Algorithmen und Datenstrukturen

D3. Kompression[1]

Marcel Lüthi

Universität Basel

23. Mai 2019

[1] Folien basierend auf Vorlesungsfolien von Sedgewick & Wayne
`https://algs4.cs.princeton.edu/lectures/55DataCompression-2x2.pdf`

Algorithmen und Datenstrukturen

23. Mai 2019 — D3. Kompression[a]

[a] Folien basierend auf Vorlesungsfolien von Sedgewick & Wayne
https://algs4.cs.princeton.edu/lectures/55DataCompression-2x2.pdf

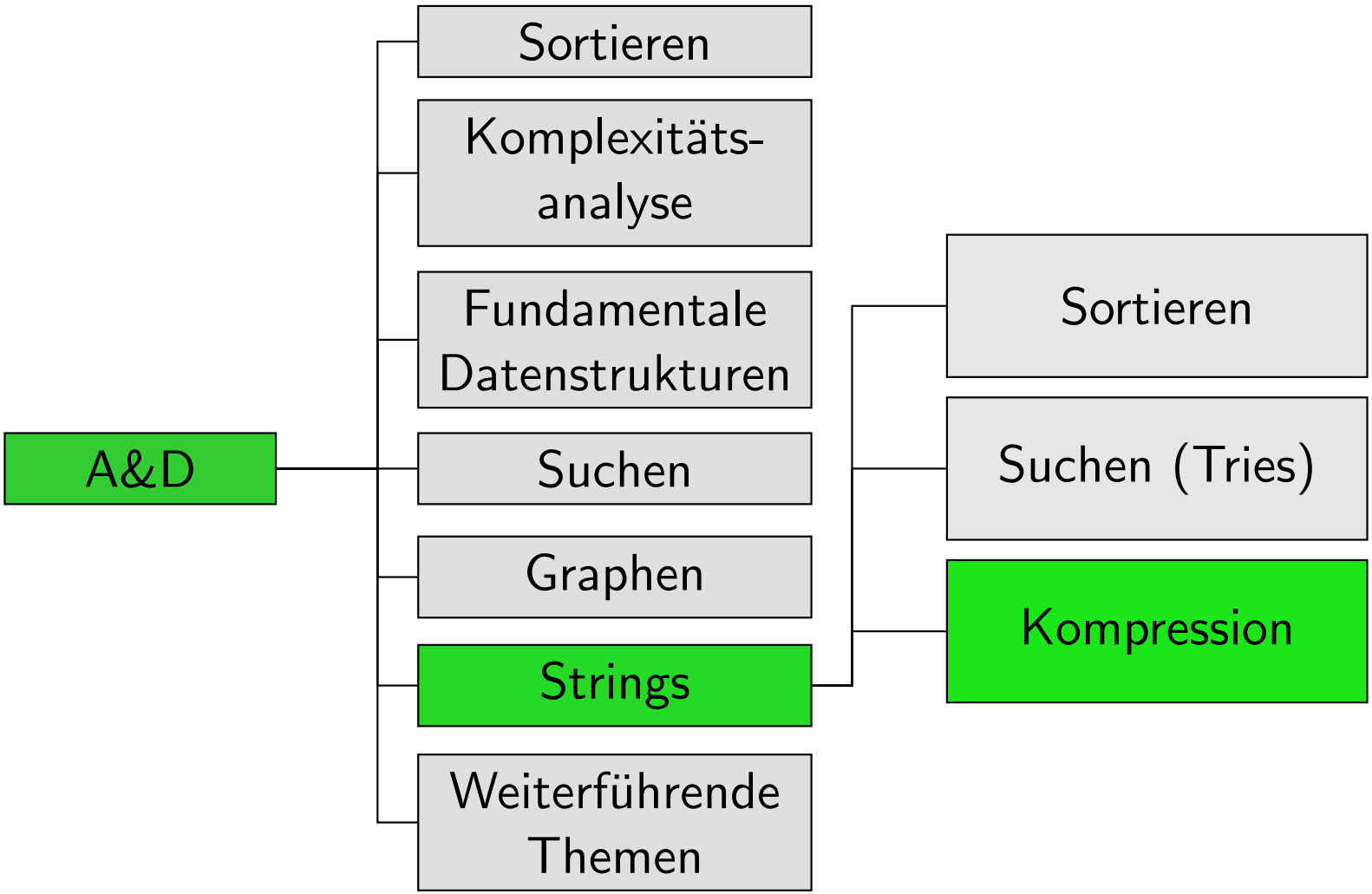
D3.1 Einführung

D3.2 Lauflängencodierung

D3.3 Huffmann coding

D3.1 Einführung

Übersicht


A&D
Sortieren
Komplexitäts-
analyse
Fundamentale
Datenstrukturen
Suchen
Graphen
Strings
Weiterführende
Themen
Sortieren
Suchen (Tries)
Kompression

Datenkompression

Kompression wird gebraucht um Grösse einer Datei zu reduzieren

- Weniger Speicherplatz
- Schnellere Übertragung (mehr Information bei kleinerer Bandbreite)

Fun Facts:

- Jede Minuten werden 60 Stunden Video auf Youtube hochgeladen.
- Im Cern werden pro Sekunde ca 1 Petabyte Daten generiert

Speicherung nur dank Kompression möglich.

Quiz

Kompression ist möglich, da viele Daten Redundanz aufweisen.

Geben Sie Beispiele von Redundanz in:

- geschriebenem Text
- Musik
- Bilder
- Videos.

Anwendungen

Komprimierung von Dateien

- Dateien (gzip, bzip, compress)
- Archivierungsprogramme (PKZip. Winzip)
- Dateisysteme (NTFS, ZFS)

Multimedia

- Bilder: Jpeg, png, ...
- Musik: MP3, ogg
- Videos: Mpeg, divX, ...

Genetik:

- Kompression von Sequenzdaten

Idee: Verlustfreie Kompression

Message: Binärdaten B die komprimiert werden sollen
Komprimieren: Komprimierte Repräsentation $C(B)$
Dekomprimieren: Rekonstruktion von Orginaldaten B

Quelle: Sedgewick & Wayne, Algorithmen, Abbildung 5.56

- Kompressionsrate: Bits in $C(B)$ / Bits in B

Beispiel: Code von fester Länge für Gensequenzen

Genom: Alphabet - ACGT

Ziel: Gensequenz der Länge N codieren (Beispiel ATAGCGTTAG)

Ascii Code

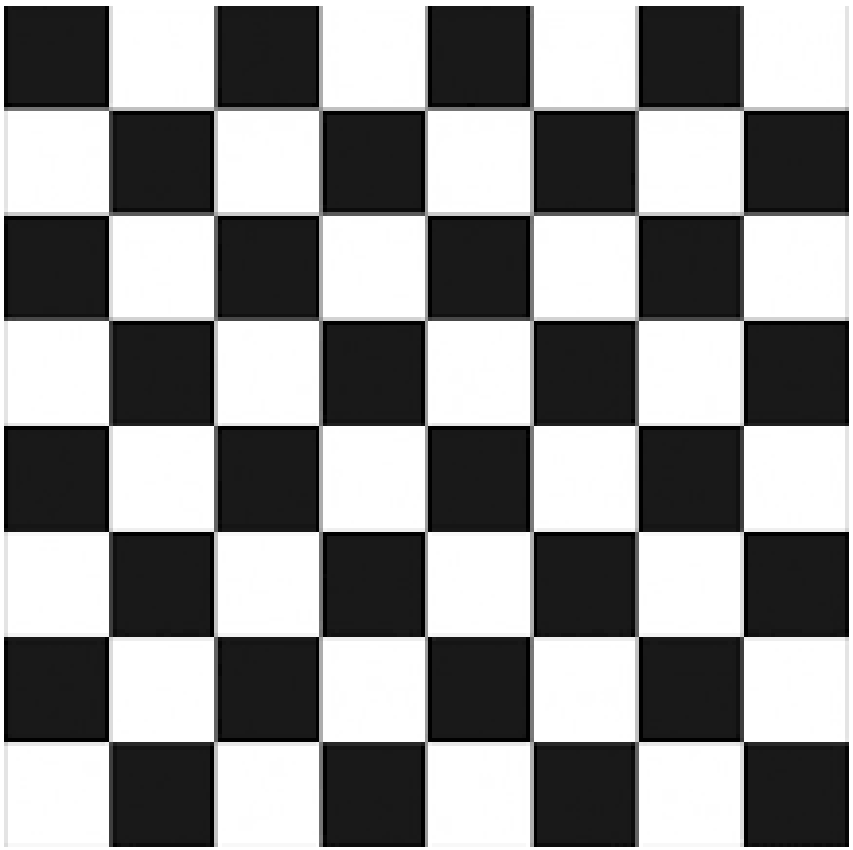
Zeichen	Binärrepräsentation
A	01000001
C	01000011
T	01010100
G	01000111

2-Bit Code

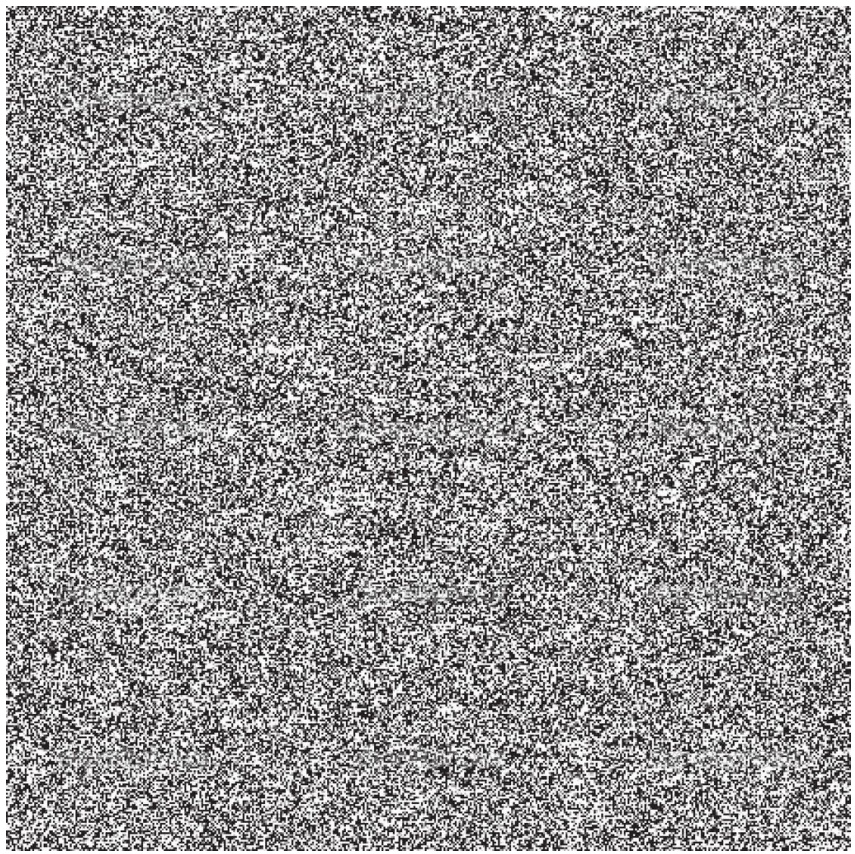
Zeichen	Binärrepräsentation
A	00
C	01
T	10
G	11

Limiten der Kompression

Nicht alle Daten können komprimiert werden.



Kompression einfach.



Kompression nicht möglich.

Universelle Kompression?

Theorem

Es gibt keinen Algorithmus der jeden beliebigen Bitstring komprimieren kann.

Argument 1:

- Idee: Kompressionsalgorithmus wird immer wieder rekursiv auf Ausgabe ausgeführt.
  - Widerspruch, da man so Grösse 0 erreicht

Universelle Kompression?

Theorem

Es gibt keinen Algorithmus der jeden beliebigen Bitstring komprimieren kann.

Argument 2:

- Annahme: Alle Bitstrings der Länge 1000 können komprimiert werden
- Also 2^{1000} verschiedene Strings können mit max 999 Bits codiert werden.
- Widerspruch, da wir nur max 2^{999} verschiedene Strings mit 999 Bits codieren können.

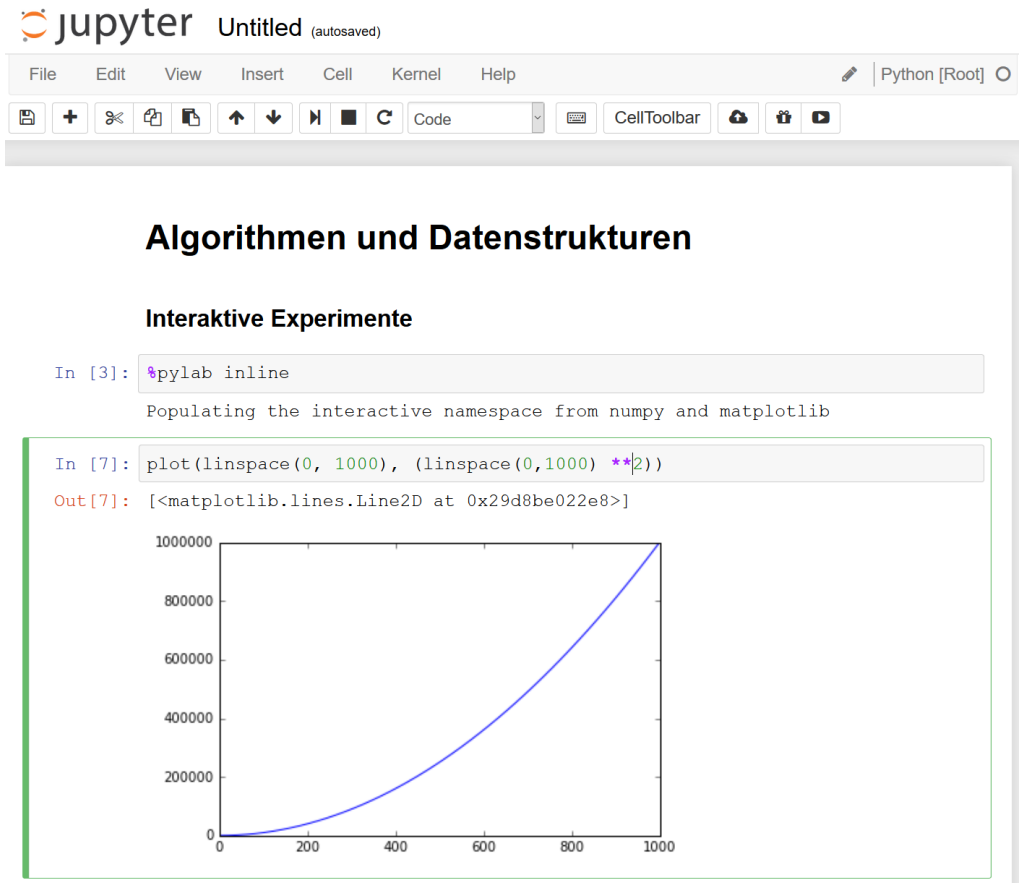
Schreiben / Lesen von binären Daten

```
class BinaryStream:

    def writeBit(b : boolean)
    def writeNBitNumber(number : int, n: int)
    def writeChar(c : char)
    def writeString(s : string)
    ...

    def readBit() -> boolean
    def readNBitNumber(n: int) -> number
    def readChar() -> char
    def readString(nChars) -> string
    ...
```

Beispiel: Datum



Jupyter Notebooks: Compression.ipynb

D3.2 Lauflängencodierung

Lauflängencodierung

Einfachste Art der Redundanz in Bitstrings: Viele aufeinanderfolgende 0 oder 1

$$000000000000000111111100000011111111$$

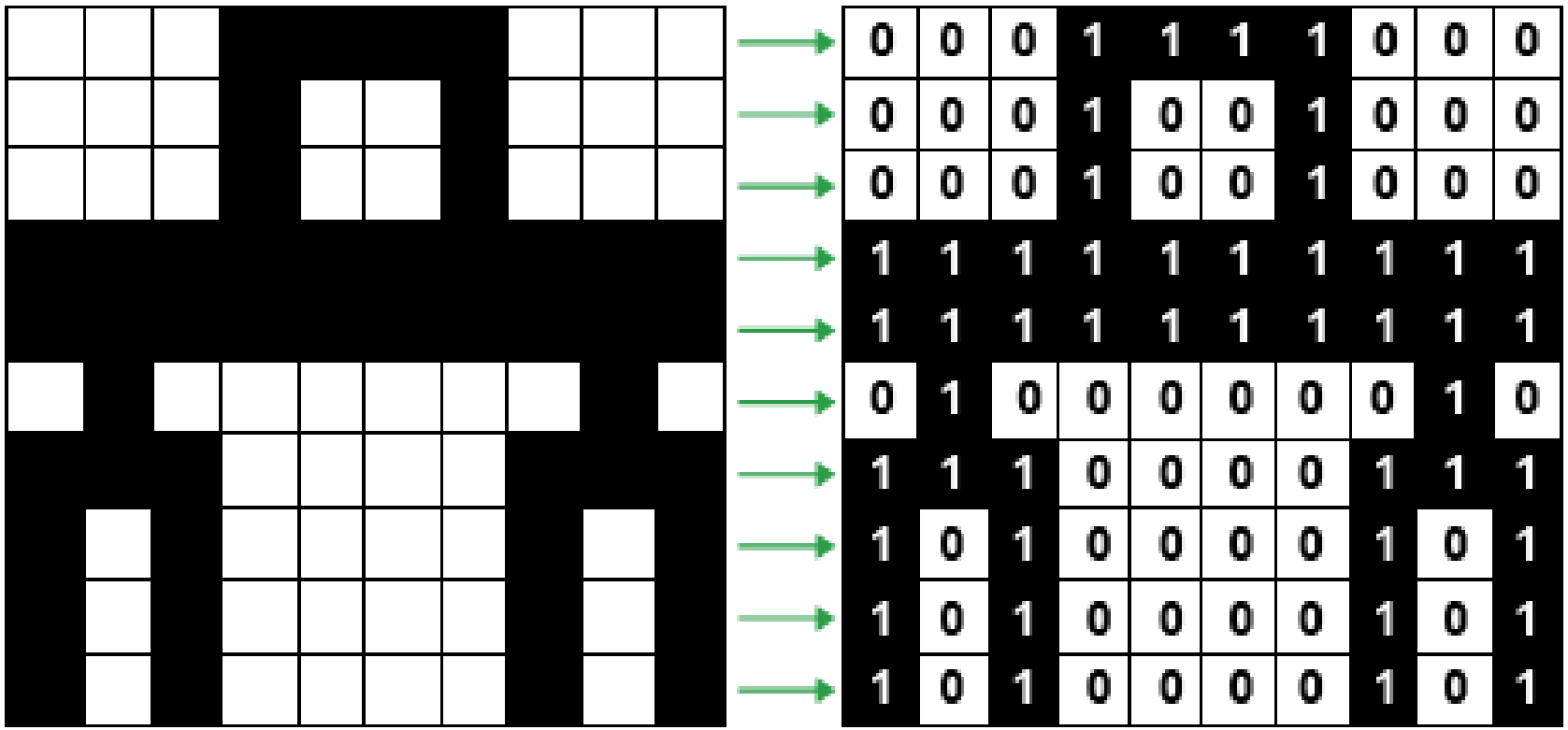
Repräsentation: 4 (oder 8) bit Zähler die alternierend für Anzahl 0 oder 1 stehen (beginnend mit 0)

$$\underbrace{1111}_{15\times 0}\,\underbrace{0111}_{7\times 1}\,\underbrace{0111}_{7\times 0}\,\underbrace{1011}_{11\times 1}$$

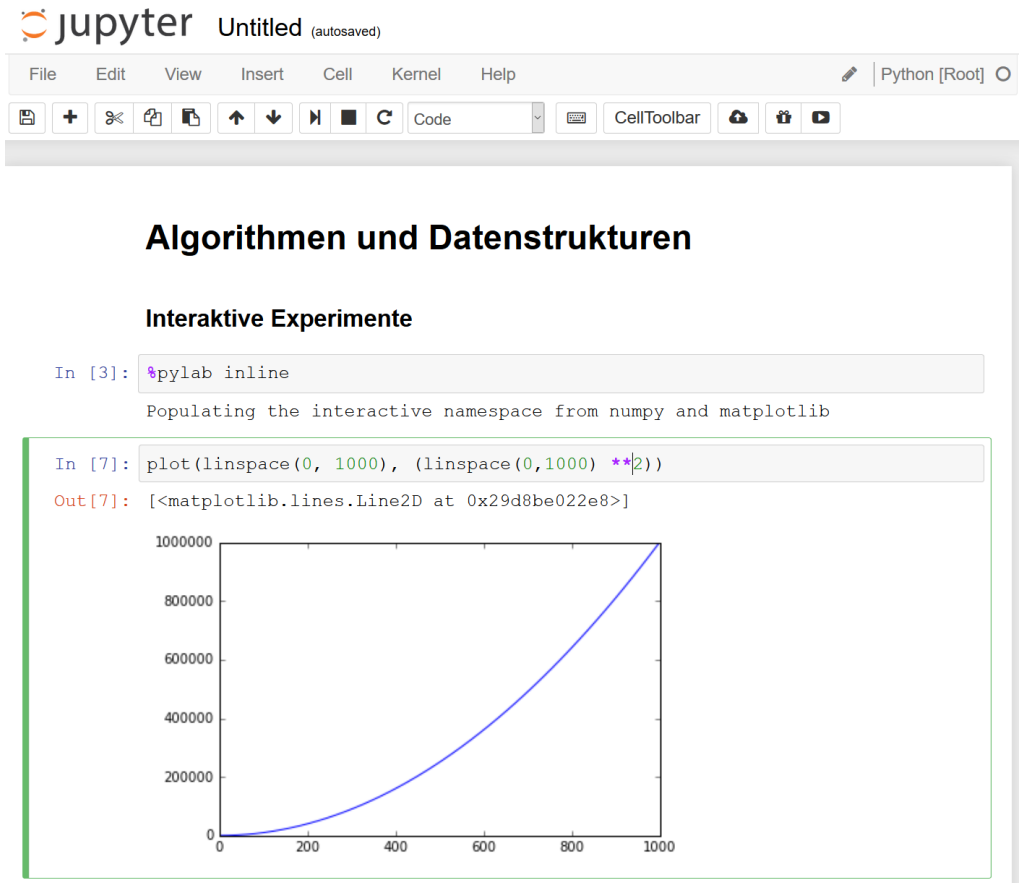
- Was machen wir wenn Sequenz von 0 oder 1 länger als Maximum des Zählers ist?
- Passende Sequenz der Länge 0 einfügen.

Lauflängencodierung: Anwendungen

Bitmaps speichern:



Beispiele und Implementation



Jupyter Notebooks: Compression.ipynb

D3.3 Huffmann coding

Huffman Komprimierung: Motivation

Ascii Codierung weist jedem Zeichen einen 8-bit Wert zu

A	B	R	A	C	A	D	A
01000001	01000010	01010010	01000001	01000011	01000001	01000100	01000001

- D benötigt gleich viel Speicher wie A.
- D kommt 1 mal vor, A 4 mal

Codes variabler Länge

Huffman Codierung weist jedem Zeichen einen Code variabler Länge zu.

Huffman Komprimierung: Motivation

A	B	R	A	C	A	D	A	B	R	A
0	1	00	0	01	0	10	0	1	00	0

Komprimierter String:

010000101001000

- Braucht viel weniger Platz
- Aber beginnt String mit AB oder C?
- Wie legen wir den Code fest?
- Wie weiss Empfänger was Code war?

Präfixfreie Codes

Zeichen bleiben unterscheidbar, wenn kein Zeichen ein Präfix eines andern ist.

Code	Zeichen
S	0
I	11
M	101
P	100

Welcher String ist hier codiert?

10111001100111 0011

Eindeutige Decodierbarkeit

Verfahren zum Entscheiden der Decodierbarkeit (Sardinas-Patterson)

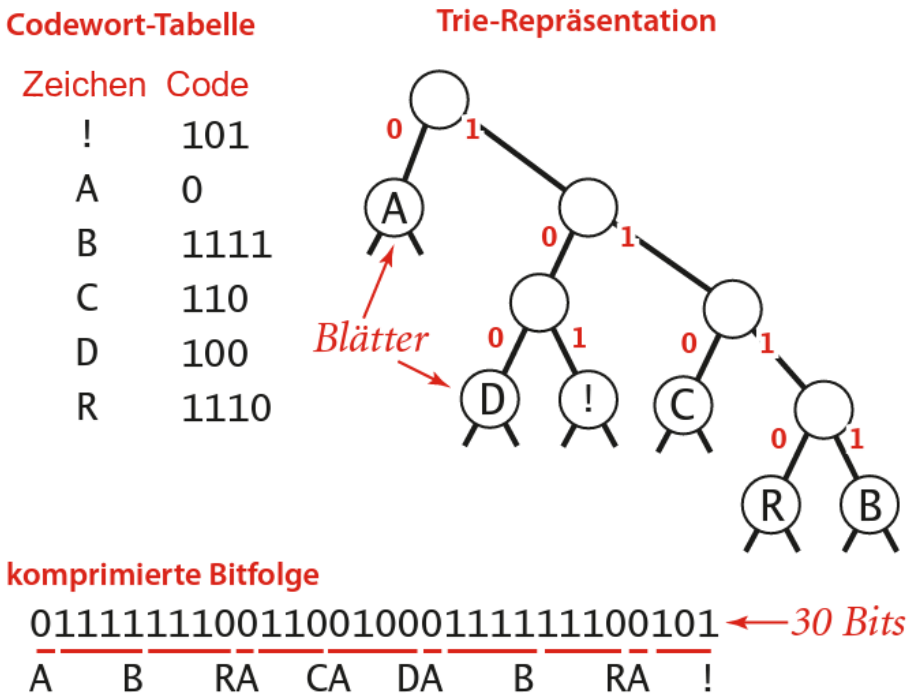
- Sei C die Menge aller Codewörter und S eine initial leere Liste.
- Bestimme für alle Paare $c_i, c_j \in C$ von Codewörter in C, ob c_i Präfix von c_j ist.
  - Falls ja (also $c_j = c_i w$), füge Suffix w zu S hinzu.
- Iteriere folgende Schritte, bis keine weiteren Einträge mehr zu S hinzukommen:
  - Für alle $s \in S$ und alle $c \in C$
  - Falls $c = sw$ (s is Präfix von c), füge Suffix w zu S hinzu.
  - Falls $s = cw$ (w is Präfix von s), füge Suffix w zu S hinzu.
- Falls $S \cap C = \emptyset$ ist Code eindeutig decodierbar.

Quiz

Welcher dieser Codes ist decodierbar?

- $\{01, 11100, 01100, 0101\}$
- $\{01, 1001, 11100, 1100, 101\}$
- $\{01, 1001, 11111, 01111, 0001\}$

Präfixfreie codes - Repräsentation als binärer Trie



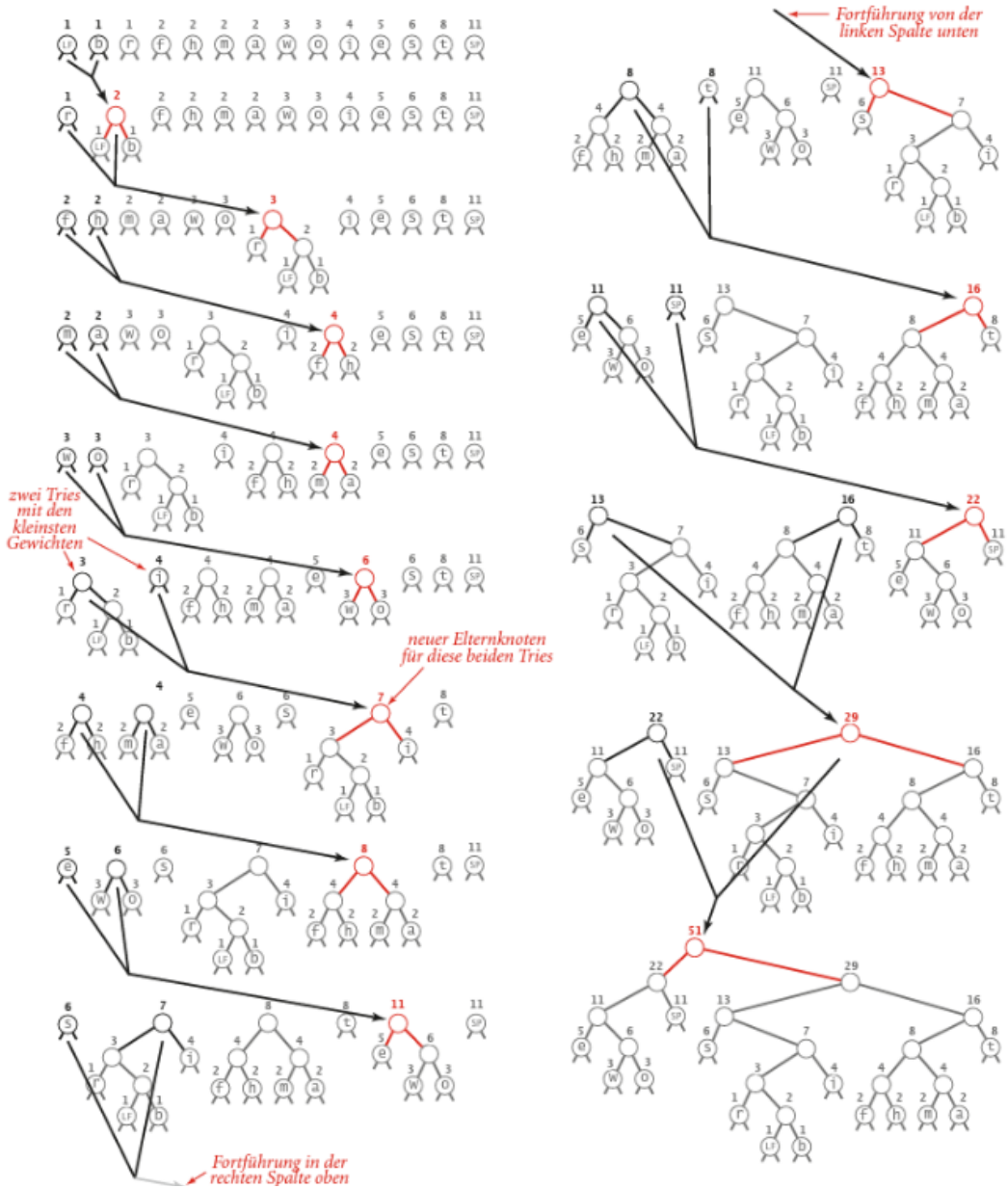
Quelle: Algorithmen, Sedgewick & Wayne, Abbildung 5.64

- Zeichen nur in Blätter gespeichert
- Kanten im Baum entsprechen 0 (links) und 1 (rechts)
- Code ist Pfad von Wurzel zu Blatt

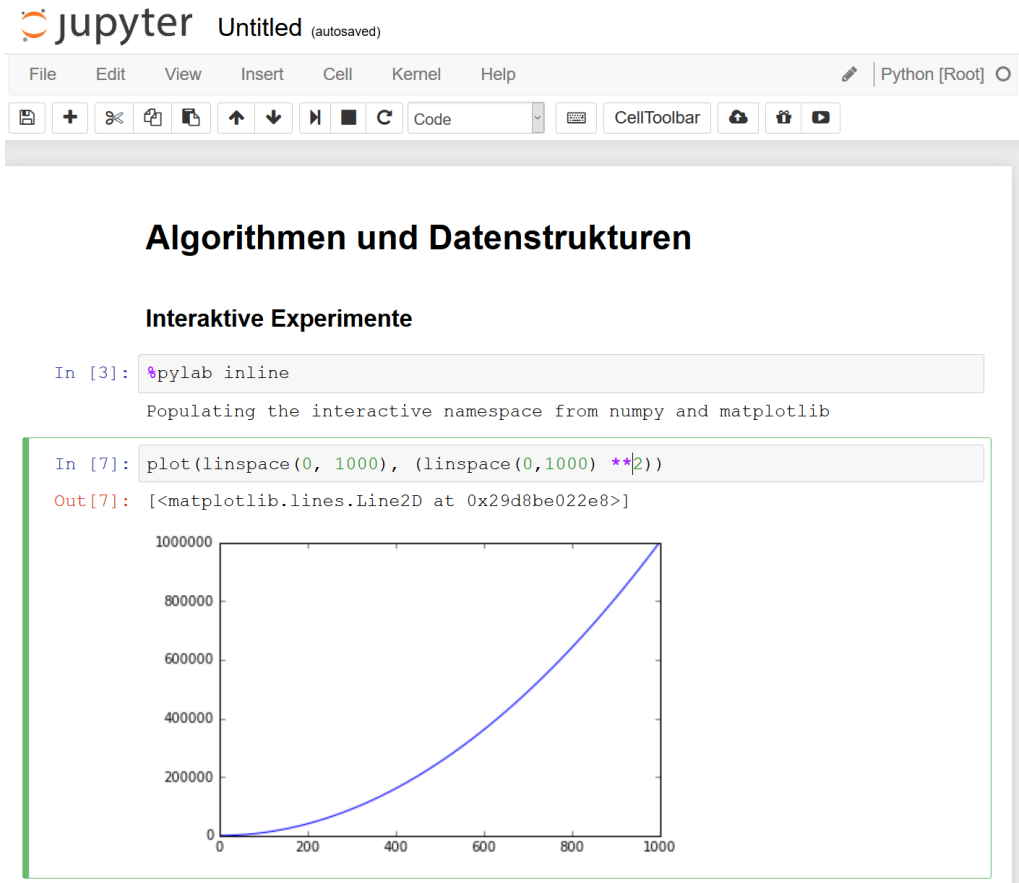
Optimalen präfixfreien Trie bauen

1. Frequenz von jedem Zeichen in String zählen
2. Wald von Bäumen (ein Baum pro Zeichen) generieren
3. Solange noch nicht alle Bäume verbunden sind:
   - Verbinde Bäume mit kleinster Frequenz

Optimalen präfixfreien Trie bauen



Optimalen präfixfreien Trie bauen



Jupyter Notebooks: Compression.ipynb

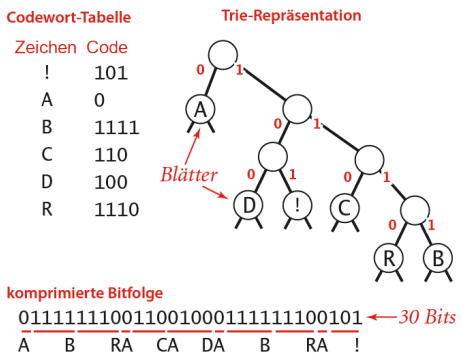
Huffman Komprimieren / Dekomprimieren

Komprimieren

- Symboltabelle mit Zeichen als Werte und Codes als Schlüssel
- String mithilfe der Symboltabelle codieren

Dekomprimierung

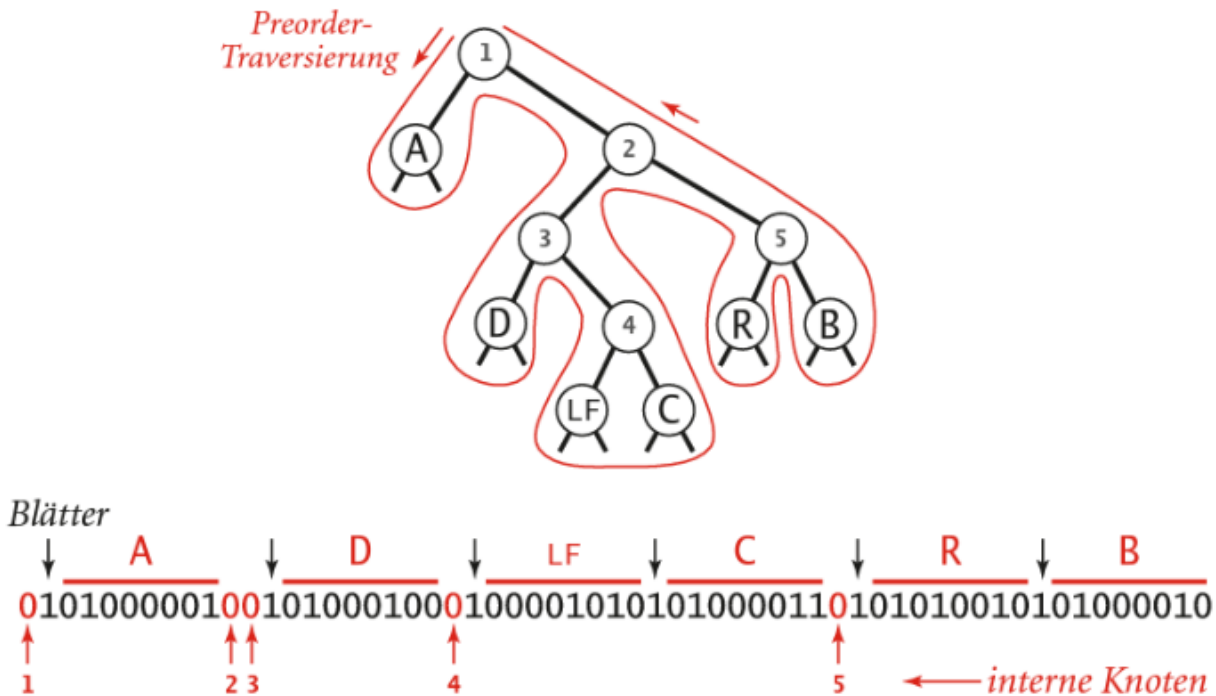
- Dem Code im Trie folgen um String zu decodieren



Quelle: Algorithmen, Sedgewick & Wayne, Abbildung 5.64

Wie weiss Empfänger, welcher Code benutzt wurde?

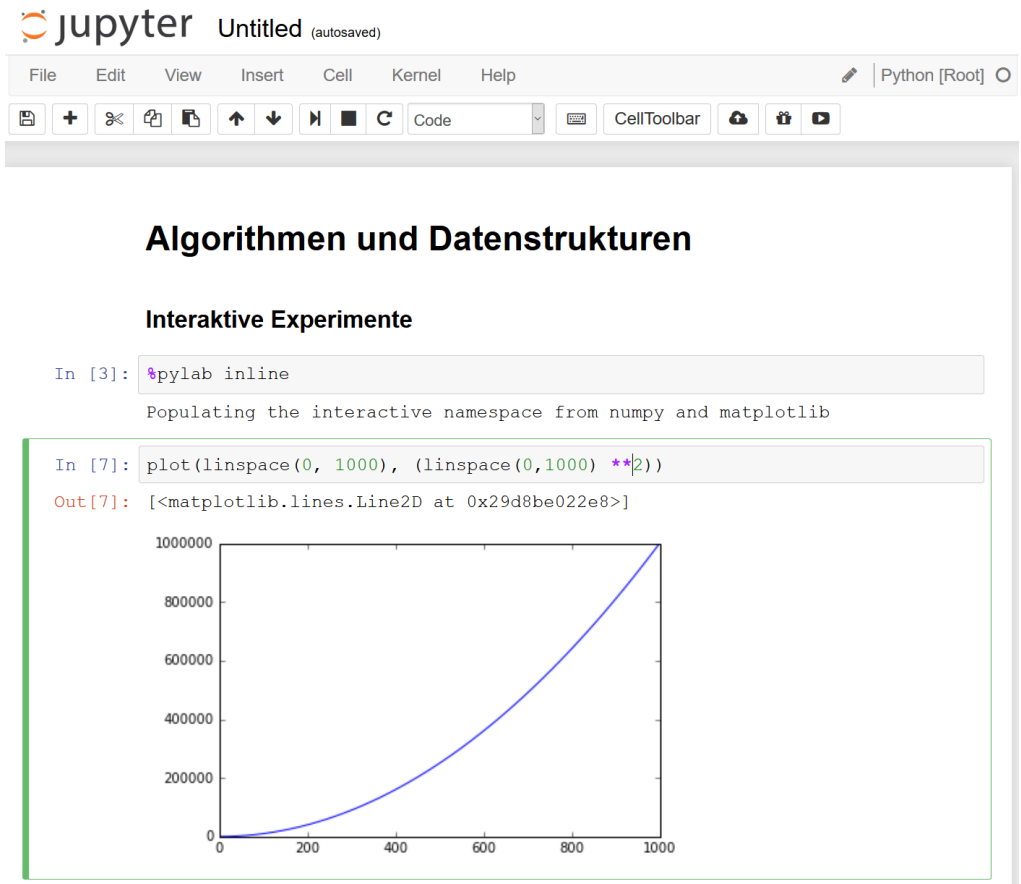
Trie schreiben und lesen



Quelle: Algorithmen, R. Sedgewick & K. Wayne, Abbildung 5.68

- Inorder Traversierung
  - Vor innerem Knoten: 0 Einfügen
  - Vor Blatt: 1 Einfügen
- Wert in Blatt (z.B. als 8 Bit Ascii code) speichern.

Beispiel und Implementation



Jupyter Notebooks: Compression.ipynb

Huffman Komprimierung: Zusammenfassung

Baut für jede Message einen speziellen Code

Kompression

- Nachricht lesen
- Code aufbauen
- Präfixfreien Code (Trie) in Datei/Bitstream schreiben
- Nachricht komprimieren

Dekompression

- Präfixfreien Code (Trie) aus Datei/Bitstream lesen
- Nachricht dekomprimieren

Huffman Komprimierung: Anwendungen

- Aber: Moderne Algorithmen nutzen eher Arithmetic Coding

Witten, Ian H., Radford M. Neal, and John G. Cleary. "Arithmetic coding for data compression." Communications of the ACM 30.6 (1987).