Datenkompression: Weitere verlustfreie Komprimierungsverfahren

H. Fernau

email: fernau@uni-trier.de

WiSe 2008/09
Universität Trier

Weitere verlustfreie Verfahren:

- Lauflängencodierung
- Facsimile-Übertragung
- BWT
- Fortschreitende Bildübertragung

Erinnerung: **Lauflängencodierung**

Sind (insbesondere in Schwarz-Weiß-Bildern) lange Sequenzen gleicher Zeichen zu erwarten, so empfiehlt sich die *Lauflängencodierung*:

Neben dem Code für ein Zeichen (der evtl. per Huffman-Codierung erzeugt wurde) wird mitgeteilt, wie oft sich das Zeichen wiederholt. Bei Bildern sind oft ganze Bereiche schwarz oder weiß.
$\leadsto$ Es gibt lange Ketten von Nullen oder Einsen.

Ist bekannt, dass lange Läufe nur eines bestimmten Zeichens a zu erwarten sind, so kann man Lauflängencodierung für a mit anderen Codierverfahren für die übrigen Zeichen mischen.

Anwendung: Facsimile Codierung (Faxen)

Aufgabe: Taste eine A4-Seite ab und komprimiere das Schwarz-Weiß-Bild, um es durch die Telefonleitung zu senden.

Wir beschreiben zwei Methoden:

das 1-dimensionale Schema **MH** (*modified Huffman*) und

das 2-dimensionale Schema **MR** (*modified READ* — *Relative Element Address Designate*).

MH Algorithmus

Wir codieren mit Hilfe der Huffman-Codierung die *Lauflängen*.

Lauf (engl. *run*): die längste Abfolge des gleichen Pixels.

Bestimme die Länge für jede Zeile getrennt.

Wir nehmen (o.E.) an, dass die erste Zahl den Lauf ein weißes Pixel codiert.

Die Anzahl der Pixel pro Zeile: 1728 (groß!)
Um die Huffman-Methode möglichst gut zu nutzen, codieren wir jede Länge $r_l > 64$:

$$r_l = 64 \times m + t \qquad \text{for} \quad m = 1, 2, \ldots, 27 \quad \text{und} \quad t = 0, 1, \ldots, 63$$

und stellen r_l als Paar $64 \times m$ und t dar.
Codes für $64 \times m$: *Laufcode*s (engl.: make-up codes) und
Codes für t: *Schlusscode*s (engl.: terminating codes).

Falls $r_l < 63$, benutzen wir nur den Schlusscode. Dann konstruieren wir den Huffman-Code für das Alphabet

$$\{64 \times 1,\ 64 \times 2,\ \ldots,\ 64 \times 27\} \cup \{0, 1, \ldots, 63\}$$

für die weißen Läufe und *separat* für das gleiche Alphabet

$$\{64 \times 1,\ 64 \times 2,\ \ldots,\ 64 \times 27\} \cup \{0, 1, \ldots, 63\}$$

für die schwarzen Läufe.

In dem CCITT-Standard (*Consultative Committee on International Telephone and Telegraph*, jetzt *ITU-T: International Telecommunications Union*) hat beispielsweise der **schwarze Schlusscode für 2** die Länge **2** und **für 63 die Länge 12**. Für die weißen Schlusscodes sind diese Längen: **4 bzw. 8** (es ist sehr wahrscheinlich, dass der s. Lauf die Länge 2 hat!)

MR Algorithmus

a_0: Das ist das Pixel, das wir gerade betrachten. Wir nehmen an, dass jede Zeile mit einem weißen Lauf anfängt (wenn nicht, betrachten wir den weißen Lauf der Länge 0).

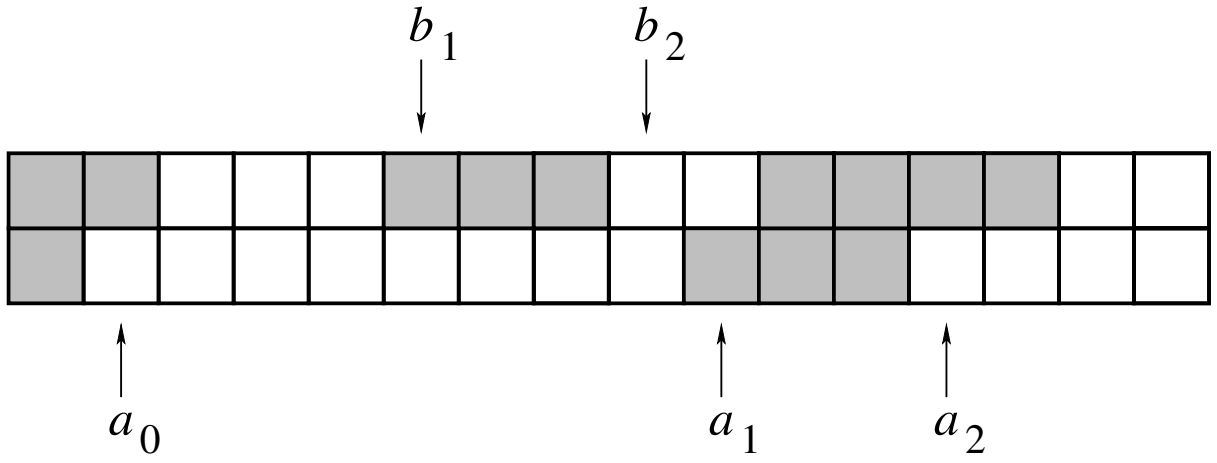
a_1: Das erste Übergangspixel rechts von Pixel a_0.

a_2: Das zweite Übergangspixel rechts von Pixel a_0.

b_1: Das erste Übergangspixel in der vorherigen Zeile, das eine andere Farbe als Pixel a_0 hat.

b_2: Das erste Übergangspixel rechts von Pixel b_1.

Das Codierungsverfahren kennt alle Positionen.
Wenn wir decodieren, kennen wir nur b_1, b_2 und a_0, um das Ziel a_1 und a_2 zu finden.



Der Codier-/Decodier-Algorithmus betrachtet drei Fälle:

1. *Zwischen-Modus*: b_1 und b_2 liegen zwischen a_0 und a_1. Der Code ist 0001. Dann nimm Position $b_2 + 1$ als neuen Wert für a_0, suche neue Werte b_1 und b_2 und wiederhole den ganzen Prozess.

2. *Vertikal-Modus*: a_1 liegt vor b_2.

   2.1 Abstand zwischen a_1 und b_1 ≤ 3. Dann sind es folgende Codes:

Pos. a_1 bezüglich b_1	Code	Länge des Codes
b_1		
a_1	1	1
$b_1 a_1$	011	3
$b_1 \sqcup a_1$	000011	6
$b_1 \sqcup \sqcup a_1$	0000011	7
$a_1 b_1$	010	3
$a_1 \sqcup b_1$	000010	6
$a_1 \sqcup \sqcup b_1$	0000010	7

   2.2 Abstand zwischen a_1 und b_1 > 3. Dann sieht der Code folgendermaßen aus: zuerst 001 und dann MH-Codes für zwei Läufe von a_0 nach $a_1 - 1$ und von a_1 nach $a_2 - 1$.

Burrows-Wheeler Compression Algorithm (BWCA)
von M. Burrows und D. Wheeler (1994)

Der Kern des Algorithmus ist die *Burrows-Wheeler Transformation* (BWT), auch *Blocksortierung* genannt: sie ist keine Kompression per se; Ziel: Zeichen mit ähnlichem Kontext nahe beieinander anzuordnen.

Zweite Stufe: *globale Struktur-Transformation* (GST).
Die lokalen Kontexte der BWT-Ausgabe werden in einen globalen Kontext überführt (*Zeichen* $\rightarrow$ *Index*). Für diese Stufe existieren eine ganze Reihe von unterschiedlichen Verfahren (in Arbeit von B/W '94: move-to-front (MTF))

Dritte Stufe: *Entropie-Codierung* (EC).

BWT lässt sich aber grundsätzlich mit vielen anderen Verfahren kombinieren.

Burrows-Wheeler-Transformation (BWT)

Algorithm C: Transformation (Codierung)

Eingabe: $S = S[0]S[1]\ldots S[N-1] \in \Sigma^N$.
Ausgabe: (L, I), wobei $L \in \Sigma^N$ eine Permutation von S ist und $0 \leq I < N$.
Beispiel: $S =$`abraca`, $N = 6$, $\Sigma = \{$`a,b,c,r`$\}$.

C1. [Sortiere Verschiebungen]
Bilde eine *konzeptuelle* $N \times N$ Matrix $\bar{M}$; jede Zeile von $\bar{M}[i]$ ist das Wort S, um i Zeichen nach rechts verschoben ($i = 0, 1, \ldots, N-1$). Dann sortiere die Zeilen von $\bar{M}$ in *lexikographischer* Ordnung $\rightsquigarrow$ Matrix M.
I sei der erste Index mit $M[I] = S$.
Im Beispiel: Die Matrix $\bar{M}$ bzw. M vor und nach der Sortierung; $I = 1$.

0	a b r a c a		0	a a b r a c	
1	b r a c a a		1	a b r a c a	← erste Zeile
2	r a c a a b	⇒	2	a c a a b r	
3	a c a a b r		3	b r a c a a	
4	c a a b r a		4	c a a b r a	
5	a a b r a c		5	r a c a a b	

C2. [Lies letzte Spalte aus]
$L = M[0, N-1]M[1, N-1]\ldots M[N-1, N-1]$. Liefere (L, I) zurück.
Beispiel: $L =$`caraab`, $I = 1$.

Algorithm D: Transformation (Decodierung)

Eingabe: (L, I) (Ausgabe von Algorithmus C).
Ausgabe: Das wiederhergestellte Wort S.

D1. [Berechne die erste Spalte F von M]
Erhalte F durch Sortieren von L. Beispiel: $F =$`aaabcr`.

D2. [Konstruiere Liste von Vorgängerzeichen]
Bem.: *Für diesen Schritt werden nur L, F und I gebraucht.*
Bezeichne M' die Matrix, die man aus M erhält, wenn man jede Zeile zyklisch um ein Zeichen nach rechts verschiebt:

$$M'[i,j] = M[i,(j-1) \mathrm{mod} N]$$

für alle $i, j \in \{0, 1, .., N-1\}$. In unserem Beispiel sind M und M':

	M	M'
0	a a b r a c	c a a b r a
1	a b r a c a	a a b r a c
2	a c a a b r	r a c a a b
3	b r a c a a	a b r a c a
4	c a a b r a	a c a a b r
5	r a c a a b	b r a c a a

D2. (Forts.)
Die Zeilen in M' sind aufsteigend gemäß ihrem zweiten Zeichen (etc.) sortiert.
Betrachten wir nur die Zeilen in M', die mit demselben Zeichen anfangen, erscheinen sie (untereinander) lexikographisch sortiert.

M'

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

D2. (Forts.)
Für jedes Zeichen x erscheinen die Zeilen in M, die mit x beginnen, daher in derselben Reihenfolge wie die Zeilen von M', die mit x beginnen.

Im Beispiel: die Zeilen `aabrac`, `abraca`, und `acaabr` sind Zeilen $0, 1, 2$ in M und Zeilen $1, 3, 4$ in M'.

	M	M'
0	a a b r a c	c a a b r a
1	a b r a c a	a a b r a c
2	a c a a b r	r a c a a b
3	b r a c a a	a b r a c a
4	c a a b r a	a c a a b r
5	r a c a a b	b r a c a a

Die Zeilen `aabrac`, `abraca`, and `acaabr` tragen Nummern $0, 1, 2$ in M und entsprechen Zeilennummern $1, 3, 4$ in M'.

D2. (Forts.)
Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$.

D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Es folgen Einzelheiten und Beispiele !

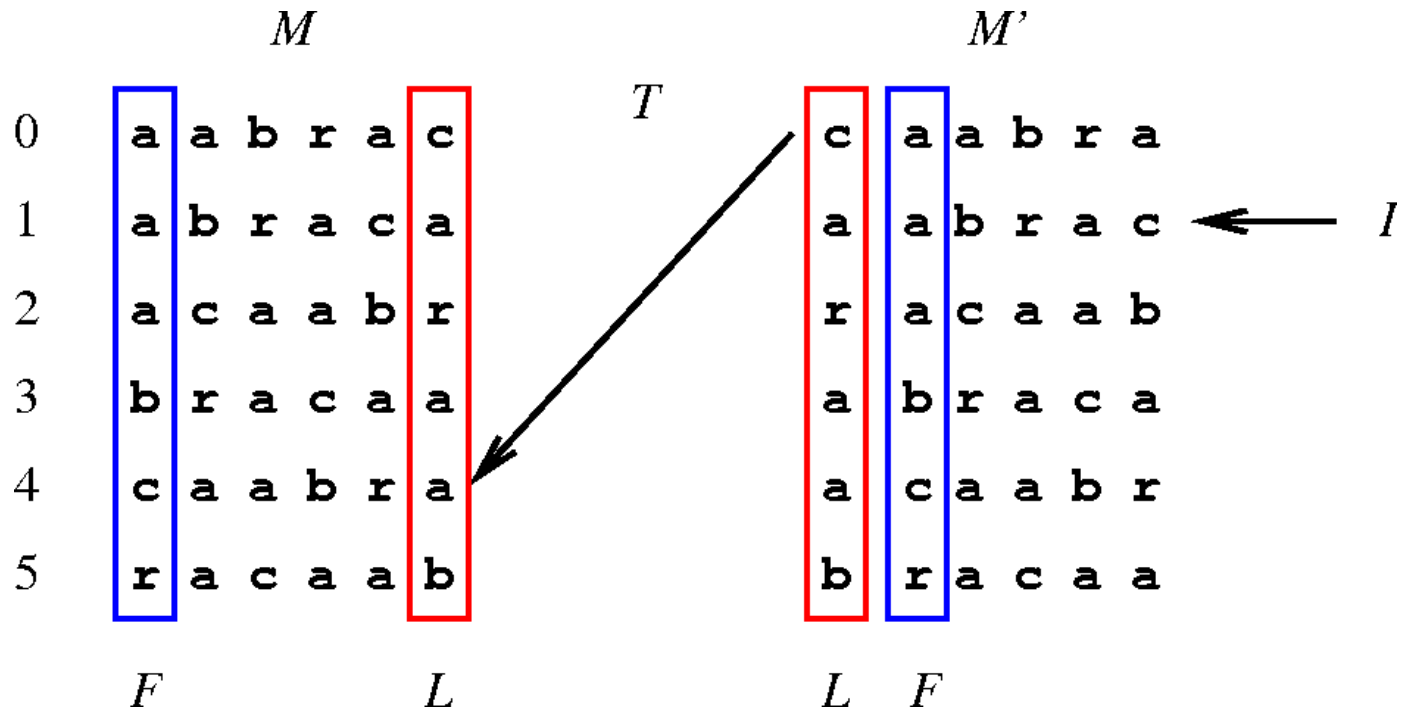
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[0] = M[4]$. Warum ?



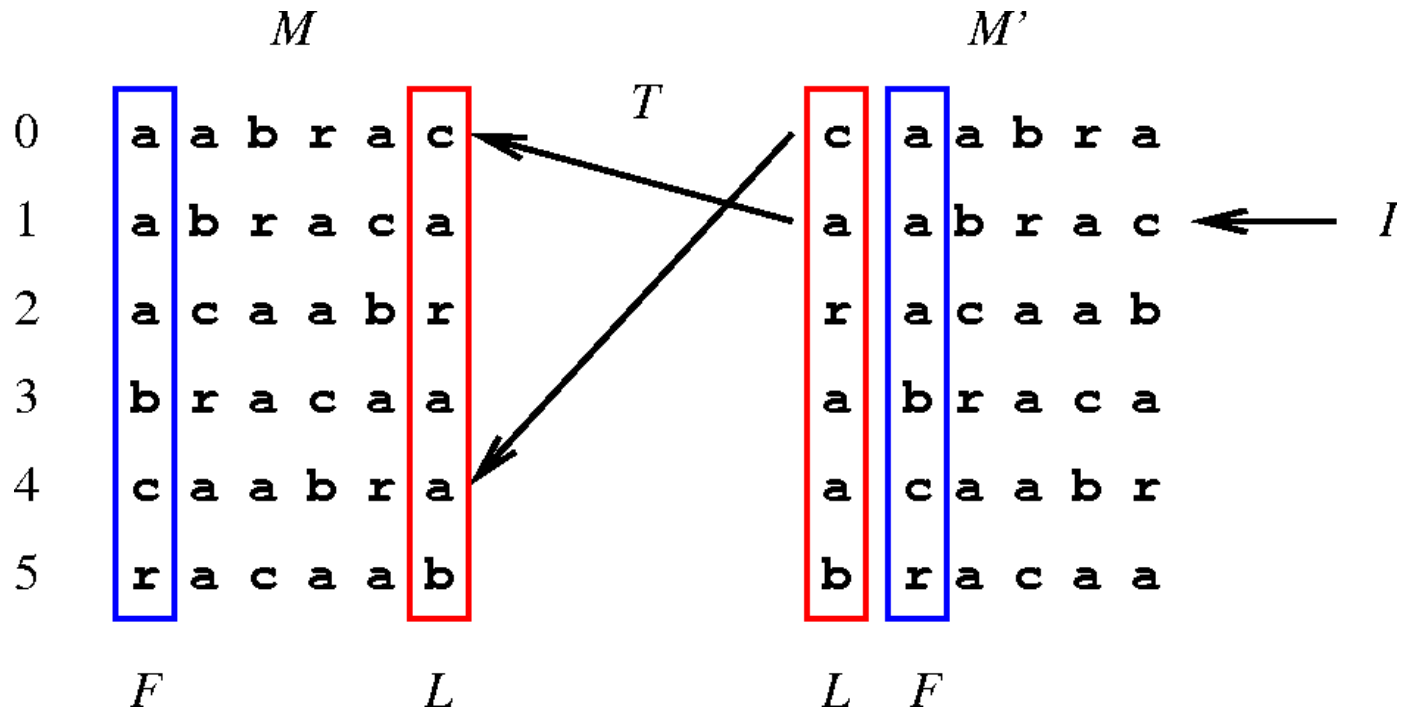
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[1] = M[0]$. Warum ?



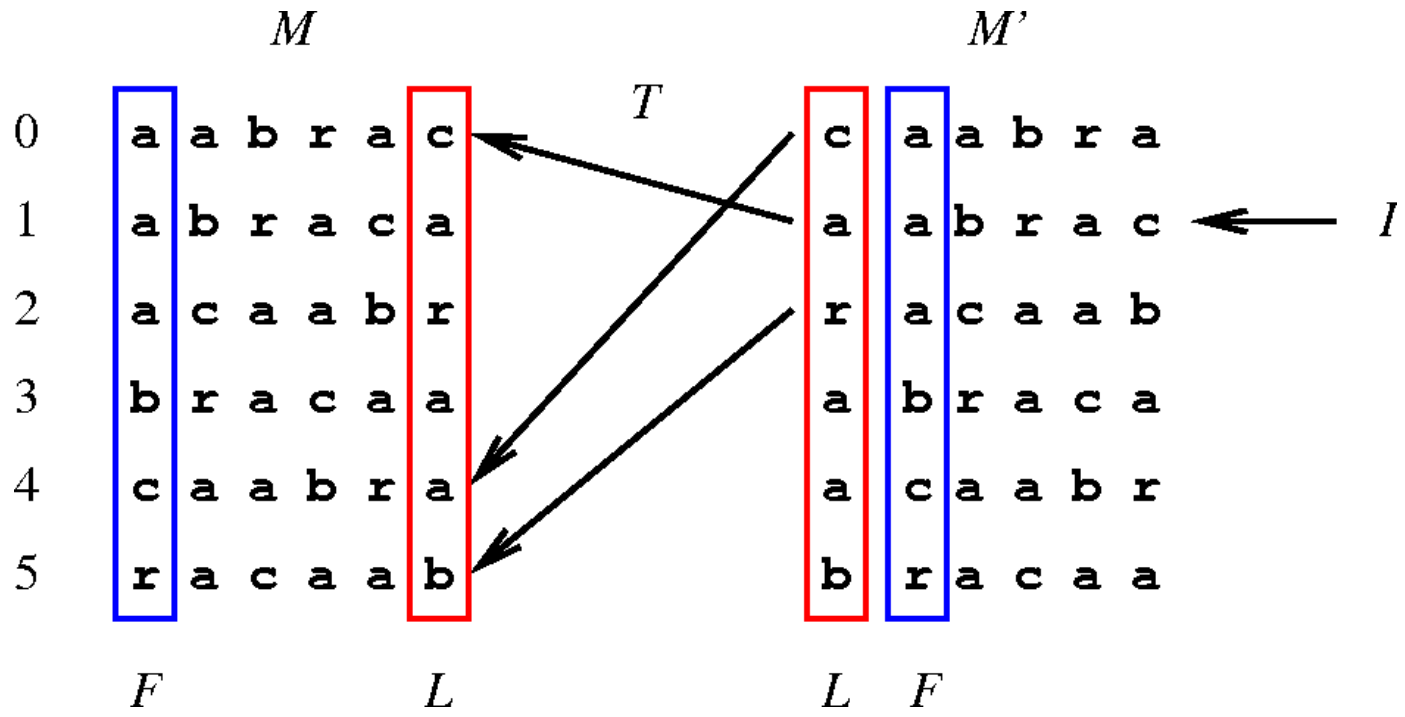
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[2] = M[5]$. Warum ?



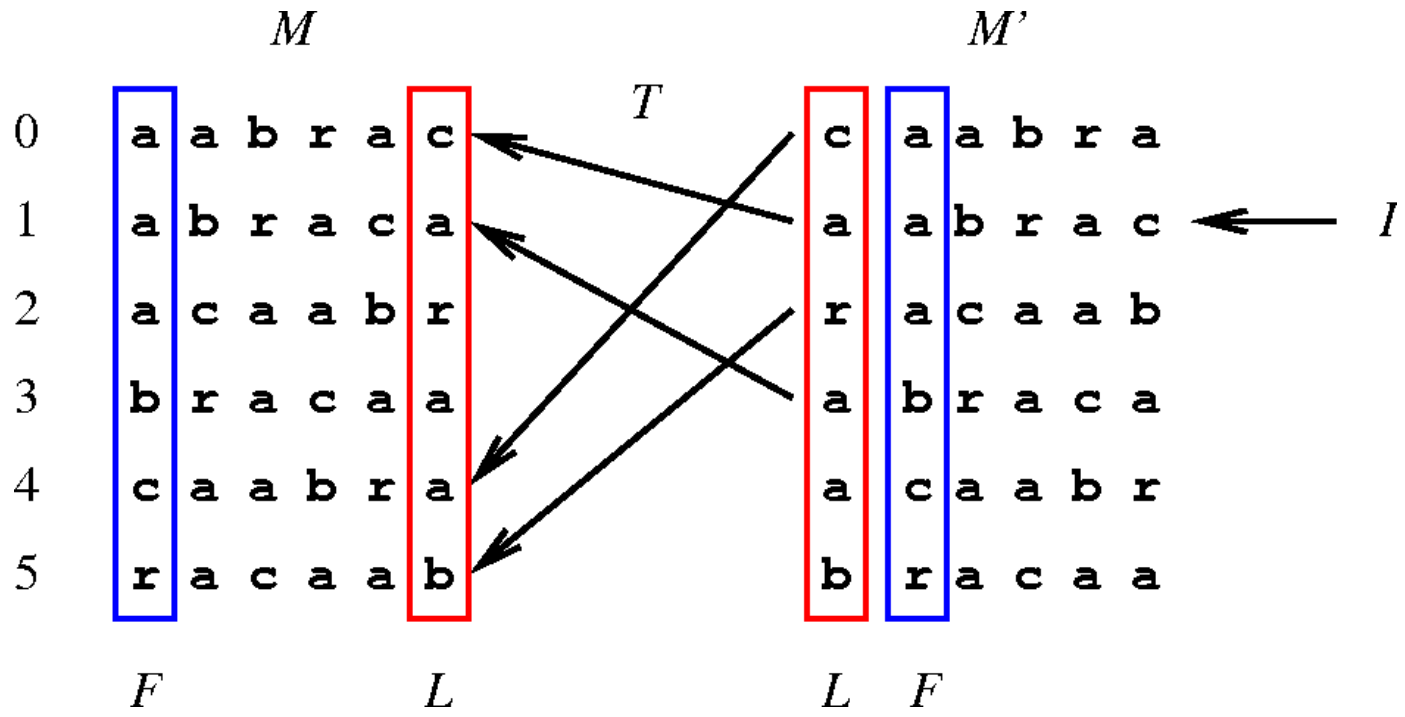
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[3] = M[1]$. Warum ?



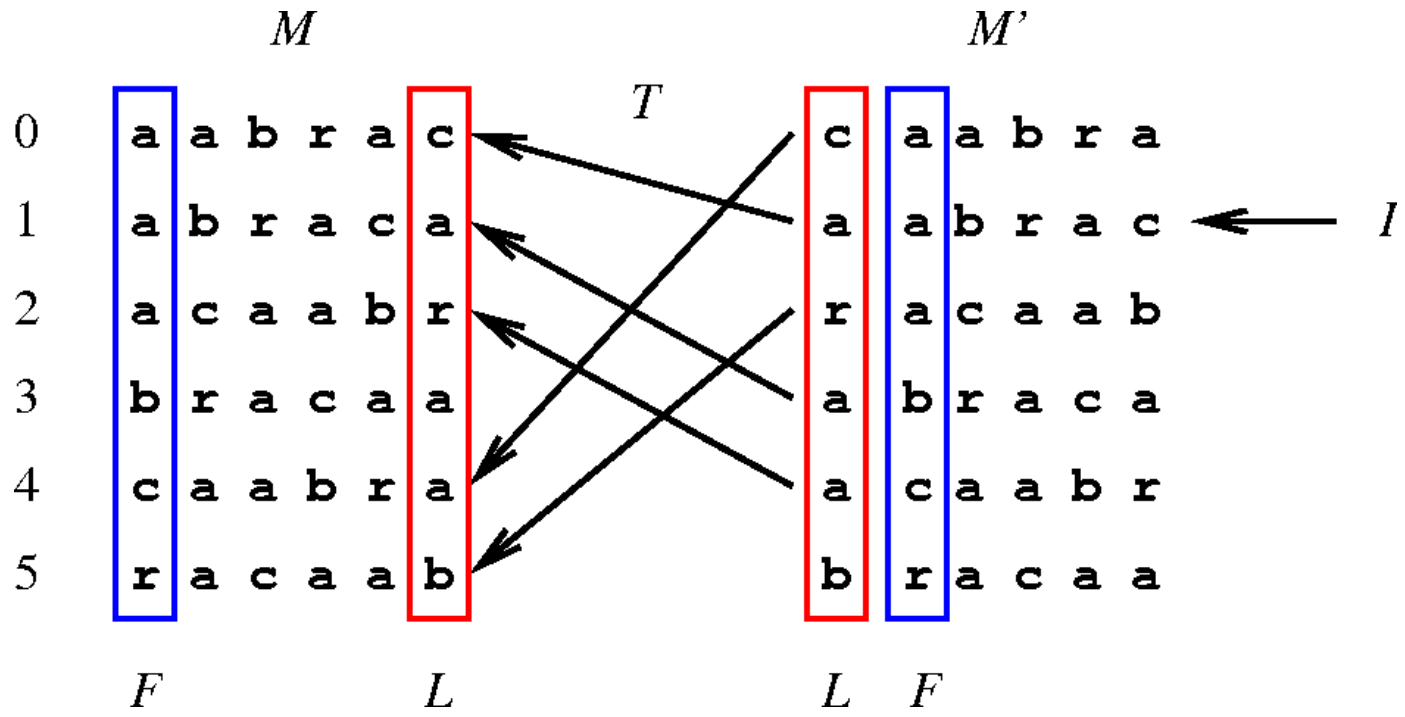
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[4] = M[2]$. Warum ?



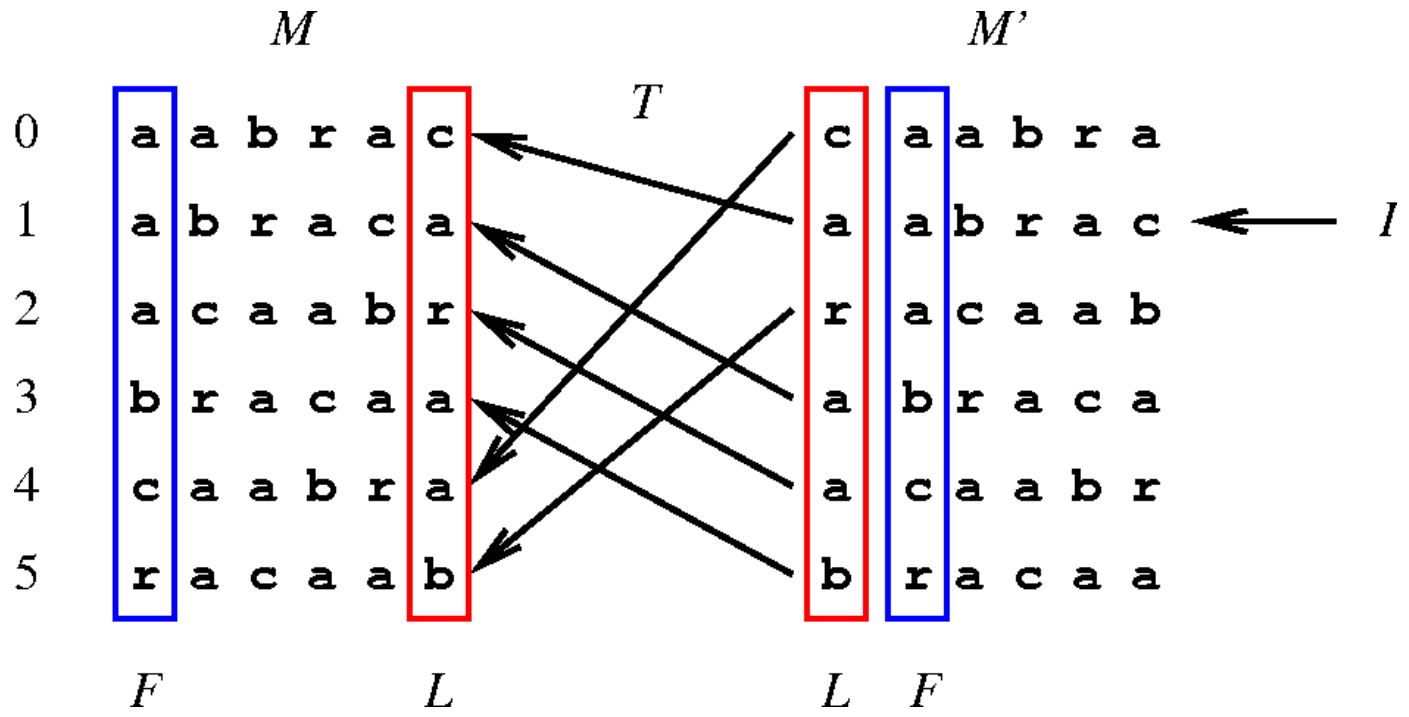
D2. (Forts.)

Mit F und L berechnet man einen Vektor T, der die Korrespondenz von den Zeilen von M und M' ausdrückt:

$$\forall j \in \{0, 1, .., N-1\} \quad M'[j] = M[T[j]]$$

Ist $L[j]$ das k-te Vorkommen von Zeichen x in $L \rightsquigarrow$
$T[j] = i$, wobei $F[i]$ das k-te Vorkommen von x in F ist.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $M'[5] = M[3]$. Warum ?

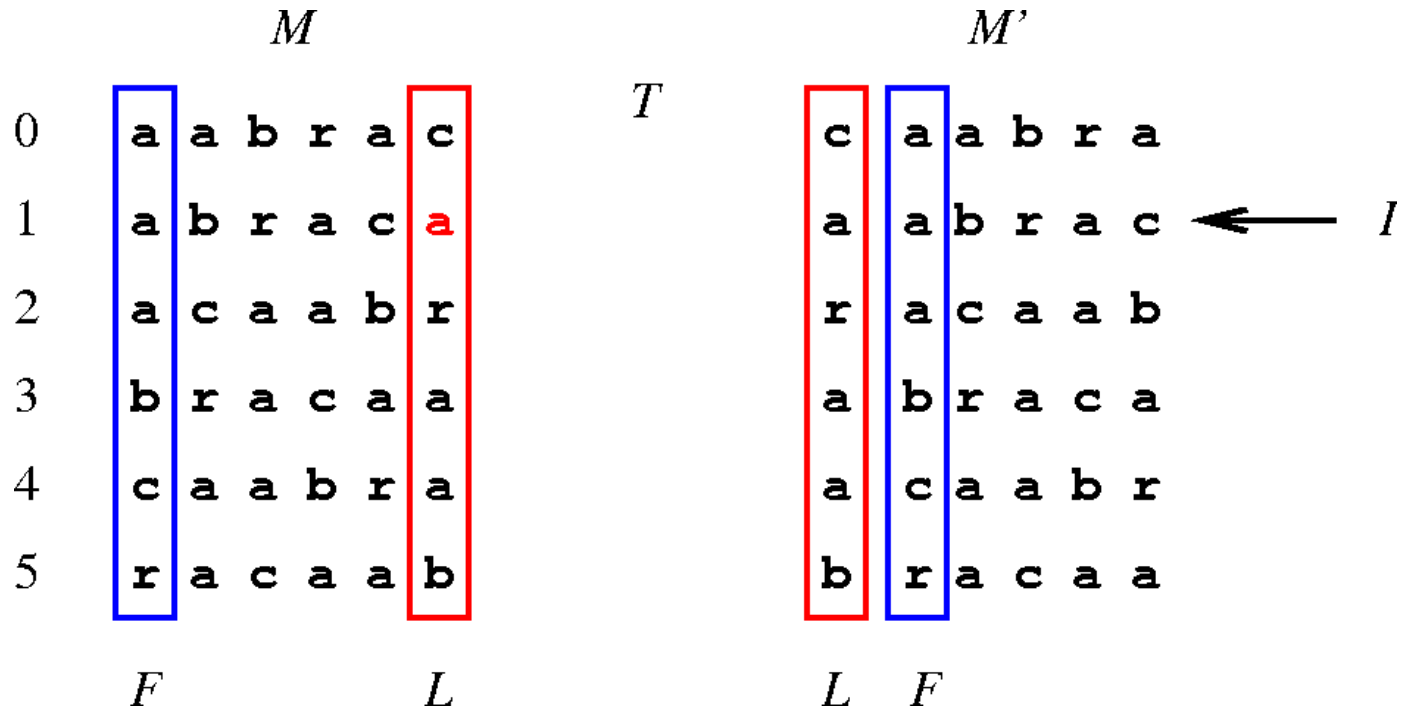


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^0[I] = 1$.

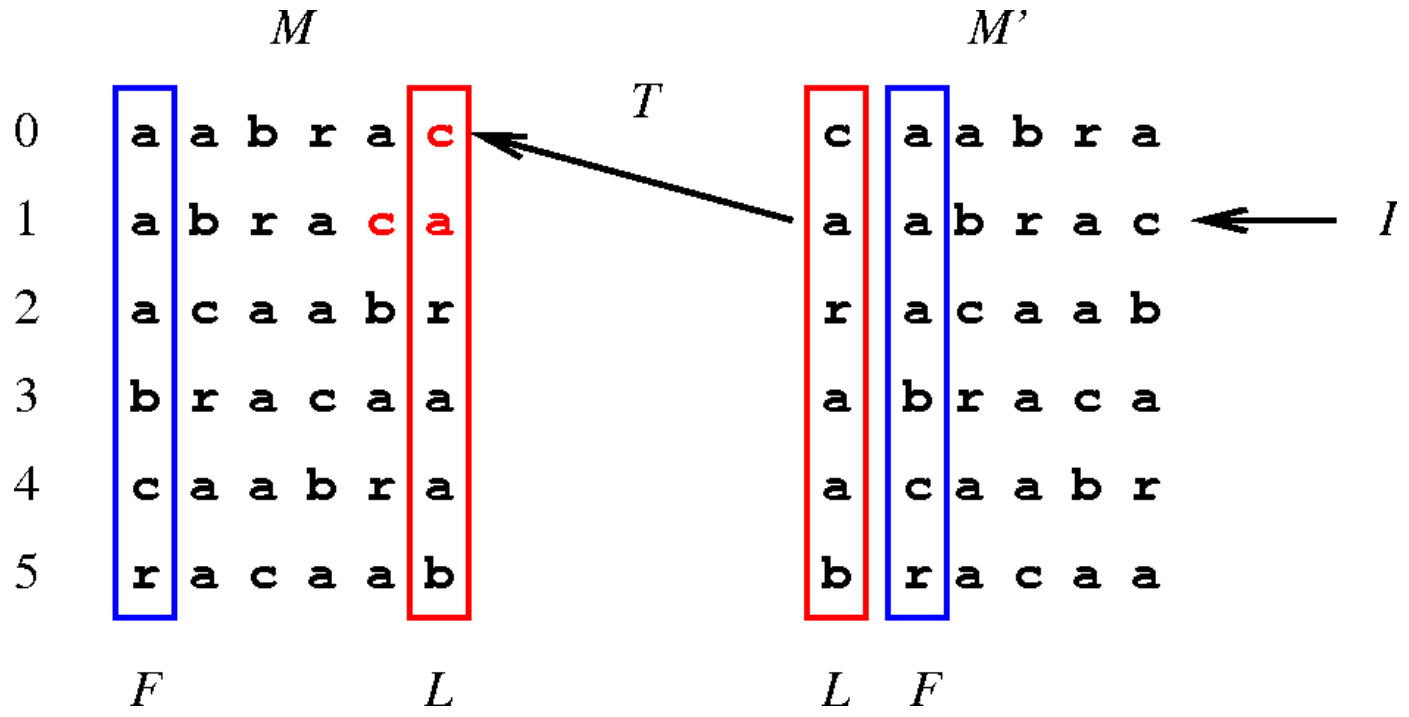


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^1[I] = T[1] = 0$.

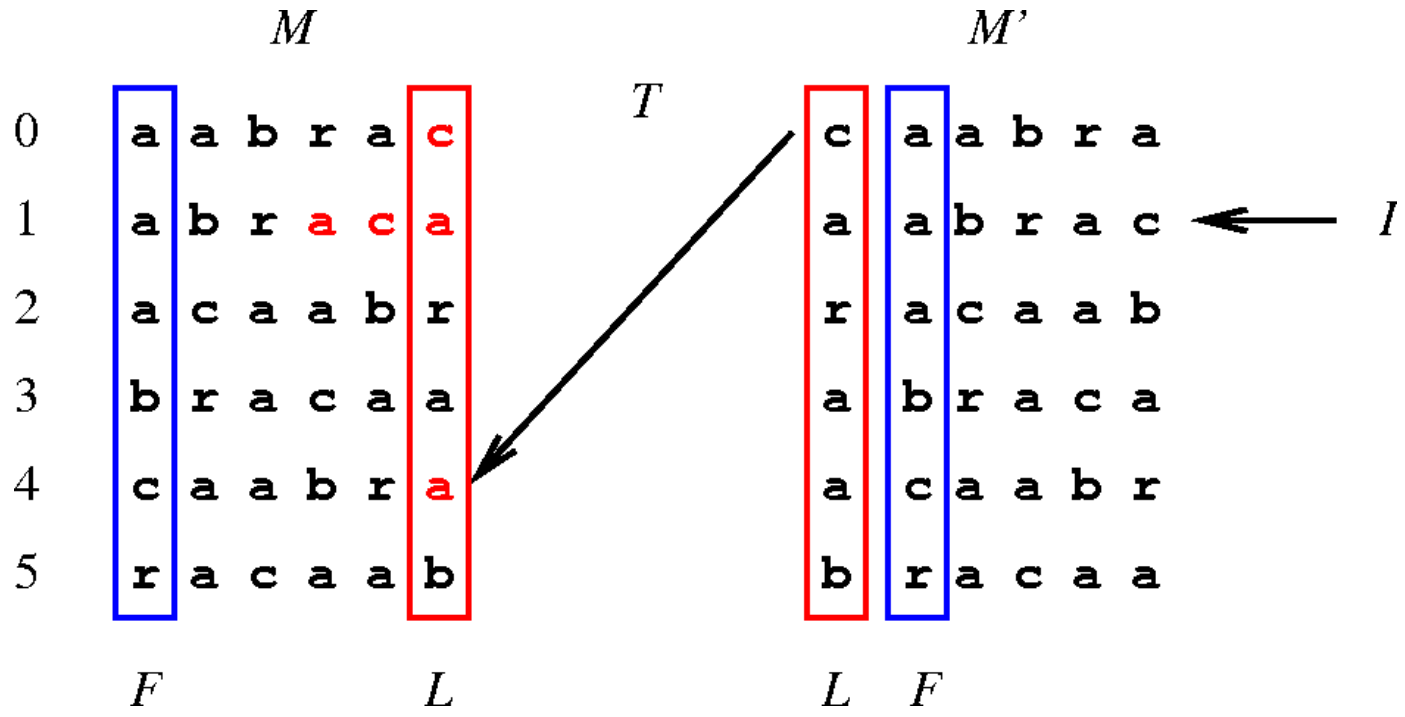


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^2[I] = T[T^1[I]] = T[0] = 4$.

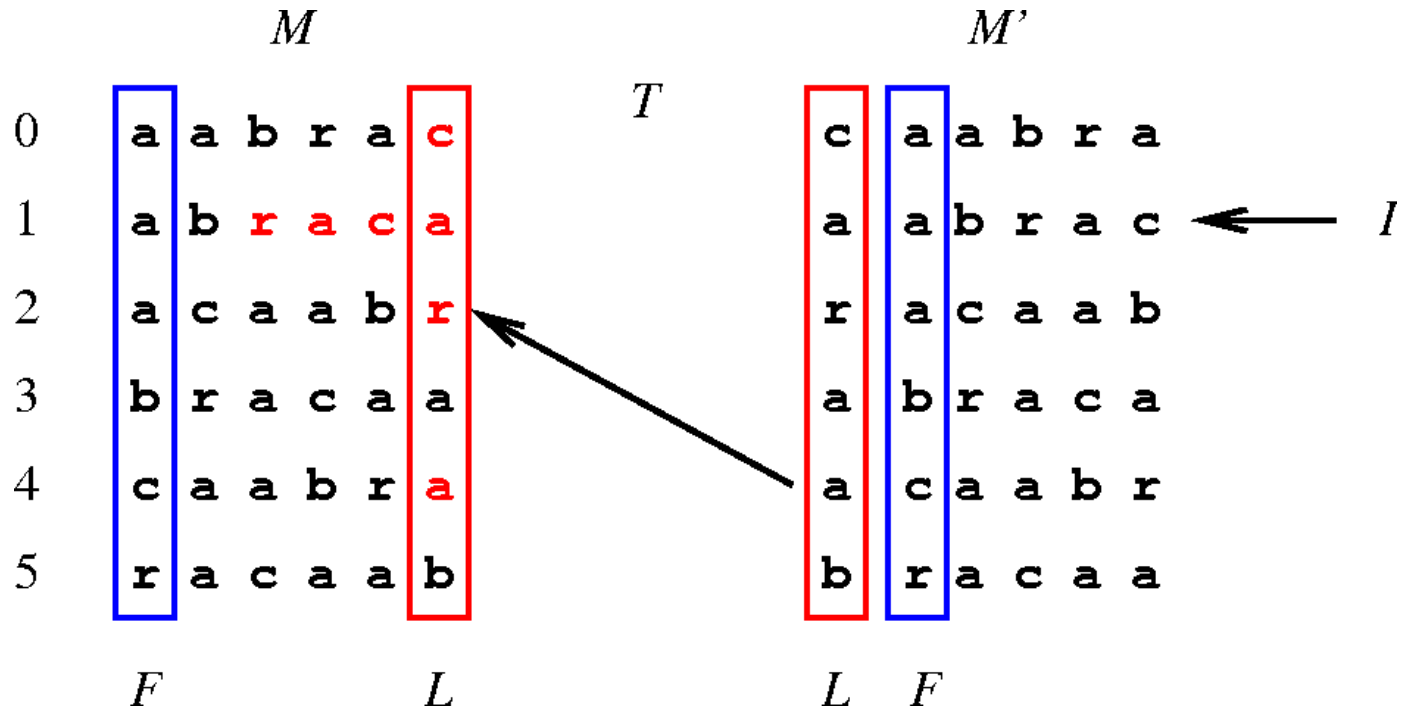


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^3[I] = T[T^2[I]] = T[4] = 2$.

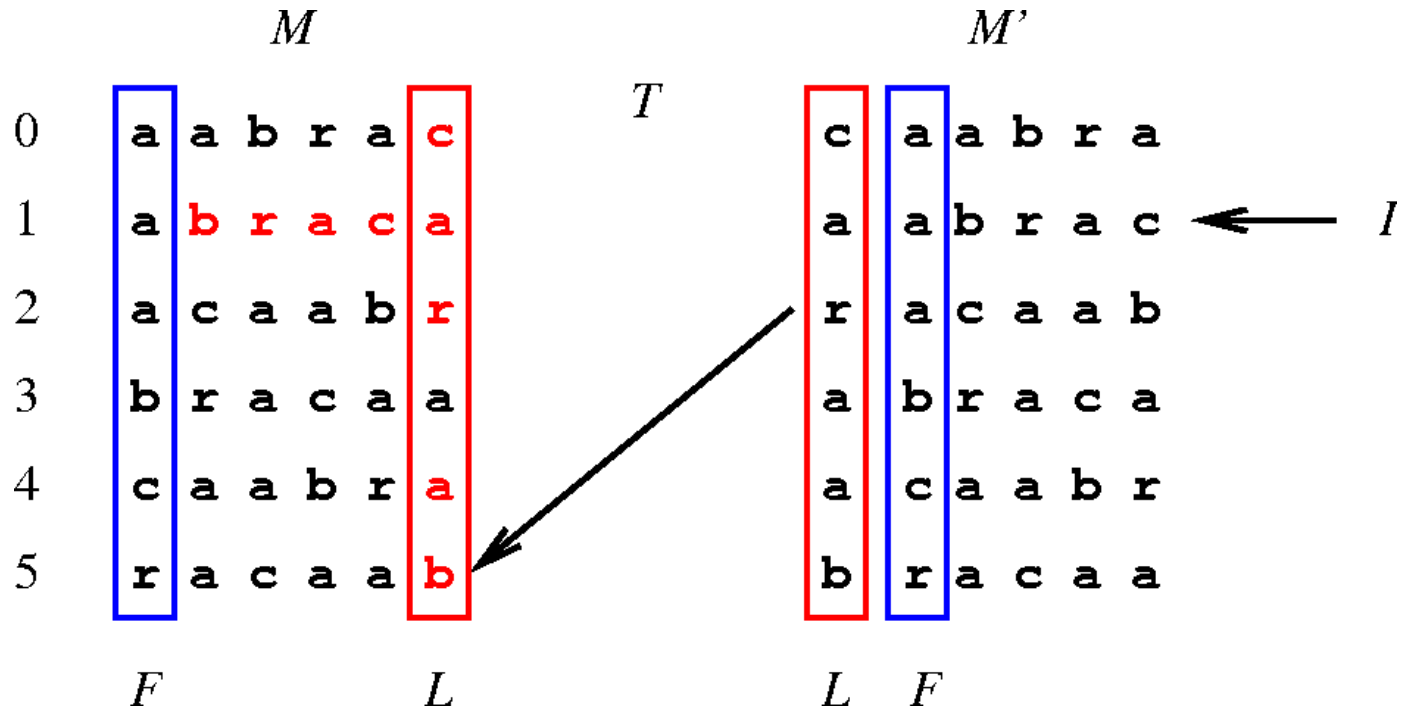


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^4[I] = T[T^3[I]] = T[2] = 5$.

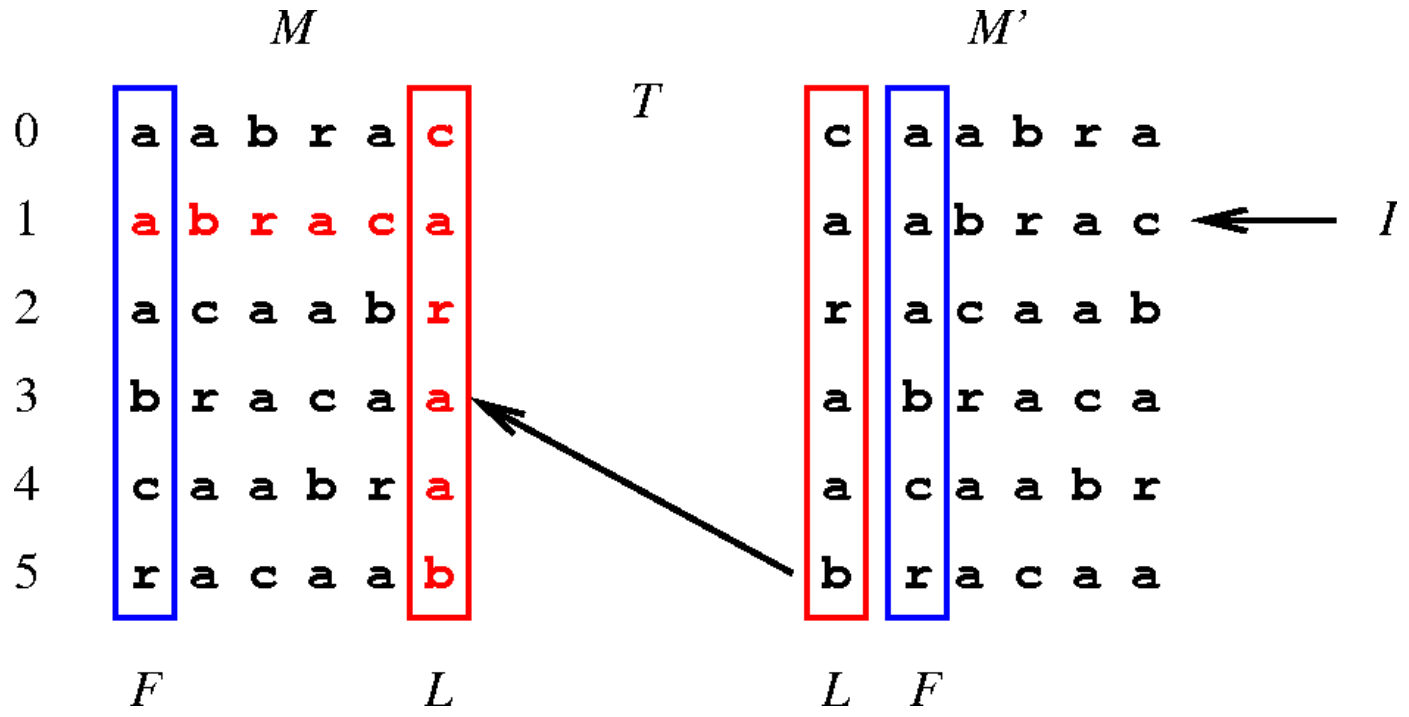


D3. [Berechne Ausgabe S]

`for` i `= 0,1,..,`$N-1$ `do` $S[N-1-i] = L[T^i[I]]$;

hierbei ist $T^0[k] = k$ und $T^{i+1}[k] = T[T^i[k]]$.

Im Beispiel: $T = (4, 0, 5, 1, 2, 3)$. $T^5[I] = T[T^4[I]] = T[5] = 3$.



Warum funktioniert BWT gut? Beispiel für *sorted rotations*: wir führen BWT-Algorithmus aus mit der Beispieleingabe: *M. Bläser, A. Jakoby, M. Liskiewicz, and B. Siebert, Privacy in Non-Private Environments*. (Ausschnitt!)

L	*sortierte Verschiebungen*
t	he network G consists of d blo
t	he network consists of three o
t	he network have to compute som
t	he network, then it is possibl
t	he networks we consider are i
T	he next result says that sG(q,
t	he nodes vj 2 VG adjacent to t
t	he notion of lossy private pro
t	he notion of privacy by introd
t	he notion of privacy: One cann
t	he notion of private protocols
t	he number of bits a player lea
t	he number of bits exchanged do
t	he number of bits known by the
t	he number of communication seq
t	he number of communication seq
t	he number of different communi
t	he number of different probabi
t	he number of different probabi

L	*sortierte Verschiebungen*
s	ume that A and B are complete
s	ume that Ai knows theinput bit
s	ume that each subsequence (may
s	ume that the enumeration of th
s	umed to be honestbut curios. T
s	umed to be prefix-free. In thi
s	umed, then the self-informatio
n	umer-ation of all bridge playe
n	umerates the blocks of G accor
n	umeration of all communica-tio
n	umeration of allstrings descri
n	umeration of the blocks reflec
j	umping function pj : Our netwo
s	umptions. Private Computation
f	unction 1-phase protocols can
f	unction and B be a two-party c
f	unction can be computed Xi nw
f	unction can be computed at lea
f	unction can be computed using

Globale Struktur-Transformation (GST) & Entropie-Codierung (EC)

Algorithm M

Eingabe: Ausgabe (L, I) von Algorithmus C.
Ausgabe: Codierung von (L, I).

M1. [GST: move-to-front coding]
Für die Menge $\Sigma = \{\sigma_0, \sigma_1, \ldots \sigma_{K-1}\}$ initialisiere die Liste Y mit

$Y[0, .., K-1] = (\sigma_0, \sigma_1, \ldots, \sigma_{K-1})$;
`for` i `= 0,1,..,`$N-1$ `do`
$R[i] =$ ‘k `mit` $Y[k] = L[i]$’;
`move` $L[i]$ `to front of` Y.

M2. [EC-Codierung]
Codiere R mit Huffman oder arithmetischer Codierung.

Decodierung

Algorithm W

Eingabe: die Ausgabe von Algorithmus M.
Ausgabe: das Paar (L, I).

W1. [EC-Decodierung]
Decodiere R je nach Wahl der EC-Codierung.

W2. [mache GST rückgängig]
Initialisiere die Liste Y mit

$Y[0,..,K-1] = (\sigma_0, \sigma_1, \ldots, \sigma_{K-1})$;
`for` i `= 0,1,..,`$N-1$ `do`
$L[i] = Y[R[i]]$;
`move` $L[i]$ `to front of` Y.

Implementierung

M. Nelson. *Data Compression with the Burrows–Wheeler Transform*, Dr. Dobbs' Journal, 21(9):46–50 1996.
`http://www.dogma.net/markn/articles/bwt/bwt.htm`

`bzip2`: J. Seward. *The bzip2 and libbzip2 official home page*,
`http://sources.redhat.com/bzip2/`.

`szip`: M. Schindler. *The szip home page*,
`http://www.compressconsult.com/szip/`.

Vergleich der Methoden

Huffman-Codierung	`pack`, `compact`, ...
Arithmetische-Codierung	`JBIG standard`, `JPEG`, ...
ppm	`ppmC`, `ppmD`, ...
LZ77	`gzip`, `ZIP`, ...
LZ78	`GIF`, Unix `compress`, ...
Burrows-Wheeler	`bzip`, `szip`, ...
Dynamic Markov Compression	`dmc`, ...

Der Calgary Corpus Entwickelt Ende der 1980er Jahre, wurde später der de facto Standard für die Beurteilung verlustfreier Komprimierung (`http://corpus.canterbury.ac.nz`)

File	Abbrev	Category	Size
bib	bib	Bibliography (refer format)	111261
book1	book1	Fiction book	768771
book2	book2	Non-fiction book (troff format)	610856
geo	geo	Geophysical data	102400
news	news	USENET batch file	377109
obj1	obj1	Object code for VAX	21504
obj2	obj2	Object code for Apple Mac	246814
paper1	paper1	Technical paper	53161
paper2	paper2	Technical paper	82199
pic	pic	Black and white fax picture	513216
progc	progc	Source code in C	39611
progl	progl	Source code in LISP	71646
progp	progp	Source code in PASCAL	49379
trans	trans	Transcript of terminal session	93695

Der Canterbury Corpus wurde 1997 als Fortentwicklung des Calgary Corpus vorgestellt. Die Auswahl der Dateien erfolgte im Hinblick darauf, dass das Verhalten (damals) existierender Komprimierungsverfahren *typische* Ergebnisse hierfür lieferte (`http://corpus.canterbury.ac.nz`).

File	Abbrev	Category	Size
alice29.txt	text	English text	152089
asyoulik.txt	play	Shakespeare	125179
cp.html	html	HTML source	24603
fields.c	Csrc	C source	11150
grammar.lsp	list	LISP source	3721
kennedy.xls	Excl	Excel Spreadsheet	1029744
lcet10.txt	tech	Technical writing	426754
plrabn12.txt	poem	Poetry	481861
ptt5	fax	CCITT test set	513216
sum	SPRC	SPARC Executable	38240
xargs.1	man	GNU manual page	4227

Ergebnisse für den *Calgary Compression Corpus*

File Name	Size (bytes)	Bits/Byte		
		PKZIP	BTW[2]	SZIP-P
bib	111,261	2.58	2.07	1.96
book1	768,771	3.29	2.49	2.35
book2	610,856	2.74	2.13	2.02
geo	102,400	5.38	4.45	4.28
news	377,109	3.10	2.59	2.48
obj1	21,504	3.84	3.98	3.77
obj2	246,814	2.65	2.64	2.47
paper1	53,161	2.80	2.55	2.49
paper2	82,199	2.90	2.51	2.43
pic	513,216	0.84	0.83	0.80
progc	39,611	2.69	2.58	2.50
progl	71,646	1.81	1.80	1.72
progp	49,379	1.82	1.79	1.77
trans	93,695	1.68	1.57	1.54
total:	3,141,622	2.63	2.18	2.07

Ähnliche Daten finden Sie auch in M. Nelsons Arbeit.

Literatur

1. R. Arnold and T. Bell. *The Canterbury corpus home page* `http://corpus.canterbury.ac.nz/`
2. M. Burrows and D.J. Wheeler. *A Block-sorting Lossless Data Compression Algorithm*, Digital Systems Research Center Research Report 124. Vergleiche auch:
   Dr. Dobb's Journal Veröffentlichung von M. Nelson; enthält auch Programmcode zum Experimentieren.
   `http://www.dogma.net/markn/articles/bwt/bwt.html`.
3. K. Sayood. *Introduction to Data Compression,* Morgan Kaufmann Publishers, Inc., Second Edition, 2000.

Fortschreitende Bildübertragung

Beispiel 1 *Mit Hilfe des Browsers wollen wir* 30 *Bilder durchsehen. Nehmen wir an, dass jedes Bild schwarz-weiß ist und dass es die Größe*

$$512 \times 512 \text{ mit } 8 \text{ Bits je Pixel}$$

hat. Wir wollen diese Bilder über eine Leitung mit der Geschwindigkeit

$$14.4\text{-kbit/s}$$

übertragen und das heißt, dass die ganze Übertragung

$$30 \times (8 \times 512 \times 512) \;/\; 14\,400 \;=\; 62\,914\,560 \;/\; 14\,400 \;\approx\; 4\,369 \text{ s} \;\approx\; 73 \text{ min.}$$

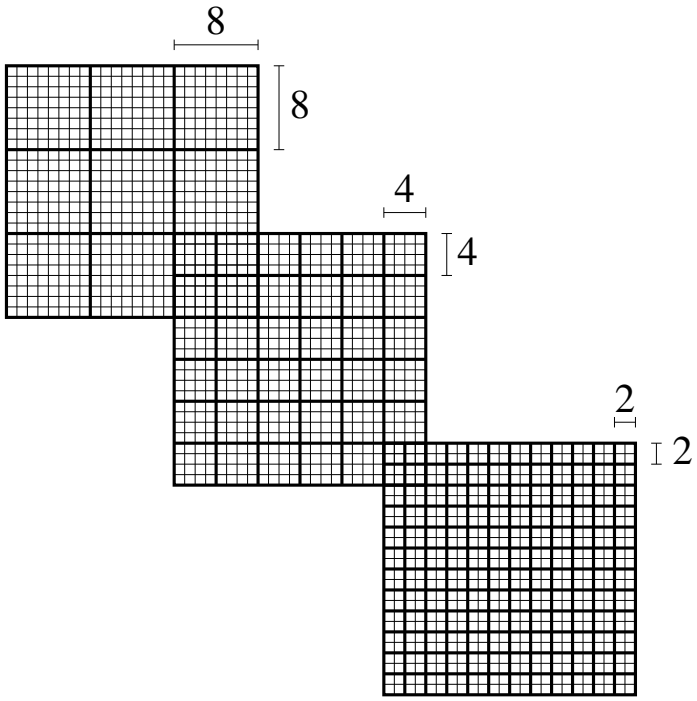
dauert.

Eine Modifikation:
statt der Bilder eine Approximationen der Bilder übertragen.

Ein einfaches und schnelles Verfahren:

Wir betrachten $b \times b$ Pixels des Bildes und dann stellen wir alle b^2 Pixel als ein Pixel dar – wir komprimieren mit dem Faktor b^2.

Um das Bild zu übertragen, senden wir z.B. drei Approximationen (mit den Parametern $b = 8$, 4 und 2 wie unten) und die verlustfreie Codierung des Bildes nacheinander.



Vergleich Approximation und Original Unten können Sie vergleichen, welche Approximationen wir bekommen, wenn wir (für $b = 4$) als Repräsentant für die $b \times b$ Pixel das im Quadrat

- links oben gelegene (linkes Bild) oder
- den Medianwert aller b^2 Pixel (rechtes Bild)

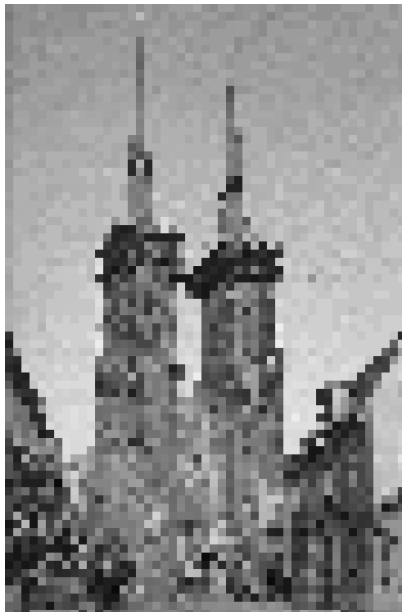
nehmen.

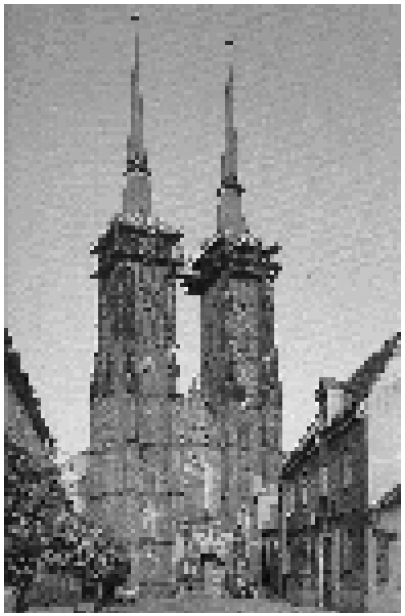
Vergleich Approximation und Original

Nun das Original:

Fortschreitende Bildübertragung vs. verlustfreie Übertragung Zeile für Zeile

Die linken Approximationen haben die gleiche Größe wie die entsprechenden Teile des Originals auf der rechten Seite, $b = 8$.





$b = 4$