Kompression von Zeichenreihen

Wir betrachten nur verlustfreie Komprimierungen

$$compress : \Sigma^* \longrightarrow \Delta^*$$

die Informationsverlust rückgängig gemacht werden können:

$$decompress(compress((w)) = w.$$

Kompressionsrate: $\frac{|w|}{|compress(w)|}$

(i) Statistische Methoden: berechne das in einem Kontext erwartete Zeichen und kodiere das tatsächliche durch die Differenz zum erwarteten

(ii) Wörterbuchmethoden: ersetze *Blöcke* = Teilwörter B von w durch Codes $c(B) \in \Delta$ aus einem Wörterbuch

$$Dict = \{(B_i, c(B_i)) \mid i = 1, \ldots, k\}.$$

  (a) *statisches Wörterbuch*: *Dict* ist unabhängig von w

  (b) *dynamisches Wörterbuch*: *Dict* wird während der Kompression von w berechnet.

Finde deterministisch eine Zerlegung $w = B_{i_1} \cdots B_{i_n} \in Dict^*$ mit „kurzem“ $c(w)$ für

$$c(w) := c(B_{i_1}) \cdots c(B_{i_n}).$$

Beispiel: Ziv-Lempel-Kompression (Unix: compress, gzip)

LEMPEL-ZIV-77 COMPRESSION

Parameters window size N (≤ 8192), buffer size F (≤ 20)

Input: a text t over alphabet Σ

Output: a sequence of triples $(i, j, c) \in (N - F) \times F \times \Sigma$

(i) a *window* of length N is sliding through the text t,

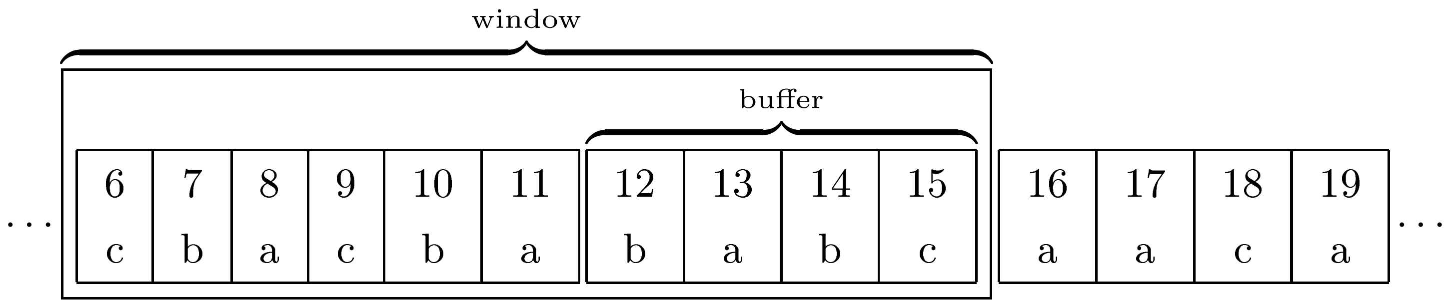
(ii) the final F window characters are a *lookahead buffer*, the first $N - F$ characters have been compressed,

(iii) find the longest prefix u of the buffer that occurs in the window at position $N - F - i$ (may reach into the buffer),

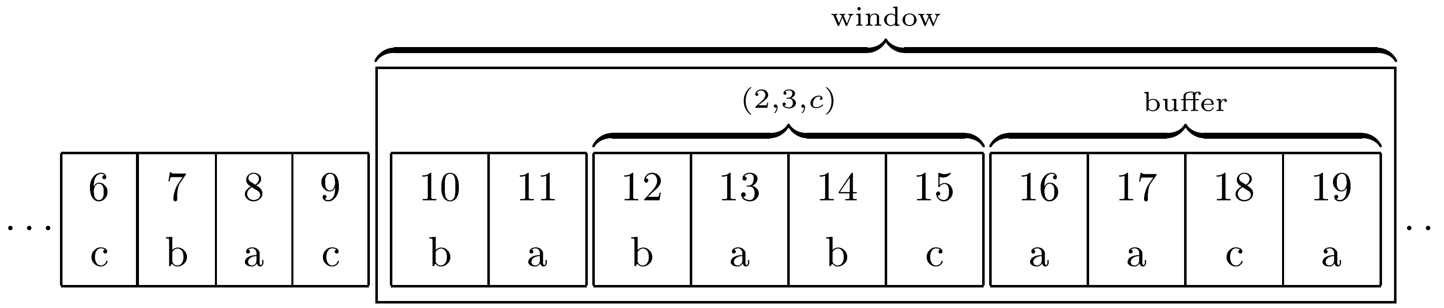
(iv) encode u by $(i, |u|, a)$ where ua is a buffer prefix, but does not occur at window position $N - F - i$,

(v) shift the window by $|u| + 1$ characters.

Expl. 3



Compress $uc = babc$ as $(2, 3, c)$ and consider next:



(i) Fenster und Puffer beschränken die Suche und die Größe der „Zeiger“ i und $|u|$ in $c(B) = c(ua) = (i, |u|, a)$.

(ii) Wörterbuch: Teilwörter der Länge $1 \leq k \leq F$ des Präfixes der Länge $N - 2$ des Fensterinhalts.

Komprimieren

(i) Zustand:

$$Codes \boxed{\;Fenster\;} Resteingabe = p_0 \cdots p_k \boxed{c_6 \ldots c_1 . b_1 \ldots b_4} v$$

(ii) Kodiere $(i, |u|, a)$, mit $ua =$ längstes Präfix des Puffers, wovon u bei Offset $i > 0$ vor dem Puffer beginnt.

(iii) Verschiebe das Fenster um $|ua|$ Positionen.

Beisp. 4 *LZ*-77 mit $N = 10, F = 4$, $w = a^{15}$

$$\begin{aligned}
\boxed{\epsilon^6.\epsilon^4} a^{15} \quad &\Rightarrow \quad \boxed{\epsilon^6.a^4} a^{11} \\
&\Rightarrow \quad (1,0,a) \boxed{\epsilon^5 a.a^4} a^{10} \\
&\Rightarrow \quad (1,0,a)(1,3,a) \boxed{\epsilon a^5.a^4} a^6 \\
&\Rightarrow \quad (1,0,a)(1,3,a)(1,3,a) \boxed{a^6.a^4} a^2 \\
&\Rightarrow \quad (1,0,a)(1,3,a)(1,3,a)(1,3,a) \boxed{a^6.a^2} \\
&\Rightarrow \quad (1,0,a)(1,3,a)(1,3,a)(1,3,a)(1,1,a)
\end{aligned}$$

Beachte: a^F wird zu $(1, 0, a)(1, F - 1, a)$ kodiert, also exponentiell kürzer bei log.Darstellung von $F - 1$.

Dekomprimieren

Zustand: $\mathit{Dekodiert}\,(i,m,a)\ \mathit{Codes} = u(i,m,a)\cdots(i_k,m_k,a_k)$:

(i) i Schritte zurück in u gehen;

(ii) von dort m Buchstaben abschreiben, dann a;

(iii) restliche $\mathit{Codes} = \cdots(i_k,m_k,a_k)$ dekodieren.

Beisp. 3 *LZ*-77 mit $N = 10, F = 4$, $w = a^{15}$

$$\begin{array}{ll}
\multicolumn{2}{l}{(1,0,a)(1,3,a)(1,3,a)(1,3,a)(1,1,a)} \\
\Rightarrow & a.(1,3,a)(1,3,a)(1,3,a)(1,1,a) \\
\Rightarrow & a.aaaa.(1,3,a)(1,3,a)(1,1,a) \\
\Rightarrow & a.aaaa.aaaa.(1,3,a)(1,1,a) \\
\Rightarrow & a.aaaa.aaaa.aaaa.(1,1,a) \\
\Rightarrow & a.aaaa.aaaa.aaaa.aa
\end{array}$$

Beisp. 4 *LZ*-77-code $(N = 10, F = 4)$ of $w = abbbaabbabbb$:

	B_0	B_1	B_2	B_3	B_4
$w =$	a	b	bba	abb	$abbb$
$LZ\text{-}77(w_1) =$	$(1,0,a)$	$(1,0,b)$	$(1,2,a)$	$(5,3,a)$	$(3,3,b)$

LEMPEL-ZIV-78 COMPRESSION

For $w \in \Sigma^+$, the *LZ-block decomposition of* w is the sequence $B_0, \ldots, B_m$ of subwords such that

(i) $B_0 =$ first letter of w,

(ii) if $w = B_0 \cdots B_{n-1} v$ for some $v \in \Sigma^+$, then

$$B_n := \begin{cases} \text{shortest non-empty prefix of } v \\ \text{that is not in } \{B_0, \ldots, B_{n-1}\}, & \text{if this exists,} \\ \\ v, & \text{else.} \end{cases}$$

Note: The set of *LZ*-blocks of w is closed under prefixes.

The *LZ-78-compression* $LZ(w)$ of w is the sequence $p_0 \cdots p_{m-1}$ of block representations $p_n = (k, a) \in \mathbb{N} \times \Sigma$ such that

$$\begin{aligned} p_n = (0, a) \quad &\Longleftrightarrow \quad B_n = a, \\ p_n = (j+1, a) \quad &\Longleftrightarrow \quad B_n = B_j a. \end{aligned}$$

Expl. 5 *LZ*-78-compression of $w = a^{15}$:

	B_0	B_1	B_2	B_3	B_4
$w =$	a	aa	aaa	$aaaa$	$aaaaa$
$LZ(w) =$	$(0, a)$	$(1, a)$	$(2, a)$	$(3, a)$	$(4, a)$.

Expl. 6 *LZ*-78-compression of $w = abbbaabbabbb$:

	B_0	B_1	B_2	B_3	B_4	B_5
$w =$	a	b	bb	aa	bba	bbb
$LZ(w) =$	$(0, a)$	$(0, b)$	$(2, b)$	$(1, a)$	$(3, a)$	$(3, b)$.

LZ-compressed strings as coloured ordered graphs

We represent $LZ(w) = (j_0, b_0) \cdots (j_{m-1}, b_{m-1})$ by the graph

$$\mathcal{G}_{LZ(w)} = (m, <, U_{a_1}, \ldots, U_{a_l}, E)$$

where

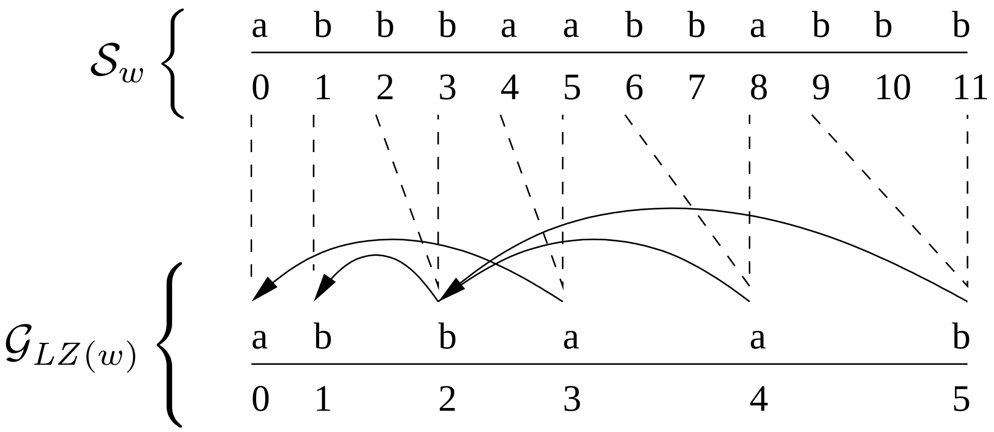
(i) $(m, <, U_{a_1}, \ldots, U_{a_l}) = \mathcal{S}_{b_0 \cdots b_{m-1}}$,

(ii) $E := \{(n, j) \mid p_n = (j+1, b_n)\} = \{(n, j) \mid B_n = B_j b_n\}$.

Expl. 7 (Cont.) For $w = a.b.bb.aa.bba.bbb.$, with

$$\begin{array}{rcl} LZ(w) & = & (0,a)(0,b)(2,b)(1,a)(3,a)(3,b), \\ & = & \quad p_0 \quad\;\; p_1 \quad\;\; p_2 \quad\;\; p_3 \quad\;\; p_4 \quad\;\; p_5 \end{array}$$

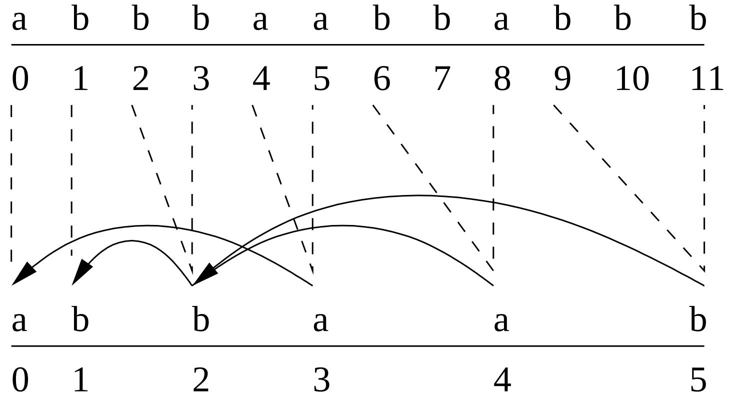
the graph $\mathcal{G}_{LZ(w)}$ is



Interpreting $\mathcal{S}_w$ in $\mathcal{G}_{LZ(w)}$

Embed $\mathcal{S}_w$ in $\mathcal{G}_{LZ(w)}$ by $h : |w| \to |LZ(w)|^2$ where $h(i) = (k, j)$ iff i lies in block B_k and B_j is the prefix of B_k ending in i.

Expl. 8 For $w = a.b.bb.aa.bba.bbb$, the positions 6,7,8 in block $B_4 = bba$ are represented by $(4,1), (4,2), (4,4)$, because the prefixes B_j of B_4 are $b = B_1$, $bb = B_2$, and $bba = B_4$.



Note: $\{j \in |w| \mid j \text{ lies in } B_k\} = \{j \mid \mathcal{G}_{LZ(w)} \models E^*(k, j)\}$.

[ALdR 2000] For every MSO-formula $\varphi(z_1, \ldots, z_n, \vec{X}^{(1)})$ about strings there is a DSO formula $\varphi^{LZ}(x_1, y_1, \ldots, x_n, y_n, \vec{X}^{(2)})$ about LZ-graphs, s.th. for all $\mathcal{S}_w$, $i_1, \ldots, i_n \in \mathcal{S}_w$ and $\vec{S} \subseteq \mathcal{S}_w$,

$$\mathcal{S}_w \models \varphi[\vec{i}, \vec{S}] \iff \mathcal{G}_{LZ(w)} \models \varphi^{LZ}[h(\vec{i}), h(\vec{S})].$$

In DSO, we can define E^*, and then φ^{LZ} using

$$\begin{aligned} (U_a(z_i))^{LZ} &:= U_a(y_i), \\ (z_i \leq z_j)^{LZ} &:= x_i < x_j \vee (x_i = x_j \wedge y_i \leq y_j), \\ (\exists X^1 \varphi)^{LZ} &:= \exists X^2 \subseteq E^* \, \varphi^{LZ}. \end{aligned} \tag{3}$$

Is it *efficient* to evaluate the DSO-formula φ^{LZ} on $\mathcal{G}_{LZ(w)}$?

LZ-compressed strings as enumerated tries

While compressing w, the blocks $B_0, B_1, \ldots \in \Sigma^+$ found are maintained as a tree by sharing common prefixes.

This tree is a Σ-*trie*, i.e. different branches leaving a node are labeled with different letters (if w has a special end symbol).

The linear order of the blocks in $LZ(w)$ amounts to an *enumeration* of the nodes of the tree.

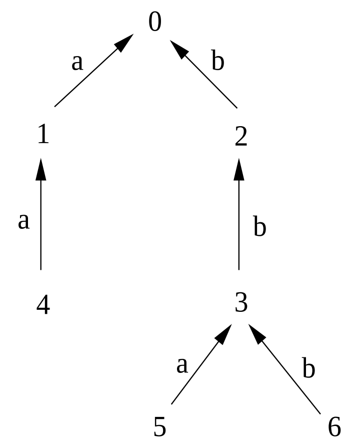
Expl. 9 (Cont.) Enumerating the pairs of

$$LZ(w) = (0,a)(0,b)(2,b)(1,a)(3,a)(3,b),$$

by $1, 2, \ldots$ in a third component gives

$$LZ'(w) = (0,a,1)(0,b,2)(2,b,3)(1,a,4)(3,a,5)(3,b,6),$$

a tree in which B_k labels the path from the root to node $k+1$:



A tuple $p_n = (k, a)$ of $LZ(w)$ is drawn as an edge $k \xleftarrow{a} n+1$.

LZ-Tries

Def. 10 A (finite, edge-labelled) Σ-tree $(T, \leq, \xleftarrow{a}, 0)_{a \in \Sigma}$ is a Σ-*trie* if to each node $n \in T$ and each $a \in \Sigma$ there is at most one $n' \in T$ such that $n \xleftarrow{a} n'$. A (finite) *enumerated* Σ-*trie*

$$\mathcal{T} = (T, \leq, \xleftarrow{a}, 0, \succ)_{a \in \Sigma},$$

or an *LZ-trie* for short, is a Σ-trie $(T, \leq, \xleftarrow{a}, 0)_{a \in \Sigma}$ with a successor relation $\succ$ on T that is compatible with the partial order $\leq$. We assume $T = \{0, 1, 2, \ldots, m\}$ and $\succ (i, j)$ iff $i+1 = j$ in $\mathbb{N}$.

We write $\mathcal{G}_\alpha$ for the LZ-graph and $\mathcal{T}_\alpha$ for the LZ-trie representing the LZ-compressed word $\alpha = LZ(w)$.

$\mathcal{G}_\alpha$ and $\mathcal{T}_\alpha$ are slightly different views of a compessed string α. Modulo the additional root node in the trie,

- $\geq$ in the trie $\mathcal{T}_\alpha$ amounts to E^* in the graph $\mathcal{G}_\alpha$,
- $\leq$ in the graph $\mathcal{G}_\alpha$ to $\succ^*$ in the trie $\mathcal{T}_\alpha$.

Since E^* resp. $\succ^*$ is $\exists$- and $\forall$-MSO-definable in the language of LZ-graphs resp. LZ-tries, we have:

Prop. 11 $\exists$-MSO-properties of LZ-graphs translate to $\exists$-MSO-properties of LZ-tries and vice versa.