
- Transformation: Wie kann $\mathcal{O}(N^2 \log N)$ vermieden werden?

Beobachtung: Rotation schiebt Suffixes nach vorne

a	b	r	a	c	a	
b	r	a	c	a		a
r	a	c	a		a	b
a	c	a		a	b	r
c	a		a	b	r	a
a		a	b	r	a	c

- Kann Suffix-Sorting statt String-Sorting verwendet werden?

Hänge an S ein extra Zeichen an, das nur dort vorkommt und minimal sortiert z.B. $\$$

- Beim ersten Vorkommen von $\$$ ist lexikographischer Vergleich eindeutig

```
a b r a c a $  
b r a c a $  
r a c a $  
a c a $  
c a $  
a $
```

```
a $ c  
a b r a c a $  
a c a $ r  
b r a c a $ a  
c a $ a  
r a c a $ b
```

- Benutze Suffix-Arrays
- Sortieren in $\mathcal{O}(N \log N)$
- Suffix-Array speichert F
- Für L einfache modulo Operation