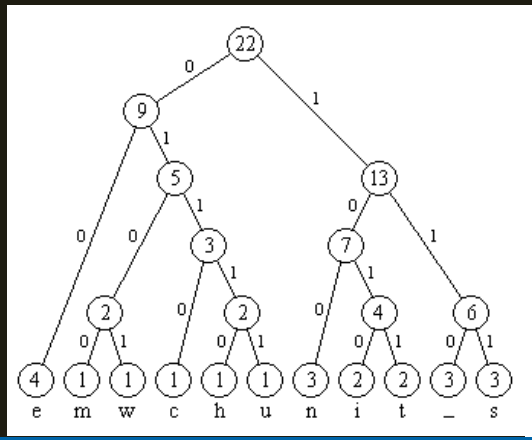
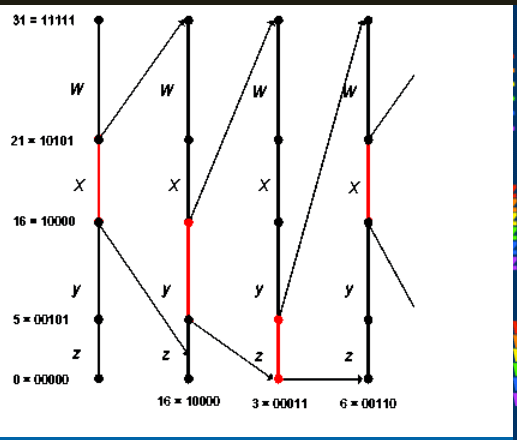
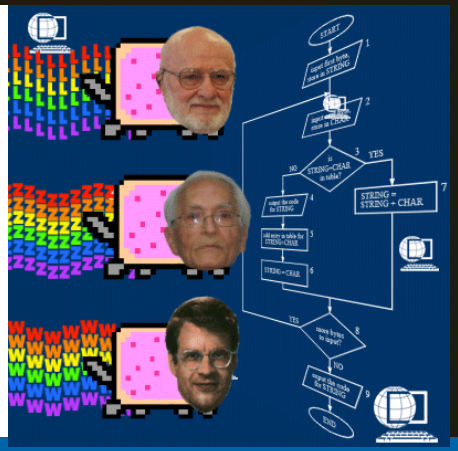
TECHNISCHE UNIVERSITÄT
BERGAKADEMIE FREIBERG
Die Ressourcenuniversität. Seit 1765.

Multimedia: Kompression







Jonas Treumer / Ben Lorenz

Themen

- Datenkompression
    - (Verlustbehaftet) später
    - Verlustfrei
        - Entropiekodierer
        - Wörterbuchkodierer

Datenkompression

- Ziele:

Datenkompression

- Ziele:
    - Datenreduktion → Speicherplatz, Übertragungsgeschwind.

- Mittel:

Datenkompression

- Ziele:
    - Datenreduktion → Speicherplatz, Übertragungsgeschwind.

- Mittel:
    - „Überflüssige“ Informationen entfernen
        - Redundanzreduktion → verlustfrei
        - Irrelevanzreduktion → verlustbehaftet

Welche Daten können komprimiert werden?

Welche Daten können komprimiert werden?

- Verlustbehaftet: alle Daten
    - erkläre Teil der Daten als redundant/irrelevant
    - Originaldaten nicht eindeutig rekonstruierbar

Welche Daten können komprimiert werden?

- Verlustbehaftet: alle Daten
    - erkläre Teil der Daten als redundant/irrelevant
    - Originaldaten nicht eindeutig rekonstruierbar

- Verlustfrei: Daten, die Redundanz enthalten
    - Redundanz entfernen
    - Originaldaten exakt rekonstruierbar

Grenzen der verlustfreien Komprimierung

Grenzen der verlustfreien Komprimierung

- Völlig zufällige Daten lassen sich überhaupt nicht (verlustfrei) komprimieren

Grenzen der verlustfreien Komprimierung

- Völlig zufällige Daten lassen sich überhaupt nicht (verlustfrei) komprimieren

- Taubenschlag/Schubfachprinzip
  - $verteile\ n\ Objekte\ auf\ m\ Mengen\ (M = \{M_1 \dots M_m\})$
    $mit\ \ (m, n > 0)\ und\ n > m \rightarrow \exists x\ mit\ |M_x| \geq 2$

Grenzen der verlustfreien Komprimierung

- Völlig zufällige Daten lassen sich überhaupt nicht (verlustfrei) komprimieren

- Taubenschlag/Schubfachprinzip
  - $verteile\ n\ Objekte\ auf\ m\ Mengen\ (M = \{M_1 \dots M_m\})$
    $mit\ \ (m, n > 0)\ und\ n > m \rightarrow \exists x\ mit\ |M_x| \geq 2$

- Kolmogorov-Komplexität K
  - Länge der „kürzesten Bildungsvorschrift“ für eine Zeichenkette s
  - $K(s) \geq length(s) \rightarrow s\ nicht\ komprimierbar$

Kleine Zahlen sind schön - Variable Length Integers

- Array aus Ganzzahlen mit großem Wertebereich? → größter Wert definiert Datentyp des Arrays

Kleine Zahlen sind schön - Variable Length Integers

- Array aus Ganzzahlen mit großem Wertebereich? → größter Wert definiert Datentyp des Arrays
- großer Wertebereich, aber häufig kleine Zahlen = Verschwendung

1111 1101 0011 1110	0000 0000 0001 1111	0000 0000 0000 1011

Kleine Zahlen sind schön - Variable Length Integers

- Array aus Ganzzahlen mit großem Wertebereich? → größter Wert definiert Datentyp des Arrays
- großer Wertebereich, aber häufig kleine Zahlen = Verschwendung

1111 1101 0011 1110	0000 0000 0001 1111	0000 0000 0000 1011

- Verbesserung durch VarInt-Codierung:

0	111 1111

1	001 1010	0	000 0101

Kleine Zahlen sind schön - Variable Length Integers

- Array aus Ganzzahlen mit großem Wertebereich? → größter Wert definiert Datentyp des Arrays
- großer Wertebereich, aber häufig kleine Zahlen = Verschwendung

1111 1101 0011 1110	0000 0000 0001 1111	0000 0000 0000 1011

- Verbesserung durch VarInt-Codierung:

0	111 1111

1	001 1010	0	000 0101

Negative Zahlen?

Kleine Zahlen sind schön - Variable Length Integers

- Bsp.-Präkodierung für VarInts:
  - Deltakodierung
    - 260, 261, 256, 265, 262 → 260, 1, -4, 5, 2
  - Move-To-Front Transformation (siehe Tafel)
    - Zeichen werden durch ihre Position in einem Alphabet kodiert
    - Alphabet „rotiert“, häufige Zeichen stehen weiter vorn

Entropiekodierer - Lauflängenkodierung (RLE)

Entropiekodierer - Lauflängenkodierung (RLE)

1000111101000011111
1341144

AAAABBBBBBBBCCCCC
{A,4}{B,8}{C,5}

Entropiekodierer - Lauflängenkodierung (RLE)

1000111101000011111
1341144

AAAABBBBBBBBCCCCC
{A,4}{B,8}{C,5}

- Burrows-Wheeler-Transformation (siehe Tafel)
    - Präkodierung, um zB Input für RLE zu verbessern

Entropiekodierer - Huffman-Kodierung

- David A. Huffman (1952)
- Quellsymbole → Codeworte variabler Länge

Entropiekodierer - Huffman-Kodierung

- David A. Huffman (1952)
- Quellsymbole → Codeworte variabler Länge
- Codeworte sind präfixfrei

Entropiekodierer - Huffman-Kodierung

- David A. Huffman (1952)
- Quellsymbole → Codeworte variabler Länge
- Codeworte sind präfixfrei
- Kodierer:
    - Input:
        - Nachricht
        - Auftrittswahrscheinlichkeit der Symbole
    - Output:
        - Codeworttabelle

Entropiekodierer - Huffman-Kodierung

- David A. Huffman (1952)
- Quellsymbole → Codeworte variabler Länge
- Codeworte sind präfixfrei
- Kodierer:
    - Input:
        - Nachricht
        - Auftrittswahrscheinlichkeit der Symbole
    - Output:
        - Codeworttabelle
- Dekodierer:
    - Input:
        - Komprimierter Datenstrom
        - Codeworttabelle
    - Output:
        - Dekodierte Nachricht

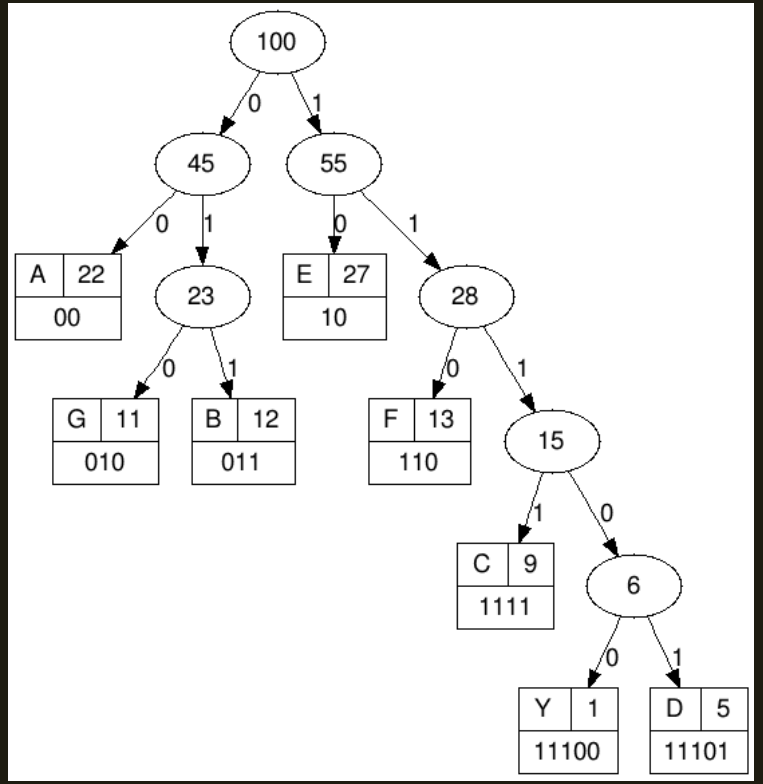
Huffman-Kodierung

- Gegeben sei folgendes Symbolalphabet und die Auftrittswahrscheinlichkeiten der Symbole:
- A:22%, B:12%, C:9%, D:5%, E:27%, F:13%, G:11%, Y:1%

- Stellen Sie mit Hilfe des Huffman-Algorithmus eine Codetabelle für dieses Symbolalphabet auf.
- Weisen Sie bei jeder Verzweigung des Baumes jeweils dem Knoten mit der kleineren kombinierten Häufigkeit die Ziffer 0 und dem Knoten mit der größeren kombinierten Häufigkeit die Ziffer 1 zu.

- Dekodieren Sie: 011000110011100000 1000

Huffman-Kodierung



- Dekodieren Sie: 0110001100111000001000

Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

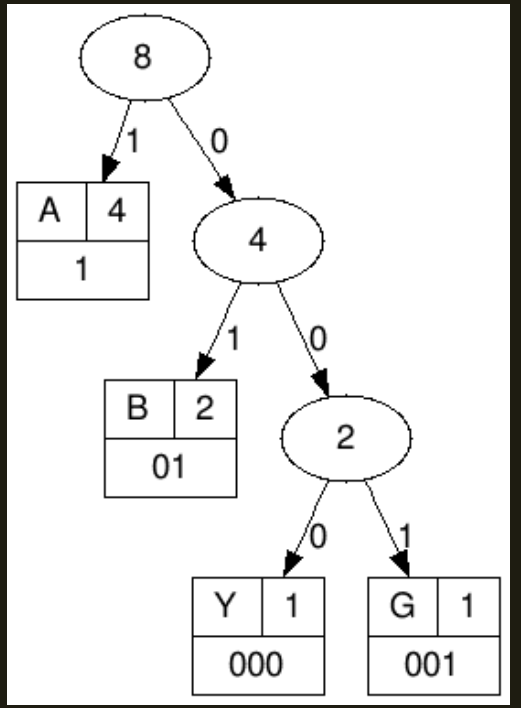
- Optimale Codewortlänge?

$$H \leq \bar{l} \leq H + 1$$

$$H = -\sum p_z \log_2 p_z$$

Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.



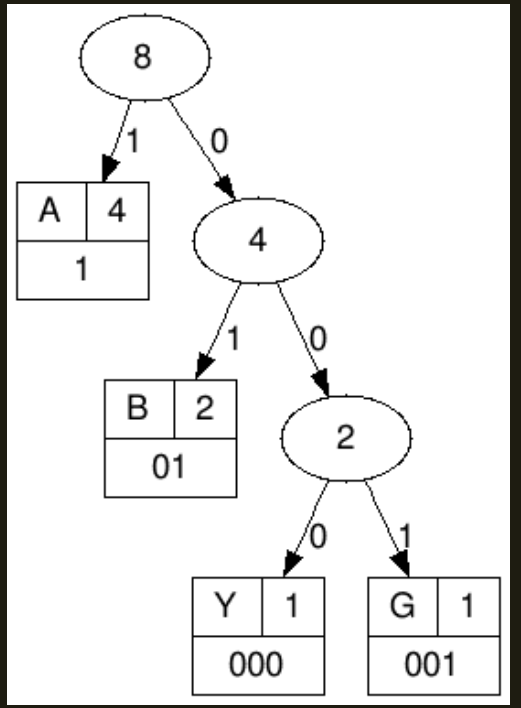
Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

- Optimale Codewortlänge?

$$H \leq \bar{l} \leq H + 1$$

$$H = -\sum p_z \log_2 p_z$$



Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

- Optimale Codewortlänge?

$$H \leq \bar{l} \leq H + 1$$

$$H = -\sum p_z \log_2 p_z$$

Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

- Optimale Codewortlänge?

$$\bar{l} = 2.25$$
$$H = 1.75$$
$$1.75 \leq 2.25 \leq 2.75 \rightarrow Codewortlänge\ optimal$$

Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

- Optimale Codewortlänge?
- Codewortlänge?

Huffman-Kodierung

- BABAYAGA
- Erzeugen Sie aus diesem Wort mit Hilfe der Huffman-Kodierung die binären Codeworte für die einzelnen Symbole.

- Optimale Codewortlänge?
- Codewortlänge?
    - 14 Bit

Entropiekodierer - Arithmetische Kodierung

Entropiekodierer - Arithmetische Kodierung

- Nachricht wird als einzelne Zahl kodiert

Entropiekodierer - Arithmetische Kodierung

- Nachricht wird als einzelne Zahl kodiert
- Kodierer:
    - Input:
        - Nachricht
        - Auftrittswahrscheinlichkeit der Zeichen
    - Output:
        - Zahl

Entropiekodierer - Arithmetische Kodierung

- Nachricht wird als einzelne Zahl kodiert
- Kodierer:
    - Input:
        - Nachricht
        - Auftrittswahrscheinlichkeit der Zeichen
    - Output:
        - Zahl

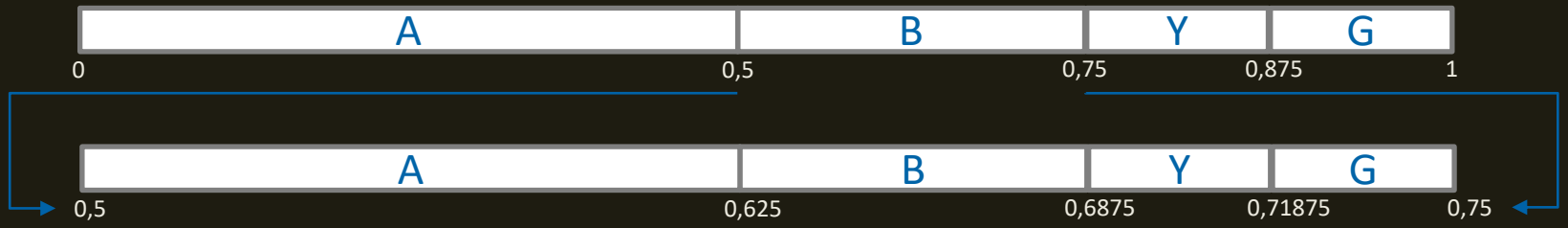
- Dekodierer:
    - Input:
        - Zahl
        - Auftrittswahrscheinlichkeit der Zeichen
        - Anzahl Zeichen in Nachricht (?)
    - Output:
        - Nachricht

Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%

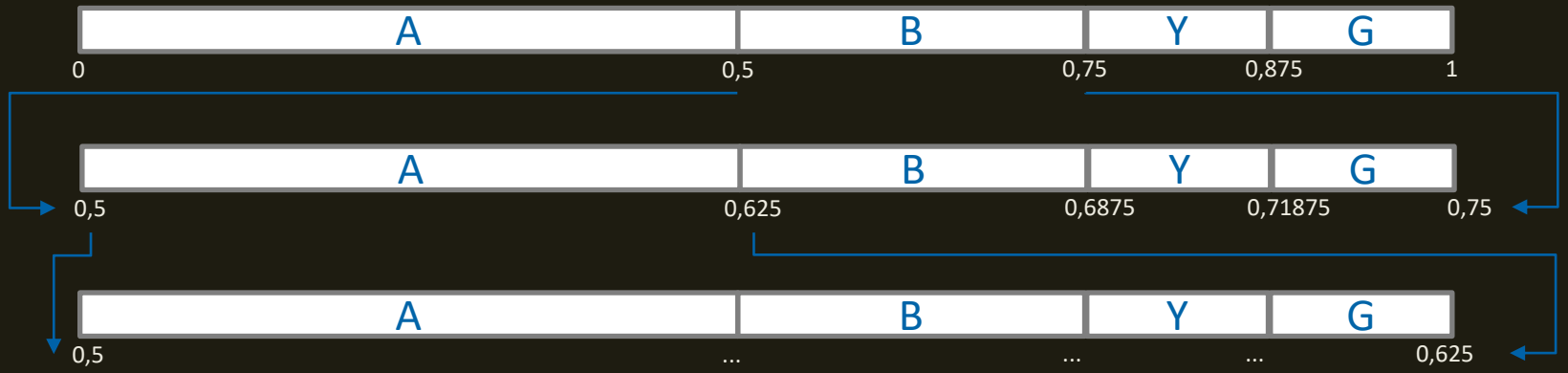
Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%



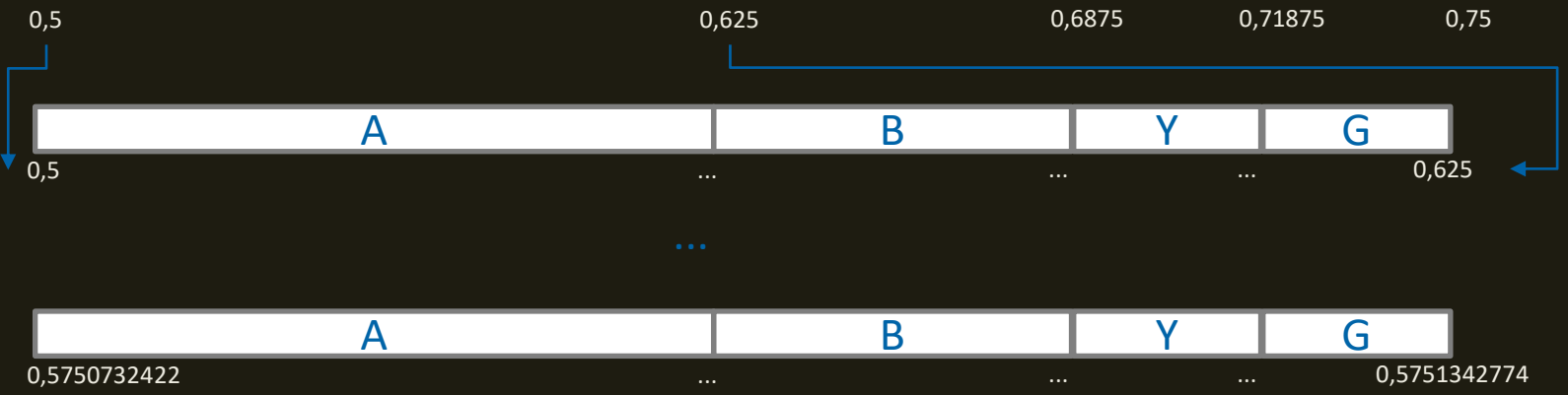
Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%



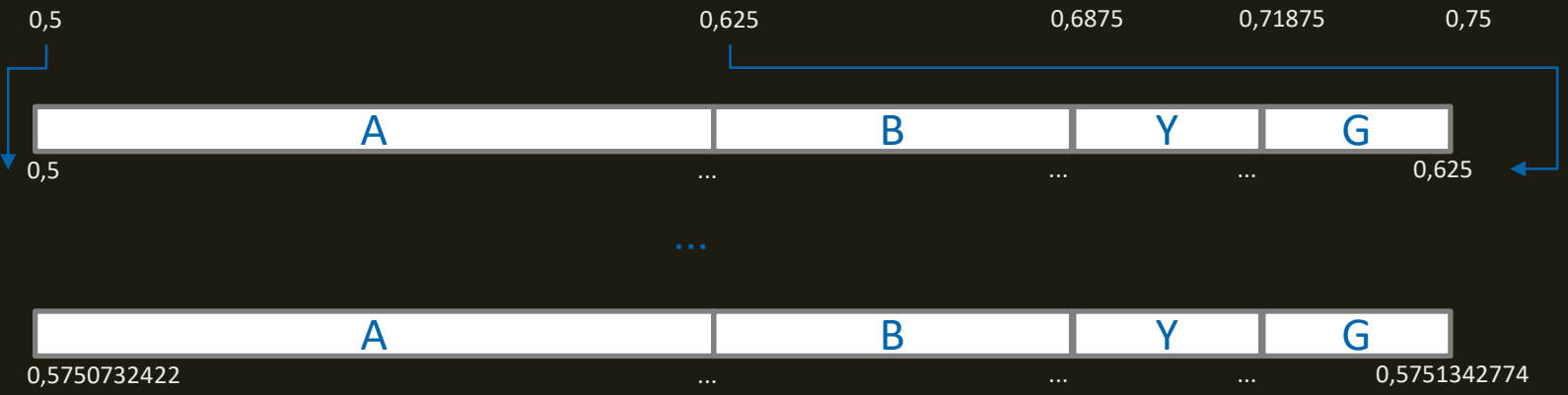
Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%



Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%



- Beliebige Zahl aus Intervall: 0,5751
- Wie viele Bits?

Arithmetische Kodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%

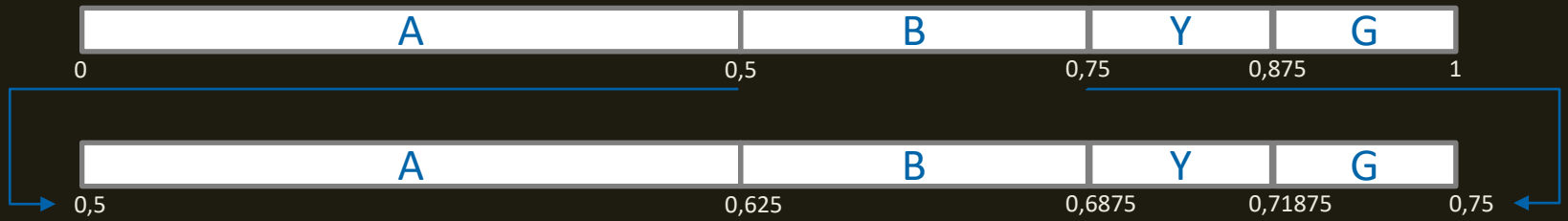
- Beliebige Zahl aus Intervall: 0,5751
- Wie viele Bits? 13 Bit (vs 14 Bit Huffman-Kodierung)

Arithmetische Dekodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%
- 0,5751 im Intervall [0,5;0,75] – B – Intervallgrenzen anpassen

Arithmetische Dekodierung

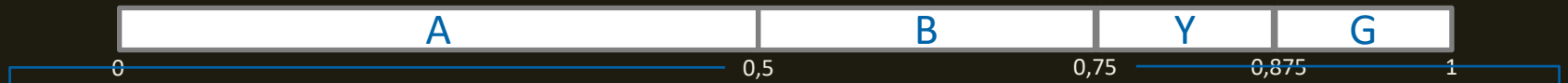
- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%
- 0,5751 im Intervall [0,5;0,75] – B – Intervallgrenzen anpassen



- 0,5751 im Intervall [0,5;0,625] – A – Intervallgrenzen anpassen ...

Arithmetische Dekodierung

- BABAYAGA A: 50%, B: 25%, Y: 12.5%, G: 12.5%



- 0,5751 im Intervall [0,5;0,75] – B – Intervallgrenzen anpassen ...

- 0,5751 im Intervall [0,5;0,625] – A – Intervallgrenzen anpassen ...

- 0,5751 im Intervall [0,5625;0,59375] – B – Intervallgrenzen anpassen...

Wörterbuchkompression - Lempel-Ziv-Welch-Kodierung

- Dynamische Wörterbuchkompression

Wörterbuchkompression - Lempel-Ziv-Welch-Kodierung

- Dynamische Wörterbuchkompression
- GIF Format, TIFF und PDF optional

Wörterbuchkompression - Lempel-Ziv-Welch-Kodierung

- Dynamische Wörterbuchkompression
- GIF Format, TIFF und PDF optional
- Kodierer und Dekodierer bauen Wörterbuch „on-the-fly“ auf
- Maximale Codewortlänge 12 Bit = 4096 Einträge im Wörterbuch
- Erste 256 Einträge fest (0-255)

Wörterbuchkompression - Lempel-Ziv-Welch-Kodierung

- Dynamische Wörterbuchkompression
- GIF Format, TIFF und PDF optional
- Kodierer und Dekodierer bauen Wörterbuch „on-the-fly“ auf
- Maximale Codewortlänge 12 Bit = 4096 Einträge im Wörterbuch
- Erste 256 Einträge fest (0-255)

Wörterbuchkompression - Lempel-Ziv-Welch-Kodierung

- ABRAKADABRA

Lempel-Ziv-Welch-Kodierung

- ABRAKADABRA -> AB -> (256) -> A ausgeben
- BRAKADABRA -> BR -> (257) -> B ausgeben
- ...

Lempel-Ziv-Welch-Kodierung

- ABRAKADABRA -> AB -> (256) -> A ausgeben
- BRAKADABRA -> BR -> (257) -> B ausgeben
- RAKADABRA -> RA -> (258) -> R ausgeben
- AKADABRA -> AK -> (259) -> A ausgeben
- KADABRA -> KA -> (260) -> K ausgeben
- ADABRA -> AD -> (261) -> A ausgeben
- DABRA -> DA -> (262) -> D ausgeben
- ABRA -> AB -> (256) ausgeben
- RA -> (258) ausgeben

- ABRAKAD(256)(258)

Lempel-Ziv-Welch-Dekodierung

- P – Präfix & C – Cache
- ABRAKAD(256)(258)

- A -> P = „A“
- B -> C = „B“; P&C = „AB“ -> (256); P = „B“
- R -> C = „R“; P&C = „BR“ -> (257); P = „R“
- A -> C = „A“; P&C = „RA“ -> (258); P = „A“
- K -> C = „K“; P&C = „AK“ -> (279); P = „K“
- A -> C = „A“; P&C = „KA“ -> (260); P = „A“
- D -> C = „D“; P&C = „AD“ -> (261); P = „D“
- (256) -> C = „A“; P&C = „DA“ -> (262); P = (256) = „AB“
- (258) -> C = „R“; P&C = „(256)R“ -> (263); P = (258) = „RA“

Lempel-Ziv-Welch-Dekodierung

- ABRAKADABRA
  - 8Bit / Zeichen
  - 11 Zeichen * 8 Bit = 88 Bit

- ABRAKAD(256)(258)
  - 9 Bit / Wörterbucheintrag
  - 7 Zeichen * 9 Bit + 2 * 9 Bit = 81 Bit

Progammieraufgabe: Honkpack

- 1 Byte Indikator, n Bytes Datenblock
- Indikator:
  - → Bit 7: Typ des Blocks
    - 1 → homogener Block
    - 0 → heterogener Block
  - → Bit 0-6: Dekomprimierte Länge der Blocks ($l <= 127$)

Indikatorbyte

```
t | 1111111
7   6543210
```

- Datenblock:
  - → Homogen: Es folgt genau ein Byte ($n = 1$), das l-mal wiederholts ausgegeben wird
  - → Heterogen: Es folgen l Bytes ($n = l$), die unverändert ausgegeben werden

Progammieraufgabe: Honkpack

Beispiel:

```
komprimiert (2 + 4 + 2 + 2 = 10 Bytes):
84 2A | 03 01 02 03 | FF 00 | 01 77
hom.    het.          hom.    het.
l=4   | l=3         | l=127 | l=1

dekomprimiert (4 + 3 + 127 + 1 = 135 Bytes):
2A 2A 2A 2A | 01 02 03 | 00 00 00 ... 00 | 77
```