# Kompressionsverfahren: Übersicht

- Klassifikationen:
  - Universell vs. speziell (für bestimmte Informationstypen)
  - Verlustfrei vs. verlustbehaftet
  - In diesem Kapitel: nur universelle & verlustfreie Verfahren

- Im folgenden vorgestellte Verfahren:
  - Statistische Verfahren:
    - » Huffman-Codierung
    - » Arithmetische Codierung
  - Zeichenorientierte Verfahren:
    - » Lauflängencodierung (RLE Run Length Encoding)
    - » LZW-Codierung 

# Wörterbuch-Kompressionen

- Grundidee:
  - Suche nach dem „Vokabular“ des Dokuments, d.h. nach sich wiederholenden Teilsequenzen
  - Erstelle Tabelle: Index <--> Teilsequenz („Wort“)
  - Tabelle wird dynamisch während der Kodierung aufgebaut
  - Codiere Original als Folge von Indizes

- Praktische Algorithmen:
  - Abraham Lempel, Jacob Ziv (Israel), Ende 70er-Jahre
    - » LZ77- und LZ78-Algorithmen
  - Verbessert 1984 von A. Welch = „LZW“-Algorithmus (Lempel/Ziv/Welch)
  - Basis vieler semantikunabhängiger Kompressionsverfahren (z.B. UNIX „compress“, Zip, gzip, V42.bis)
  - Verwendet in vielen Multimedia-Datenformaten (z.B. GIF)

# “Wörterbuch” für LZW

- Abbildung von Zeichenreihen auf Zahlen (Code)
- Annahme:
  - Vorbesetzung der Tabelle mit fest vereinbarten Codes für Einzelzeichen (muß nicht explizit gespeichert und übertragen werden)

Startzustand:

| Zeichenreihe | Code |
|---|---|
| “a” | 97 |
| “b” | 98 |
| “c” | 99 |
| … | |
| “z” | 122 |

Während Berechnung:

| Zeichenreihe | Code |
|---|---|
| “a” | 97 |
| … | |
| “z” | 122 |
| “ba” | 256 |
| “an” | 257 |
| … | |

Für “neue” Codes Bereich verwendet, der von den vorbesetzten Codes verschieden ist (z.B. > 255)

# Prinzip der LZW-Codierung

- Nicht alle Teilworte ins Wörterbuch, sondern nur eine "Kette" von Teilworten, die sich um je ein Zeichen überschneiden.
- Sequentieller Aufbau:
  Neu einzutragendes Teilwort =
  Kürzestes ("erstes") noch nicht eingetragenes Teilwort
- Beispiel:

# Aufbau des LZW-Wörterbuchs

Neu einzutragendes Teilwort = Kürzestes ("erstes") noch nicht eingetragenes Teilwort

| Gelesenes Zeichen | Bereits in Wörterbuch? | Neueinträge mit Code |
|---|---|---|
| **b** | "b": Ja | |
| **a** | "b": Ja<br>"ba": Nein | "ba" - neuer Code (256) |
| **n** | "a": Ja<br>"an": Nein | "an" - neuer Code (257) |
| **a** | "n": Ja<br>"na": Nein | "na" - neuer Code (258) |
| **n** | "a": Ja<br>"an": Ja | |
| **e** | … "ane": Nein | "ane" - neuer Code (259) |

b a n a n e . . .

ba | an | na | ane

# LZW-Codierung (1)

Prinzipieller Ablauf:

**SeqChar** *p* = < NächstesEingabezeichen >;
**Char** *k* = NächstesEingabezeichen;
**Wiederhole:**
    **Falls** *p* & < *k* > in Tabelle enthalten
        **dann** *p* = *p* & < *k* >
        **sonst** trage *p* & <*k*> neu in Tabelle ein
            (und erzeuge neuen Index dafür);
            Schreibe Tabellenindex von *p* auf Ausgabe;
            *p* = < *k* >;
    **Ende Fallunterscheidung;**
    *k* = NächstesEingabezeichen;
**solange bis** Eingabeende
Schreibe Tabellenindex von *p* auf Ausgabe;

Achtung:
"alter" Puffer ***p***,
nicht **p&<k>**

# Algorithmus-Beschreibung (“Pseudo-Code”)

- Variablen (ähnlich zu C/Java-Syntax):
  - Datentyp fett geschrieben, gefolgt vom Namen der Variablen
  - Zuweisung an Variable mit “=“
- Datentypen:
  - **int**: Ganze Zahlen
  - **Char**: Zeichen (Buchstaben, Zahlen, Sonderzeichen)
  - **SeqChar**: Zeichenreihen (Sequenzen von Zeichen)
    - » Einelementige Zeichenreihe aus einem Zeichen: < x >
    - » Aneinanderreihung (Konkatenation) mit &
- NächstesEingabezeichen:
  - Liefert nächstes Zeichen der Eingabe und schaltet Leseposition im Eingabepuffer um ein Zeichen weiter

# LZW-Codierung (3)

Beispieltext: “bananenanbau"

Ablauf:

**Wiederhole:**
  **Falls** *p* & < *k* > in Tabelle enthalten
    **dann** *p* = *p* & < *k* >
    **sonst** trage *p* & <*k*> neu in Tabelle ein
      (und erzeuge neuen Index dafür);
      Schreibe Tabellenindex von *p* auf Ausgabe;
      *p* = < *k* >;
  **Ende Fallunterscheidung;**
  *k* = NächstesEingabezeichen;
**solange bis** Eingabeende

| Lesen (k) | Codetabelle schreiben (p & <k>) | Ausgabe | Puffer füllen (p) |
|---|---|---|---|
| | | | <b> |
| a | (<ba>, 256) | 98 | <a> |
| n | (<an>, 257) | 97 | <n> |
| a | (<na>, 258) | 110 | <a> |
| n | | | <an> |
| e | (<ane>, 259) | 257 | <e> |
| n | (<en>, 260) | 101 | <n> |
| a | | | <na> |
| n | (<nan>, 261) | 258 | <n> |
| b | (<nb>, 262) | 110 | <b> |
| a | | | <ba> |
| u | (<bau>, 263) | 256 | <u> |
| EOF | | 117 | |

# LZW-Decodierung bei bekannter Tabelle

**Wiederhole** solange Eingabe nicht leer:

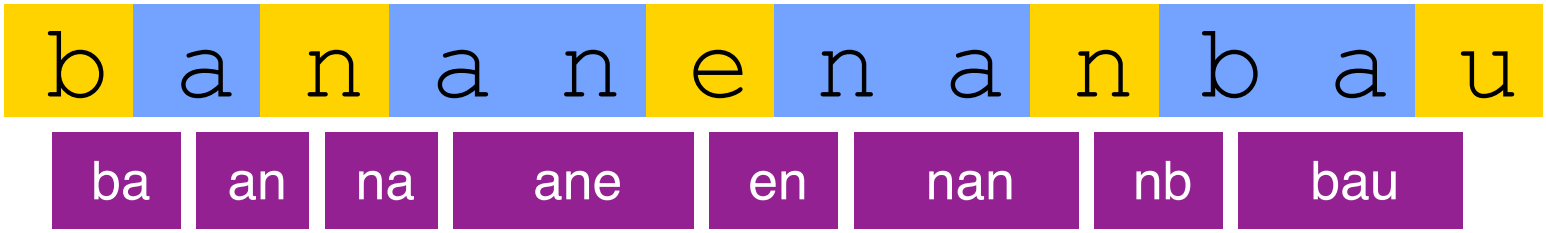
    *k* = NächsteEingabezahl;

    Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;

**Ende Wiederholung**;

# LZW-Decodierung (1)

- Grundidee („symmetrische Codierung“):
  - Das aufgebaute Wörterbuch muß *nicht* zum Empfänger übertragen werden.
  - Das Wörterbuch wird nach dem gleichen Prinzip wie bei der Codierung bei der Decodierung dynamisch aufgebaut.
  - Das funktioniert, weil bei der Codierung immer *zuerst* der neue Eintrag für das Wörterbuch nach bekannten Regeln aus dem schon gelesenen Text aufgebaut wird, bevor der neue Eintrag in der Ausgabe verwendet wird.
- Algorithmusidee:
  - Neu einzutragendes Teilwort = letztes Teilwort plus erstes Zeichen des aktuellen Teilworts



# LZW-Decodierung (2)

Neu einzutragendes Teilwort = letztes Teilwort plus erstes Zeichen des aktuellen Teilworts

Prinzipieller Algorithmus:

**SeqChar** *p* := <>;
**int** *k* = NächsteEingabezahl;
Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;
**int** *old* = *k*;
**Wiederhole** solange Eingabe nicht leer:
    *k* = NächsteEingabezahl;
    **SeqChar** *akt* = Zeichenreihe mit Tabellenindex *k*;
    Schreibe Zeichenreihe *akt* auf Ausgabe;
    *p* = Zeichenreihe mit Tabellenindex *old* (letztes Teilwort);
    **Char** *q* = erstes Zeichen von *akt*;
    Trage *p* & <*q*> in Tabelle ein
    (und erzeuge neuen Index dafür);
    *old* = *k*;
**Ende Wiederholung**;

# LZW-Decodierung (3)

Beispielzeichenreihe:
“98-97-110-257-101-
258-110-256-117"

**Wiederhole** solange Eingabe nicht leer:

*k* = NächsteEingabezahl;

**SeqChar** *akt* = Zeichenreihe mit Tabellenindex *k*;

Schreibe Zeichenreihe *akt* auf Ausgabe;

*p* = Zeichenreihe mit Tabellenindex *old*
(letztes Teilwort);

**Char** *q* = erstes Zeichen von *akt*;

Trage *p* & <*q*> in Tabelle ein
(und erzeuge neuen Index dafür);

*old* = *k*;

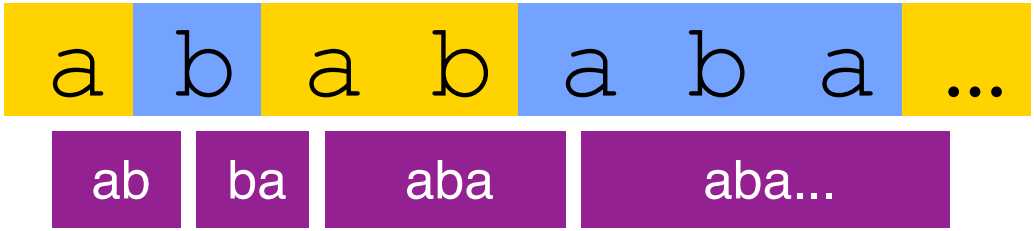
**Ende Wiederholung**;

| **Lesen (k)** | **Ausgabe** (q ist jeweils unterstrichen) | **Puffer füllen (p)** | **Codetabelle schreiben (p & <q>)** | **Merken (old)** |
|---|---|---|---|---|
| 98 | b | | | 98 |
| 97 | **a** | b | (<ba>, 256) | 97 |
| 110 | **n** | a | (<an>, 257) | 110 |
| 257 | **a**n | n | (<na>, 258) | 257 |
| 101 | **e** | an | (<ane>, 259) | 101 |
| 258 | **n**a | e | (<en>, 260) | 258 |
| 110 | **n** | na | (<nan>, 261) | 110 |
| 256 | **b**a | n | (<nb>, 262) | 256 |
| 117 | **u** | ba | (<bau>, 263) | 117 |
| EOF | | | | |

# LZW-Decodierung (4)

- Beispielzeichenreihe: “abababa...“, Beispielcode: “97-98-256-258“
- Ablauf:

| Lesen (k) | Ausgabe (q ist jeweils unterstrichen) | Puffer füllen (p) | Codetabelle schreiben (p & <q>) | Merken (old) |
|---|---|---|---|---|
| 97 | a | | | 97 |
| 98 | **b** | a | (<ab>, 256) | 98 |
| 256 | **a**b | b | (<ba>, 257) | 256 |
| 258 | **???** | | | |




Decodierung ist so noch nicht korrekt!

# LZW-Decodierung, vollständige Fassung

**SeqChar** *p* := <>;
**int** *k* = NächsteEingabezahl;
Schreibe Zeichenreihe mit Tabellenindex *k* auf Ausgabe;
**int** *old* = *k*;
**Wiederhole** solange Eingabe nicht leer:
    *k* = NächsteEingabezahl;
    **SeqChar** *akt* = Zeichenreihe mit Tabellenindex *k*;
    *p* = Zeichenreihe mit Tabellenindex *old* (letztes Teilwort);
    **Falls** Index *k* in Tabelle enthalten
        **dann** **Char** *q* = erstes Zeichen von *akt*;
            Schreibe Zeichenreihe *akt* auf Ausgabe;
        **sonst** **Char** *q* = erstes Zeichen von *p*;
            Schreibe Zeichenreihe *p* & <*q*> auf Ausgabe;
    **Ende Fallunterscheidung;**
    Trage *p* & <*q*> in Tabelle ein
    (und erzeuge neuen Index dafür);
    *old* = *k*;
**Ende Wiederholung**;