# b) Kodierung & Komprimierung

Mittwoch, 8. Februar 2023 03:58

**Lauflängenkodierung**

-Einfachste Variante der Komprimierung, Bsp.:

**AAAADEBBHHHHHCAAABCCCC**

-Folgen von sich wiederholenden Zeichen lassen sich kompakter kodieren, indem man die Folge nur einmal angibt und dazu die Anzahl der jeweiligen Wiederholungen

**4ADE2B5HC3AB4C**

**Huffman-Codierung**

•Fano-Bedingung: Achte darauf, dass kein Code zugleich der Beginn eines anderen Codes ist

•Präfixfreie Kodierung: -kein Code ist zugleich Anfang eines anderen Codes
-lässt sich am leichtesten durch Binärbaumkodierung erzeugen

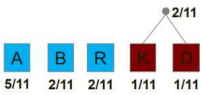
•Ablauf:

1. Ermittle die relative Häufigkeit der zu kodierenden Zeichen

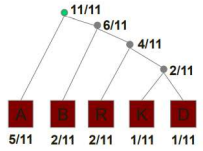
**ABRAKADABRA** 

| Buchstabe | Anzahl | Häufigkeit |
|---|---|---|
| A | 5 | 5/11 |
| B | 2 | 2/11 |
| R | 2 | 2/11 |
| K | 1 | 1/11 |
| D | 1 | 1/11 |

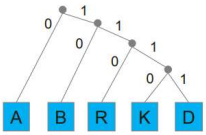
2. Fasse die beiden Zeichen $c_i$ und $c_j$ mit der geringsten Häufigkeit $f(c_i)$ und $f(c_j)$ zusammen zu einem neuen Knoten mit der Häufigkeit $f(c_i)+f(c_j)$




3. Fahre fort, bis alle Blattknoten in einem gemeinsamen Baum verbunden sind




4. Interpretiere Baum als Binärbaumkodierung




5.Ergebnis/Code:

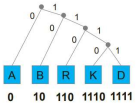
**ABRAKADABRA** 

**0 10 110 0 1110 0 1111 0 10 110 0**
A B R A K A D A B R A

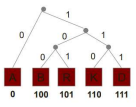
* Gesamtlänge: **23 Bit**

* **Achtung**: Es kann mehrere, unterschiedliche optimale Codes geben

->Es gibt verschiedene Varianten:




**0 10 110 0 1110 0 1111 0 10 110 0**
A B R A K A D A B R A




**0 100 101 0 110 0 111 0 100 101 0**
A B R A K A D A B R A

* Gesamtlänge: **23 Bit**

* **Achtung**: Es kann mehrere, unterschiedliche optimale Codes geben

**LZW-Komprimierung**

-Wörterbuchbasierte Komprimierung
-Lempel, Ziv, Welch
-Adaptives Verfahren (zip-Komprimierung)
-wird für Grafikformate GIF/TIFF genutzt
-Prinzipieller Ablauf: (1) erzeuge aus zu komprimierenden Zeichenketten ein Wörterbuch
(2) Daten werden mit Wörterbuch kodiert (komprimiert)
(3) Wörterbuch muss (implizit) mit übertragen (gespeichert) werden

•Ablauf: LZW-Algorithmus :

1. Lese Zeichen aus zu komprimierenden Daten und akkumuliere diese zu Zeichenkette S, solange sich S als Wörterbucheintrag findet.
2. Sobald Zeichen x gelesen wird, für da sich Sx nicht im Wörterbuch findet, fahre folgendermaßen fort:
   ->nehme Sx in das Wörterbuch auf
   ->kodiere S gemäß Wörterbuch
   ->Starte eine neue Zeichenkette S‘ mit dem Zeichen x
3. Wiederhole (1,2) bis das Ende der zu komprimierenden Daten erreicht ist

• Beispiel: -Algorithmus startet mit Wörterbuch, in dem die ersten 256 Einträge aus den 8-Bit ASCII-Zeichen besteht
-Die Wörterbucheinträge bestehen typischerweise aus 12 Bit langen Codewörtern (= 4096 Einträge)
-Zu komprimieren ist die folgende Zeichenfolge: ABRAKADABRAABRAKADABRA
-Als 8-Bit ASCII Kodierung beträgt die Länge der Zeichenkette 22 x 8 Bit = 176 Bit

| Restzeichenkette | Gefundener Eintrag | Ausgabe | Neuer Eintrag |
|---|---|---|---|
| ABRAKADABRAABRAKADABRA | A | A | AB <256> |
| BRAKADABRAABRAKADABRA | B | B | BR <257> |
| RAKADABRAABRAKADABRA | R | R | RA <258> |
| AKADABRAABRAKADABRA | A | A | AK <259> |
| KADABRAABRAKADABRA | K | K | KA <260> |
| ADABRAABRAKADABRA | A | A | AD <261> |
| DABRAABRAKADABRA | D | D | DA <262> |
| ABRAABRAKADABRA | AB | <256> | ABR <263> |
| RAABRAKADABRA | RA | <258> | RAA <264> |
| ABRAKADABRA | ABR | <263> | ABRA <265> |
| AKADABRA | AK | <259> | AKA <266> |
| ADABRA | AD | <261> | ADA <267> |
| ABRA | ABRA | <265> | |

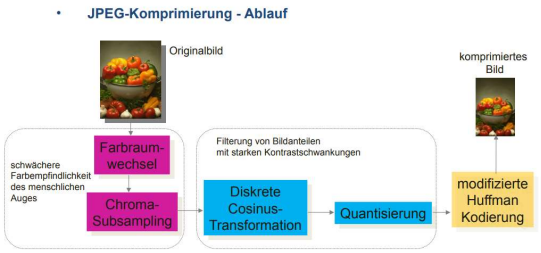
• Beispiel: -Ausgabe: ABRAKAD<256><258><263><259><261><265>
-> Länge der komprimierten Zeichenkette: 13 x 12 Bit = 156 Bit
-Bei der Dekodierung wird das Wörterbuch schrittweise rekonstruiert
-Dies ist möglich, da die Ausgabe des LZW-Algorithmus stets nur Codewörter enthält, die zu diesem Zeitpunkt bereits im Wörterbuch standen

| Erstes Zeichen | Ausgabe | Neuer Eintrag |
|---|---|---|
| A | A | |
| B | B | AB <256> |
| R | R | BR <257> |
| A | A | RA <258> |
| K | K | AK <259> |
| A | A | KA <260> |
| D | D | AD <261> |
| <256> | AB | DA <262> |
| <258> | RA | ABR <263> |
| <263> | ABR | RAA <264> |
| <259> | AK | ABRA <265> |
| <261> | AD | AKA <266> |
| <265> | ABRA | ADA <267> |

JPEG-Komprimierung

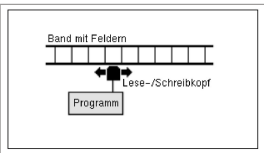
-verlustbehaftete Komprimierung
-sehr gut geeignet für natürliche Bildquellen
-Komprimierung bis 1:20 bei kaum nennenswerten Verlust der Darstellungsqualität
-Ausnutzung der Physiologie der menschlichen Wahrnehmung
->Das menschliche Auge reagiert auf Änderungen der Helligkeit empfindlicher als auf Farbänderungen
-Natürliche Bildquellen: ->häufig Farb- / Helligkeitsverläufe
->häufig keine starken Kontrastschwankungen

•Ablauf:




Turingmashine

Aufbau




Bsp

**BEISPIEL**

**Aufgabe: AAA+AA = AAAAA**

**Annahme: Lese-Schreibkopf steht zu Beginn beim ersten A**

**(z1,A) -> (z1,A,R) [wir befinden uns im ersten A-Block und bewegen uns nach rechts]**

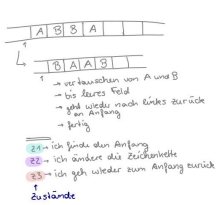
**(z1,+) → (z1,A,R) [+ gefunden, wird durch A ersetzt, wir gehen weiter nach rechts]**

**(z1,_) → (z2,_,L)[wir haben das Ende der Folge erreicht und müssen jetzt nach links gehen, damit wir dort das letzte A löschen können]**

**(z2,A)→ (z2,_,H)[letztes A wird gelöscht (damit sind wieder so viele As wie am Anfang da, aber jetzt ohne + dazwischen), damit ist die Aufgabe erfüllt und wir bleiben stehen]**

**WAS MACHT DIESES PROGRAMM?**

**(z1 ,A) ---> (z2 ,B,R )**
**(z1 ,B) ---> (z2 ,A,R)**
**(z1 ,_) ---> (z1 ,_,R)**
**(z2 ,A) ---> (z2 ,B,R)**
**(z2 ,B) ---> (z2 ,A,R)**
**(z2 ,_) ---> (z3 ,_,L)**
**(z3 ,A) ---> (z3 ,A,L)**
**(z3 ,B) ---> (z3 ,B,L) [**
**(z3 ,_) ---> (z3 ,_,H)**



## WAS MACHT DIESES PROGRAMM?

**(z1 ,A) ---> (z2 ,B,R) [erster Buchstabe der gesehen wird ist ein A, das wird in ein B umgewandelt und der Zustand gewechselt (wir merken uns also, dass wir einen Buchstaben gesehen haben)]**

**(z1 ,B) ---> (z2 ,A,R) [wie oben nur mit gelesenem B]**

**(z1 ,_) ---> (z1 ,_,R) [im Anfangszustand befindet sich der Lese-Schreibkopf noch links von der Zeichenfolge, wir gehen auf der Suche nach dem Beginn der Folge nach rechts und bleiben im Anfangszustand]**

**(z2 ,A) ---> (z2 ,B,R) [zweites und folgendes A, wird durch B ersetzt und wir arbeiten uns weiter durch die Folge, gehen also nach rechts, Zustand bleibt gleich]**

**(z2 ,B) ---> (z2 ,A,R) [wie oben nur mit gelesenem B]**

**(z2 ,_) ---> (z3 ,_,L) [Ende der Folge ist erreicht; wir merken uns das (Wechsel nach Zustand z3) und machen uns auf den Rückweg an den Anfang der Folge]**

**(z3 ,A) ---> (z3 ,A,L) [wir laufen weiter zurück durch die Folge, sind noch nicht am Anfang angekommen]**

**(z3 ,B) ---> (z3 ,B,L) [wie oben]**

**(z3 ,_) ---> (z3 ,_,H) [Anfang wurde erreicht. Wir bleiben (im Feld vor dem Anfang )stehen. ]**

Bsp aus Vorlesung

Linke Hälfte des Hörsaals: Entwerfen Sie eine Turingmaschine, die die Vorkommen von A in einer Eingabe „zählt"

Eingabe: ABAABABBB
Ausgabe: AAAA

Rechte Hälfte des Hörsaals: Entwerfen Sie eine Turingmaschine, die einer Zeichenkette aus Einsen und Nullen ein „Paritätsbit" hinzufügt: 1 falls Zahl der Einsen ungerade 0 falls Zahl der Einsen gerade

```
Paritätsbit
(z1 ,0) ---> (z1 ,0,R) [keine oder gerade Zahl von Einsen gefunden]
(z1 ,1) ---> (z2 ,1,R) [vorher gerade Zahl von Einsen gefunden, jetzt noch eine, also ungerade]
(z1 ,_) ---> (z1 ,0,H) [bisher gerade Zahl von Einsen gefunden, jetzt am Ende angekommen, also insgesamt gerade 0 schreiben]
(z2 ,0) ---> (z2 ,0,R) [ungerade Zahl von Einsen gefunden]
(z2 ,1) ---> (z1 ,1,R) [vorher ungerade Zahl von Einsen gefunden, jetzt noch eine, also gerade ]
(z2 ,_) ---> (z2 ,1,H) [bisher ungerade Zahl von Einsen gefunden, jetzt am Ende angekommen, also insgesamt ungerade, 1 schreiben]


Zählen


(z1 ,%) ---> (z1 ,_,R) []
(z1 ,A) ---> (z2 ,%,R) [A gefunden, durch % ersetzen, nach rechts gehen]
(z1 ,B) ---> (z1 ,%,R) [noch kein A gefunden, B gefunden, durch % ersetzen, nach rechts gehen]
(z1 ,_) ---> (z5 ,_,L) [kein (weiteres) A gefunden, anfangen aufzuräumen]
(z2 ,A) ---> (z2 ,A,R) [mit gefundenem A auf dem Weg ans Ende]
(z2 ,B) ---> (z2 ,B,R) [mit gefundenem A auf dem Weg ans Ende]
(z2 ,_) ---> (z3 ,_,R) [mit gefundenem A am Ende der Zeichenkette angekommen, Platz zum Schreiben suchen]
(z3 ,A) ---> (z3 ,A,R) [hier ist schon voll, weiter nach rechts]
(z3 ,_) ---> (z4 ,A,L) [freien Platz gefunden, A ablegen]
(z4 ,%) ---> (z1 ,%,R) [Anfang der noch nicht bearbeiteten Zeichenkette gefunden, rechts weitersuchen]
(z4 ,A) ---> (z4 ,A,L) [unterwegs an den Anfang der Zeichenkette]
(z4 ,B) ---> (z4 ,B,L) [unterwegs an den Anfang der Zeichenkette]
(z4 ,_) ---> (z4 ,_,L) [unterwegs an den Anfang der Zeichenkette]
(z5 ,%) ---> (z5 ,_,L) [alle % löschen]
(z5 ,_) ---> (z6 ,_,R) [alle % gelöschst, zurück an den Anfang des Ergebnisses]
(z6 ,A) ---> (z6 ,A,H) [Ergebnisanfange gefunden, fertig]
(z6 ,_) ---> (z6 ,_,R) [zurück an den Anfang des Ergebnisses]
```