Praktikum Rechnerarchitektur
Abschlussvortrag

Huffmankodierung

Aufgabenbereich Theorie der Informationsverarbeitung

Amir Taghizadegan Uros Maletic Lukas Weng

Gliederung

1. Einleitung: Huffmankodierung

- verlustfreies Datenkompressionsverfahren
- generiert präfix-freien Code
- Grundprinzip: Häufig auftretende Symbole werden durch kurze Codewörter und selten auftretende Symbole durch lange Codewörter beschrieben.

1. Einleitung: Huffmanalgorithmus

1. Häufigkeitsanalyse
2. Erstellung des Wörterbuchs
3. Übersetzung des Eingabeworts

2. Lösungsansatz

int huffman_encode(char *data, char *result, unsigned int result_size)

Vorstellung der Häufigkeitsanalyse am Beispieleingabewort

AADZDAADDZAAAADAZZD

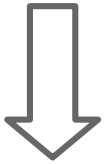
2.1 Häufigkeitsanalyse

Zeichen 0 bis 15 aus dem data-Array:

A	A	D	Z	D	A	A	D	D	Z	A	A	A	A	D	A
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Maske, die in jedem Byte das zu vergleichende Zeichen enthält:

A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



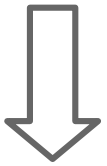
Maske, die in den übereinstimmenden Bytes -1 und sonst 0 enthält:

-1	-1	0	0	0	-1	-1	0	0	0	-1	-1	-1	-1	0	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2.1 Häufigkeitsanalyse

Maske, die in den übereinstimmenden Bytes -1 und sonst 0 enthält:

-1	-1	0	0	0	-1	-1	0	0	0	-1	-1	-1	-1	0	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15



Data-Array nach Vergleich:

-1	-1	D	Z	D	-1	-1	D	D	Z	-1	-1	-1	-1	D	-1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

2.1 Häufigkeitsanalyse

Ergebnis der Häufigkeitsanalyse für das Beispiel *AADZDAADDZAAAADAZZD*:

Zeichen	Häufigkeit
A	9
D	6
Z	4

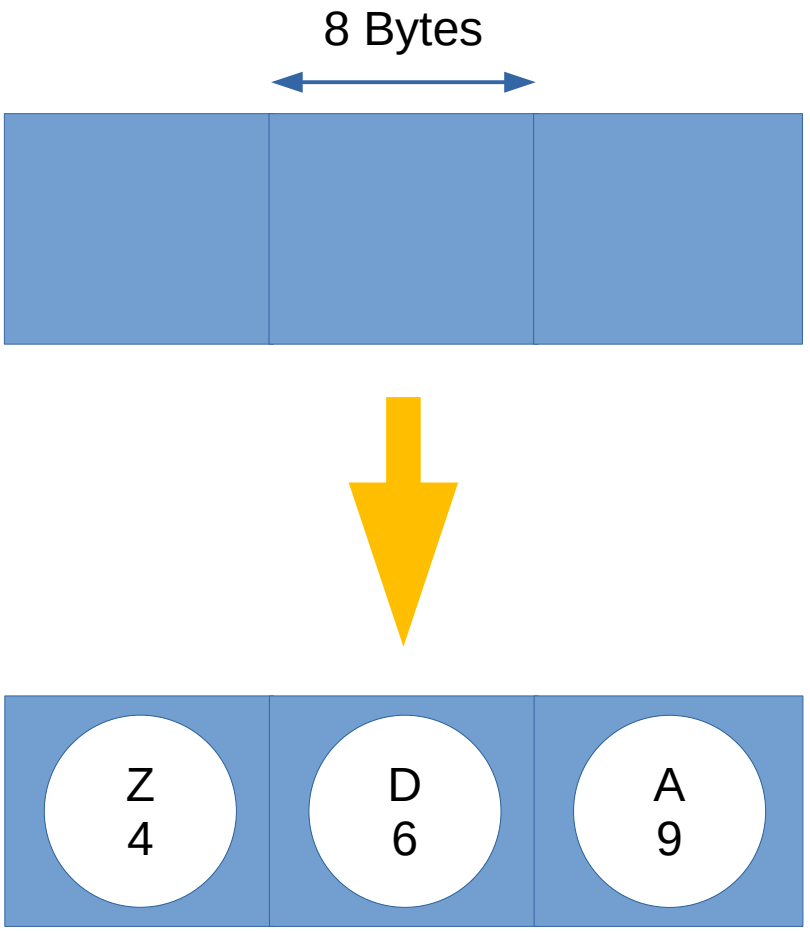
2.1 Häufigkeitsanalyse: Fehlerbehandlung

- ein Zeichen kommt zu häufig vor
- ein Zeichen wird nicht von der 7-Bit-ASCII-Variante kodiert
- das result-Array ist zu klein

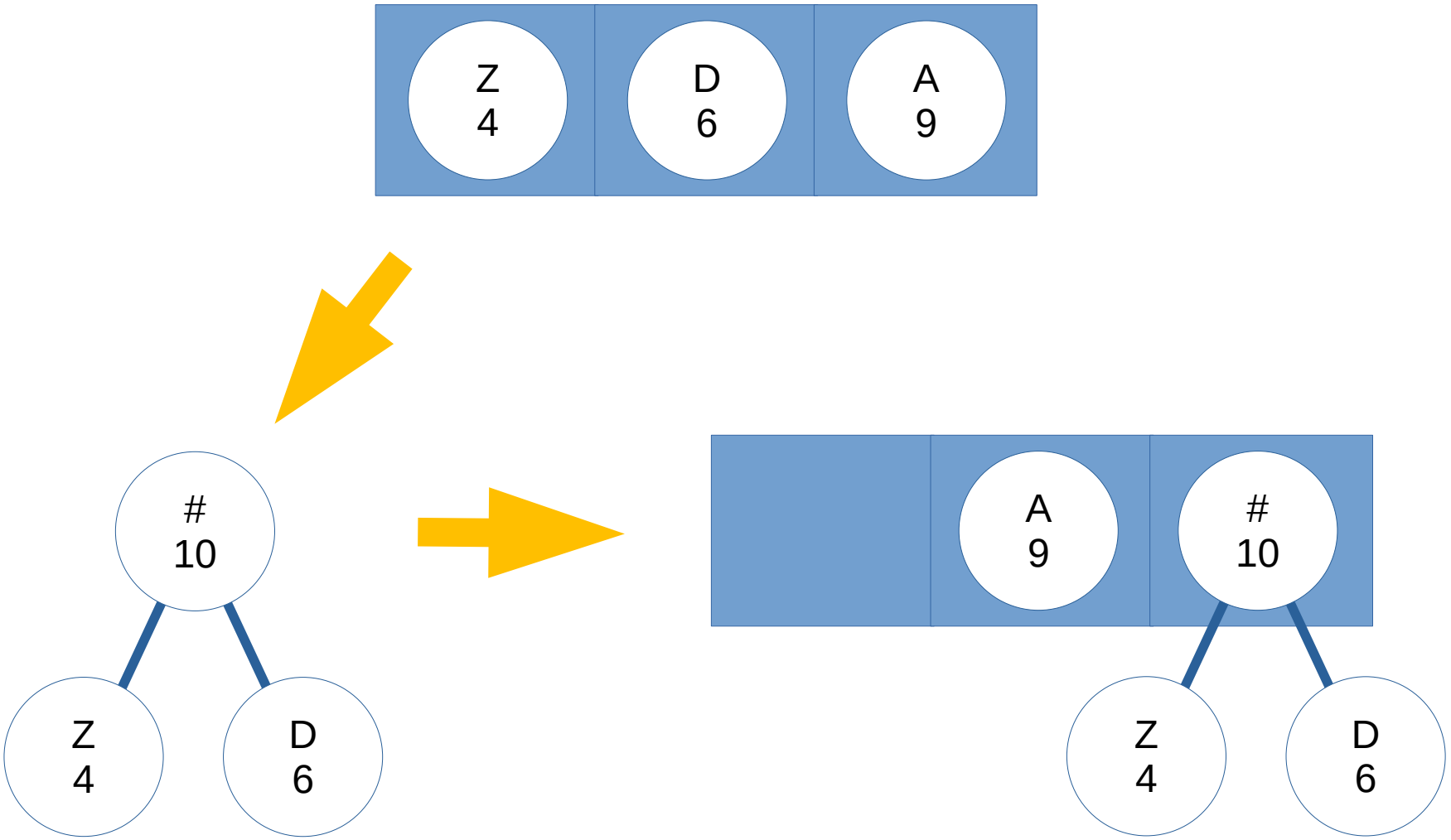
2.2 Erstellung des Wörterbuchs

1. Es wird ein Haufen erstellt und alle Elemente hinzufügt
2. Anhand des Haufens wird der Huffman-Baum erstellt
3. Der Baum wird traversiert und die Codewörter in das result-Array eingetragen

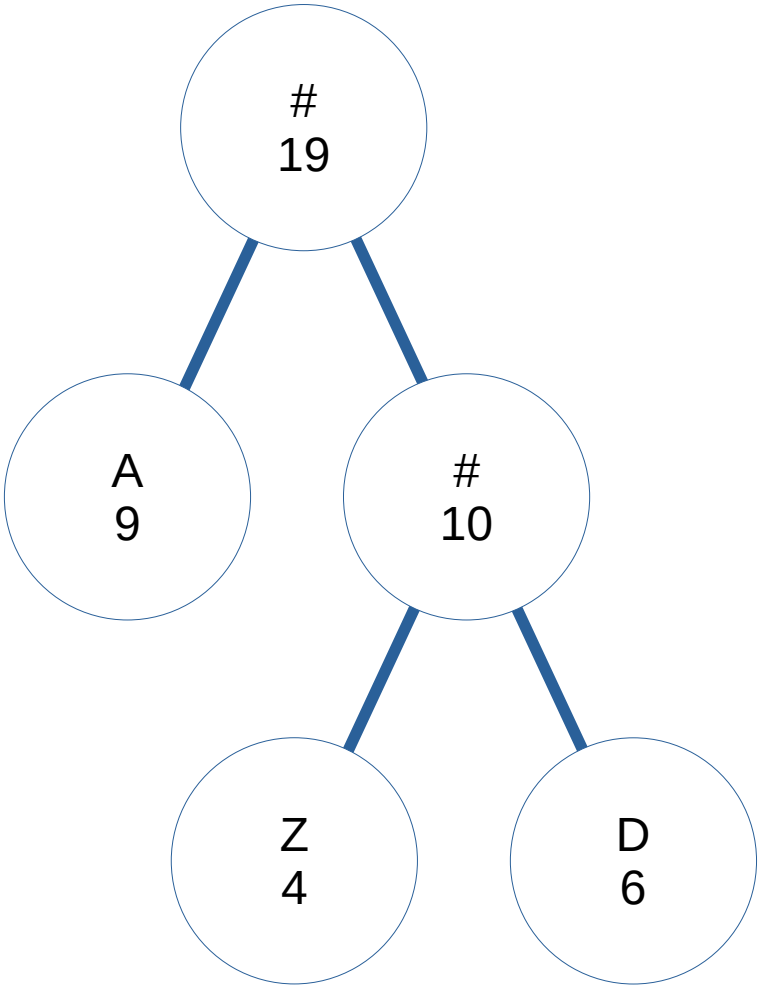
2.2 Wörterbuch: Erstellung des Heaps



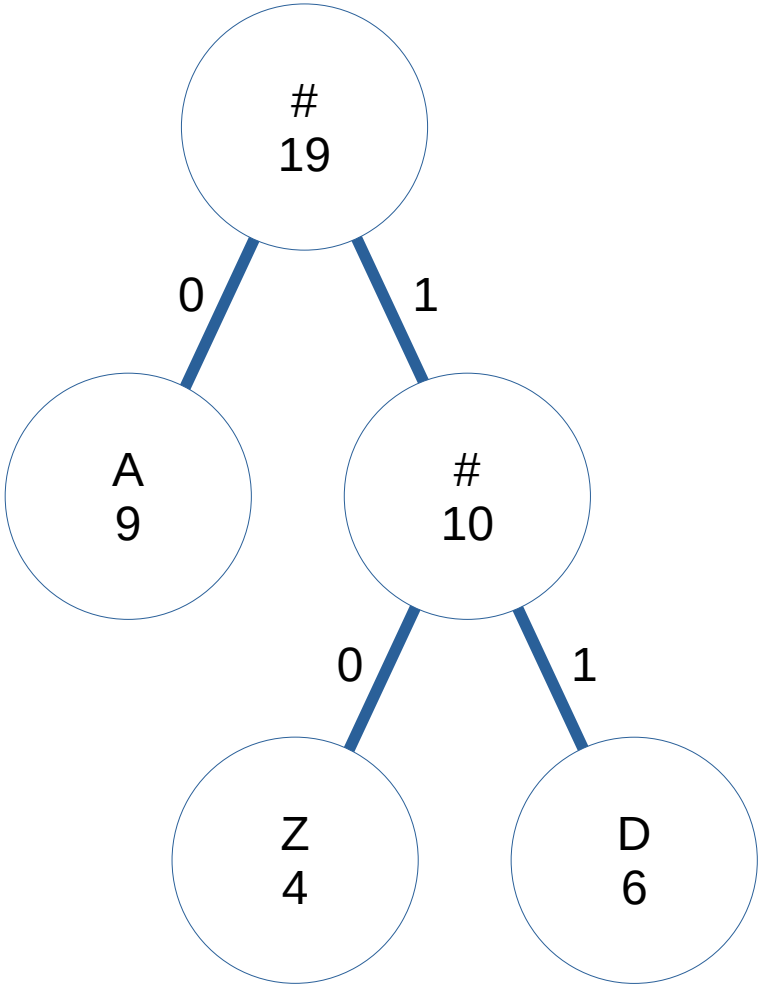
2.2 Wörterbuch: Erstellung des Baums



2.2 Wörterbuch: Finaler Baum



2.2 Wörterbuch: Traversierung



Buchstabe	Codewort	Im Speicher
A	0	0100 0001 1111 1110
Z	10	1101 1010 0000 0010
D	11	1100 0100 0000 0011

2.3 Übersetzung des Eingabeworts

1. Ein Quellsymbol im Eingabewort wird gelesen.

AADZDAADDZAAAADAZZD

--------↑----------

Nächstes Quellsymbol: Z ≙ ***01011010***

2.3 Übersetzung des Eingabeworts

2. Der Wörterbuch Eintrag zu dem Zeichen wird gesucht.

Wörterbuch:

01000001	11111110	11011010	00000010	11000100	00000011
Eintrag zu ‘A’		Eintrag zu ‘Z’		Eintrag zu ‘D’	

Aktuelles Quellsymbol: *’Z’* ≙ *0**1011010***

Modifiziertes Quellsymbol: *’Z̃’* ≙ ***01011010***

2.3 Übersetzung des Eingabeworts

2. Der Wörterbuch Eintrag zu dem Zeichen wird gesucht.

Wörterbuch:

01000001	11111110	1**1011010**	00000010	11000100	00000011
Eintrag zu ‘A’		Eintrag zu ‘Z’		Eintrag zu ‘D’	

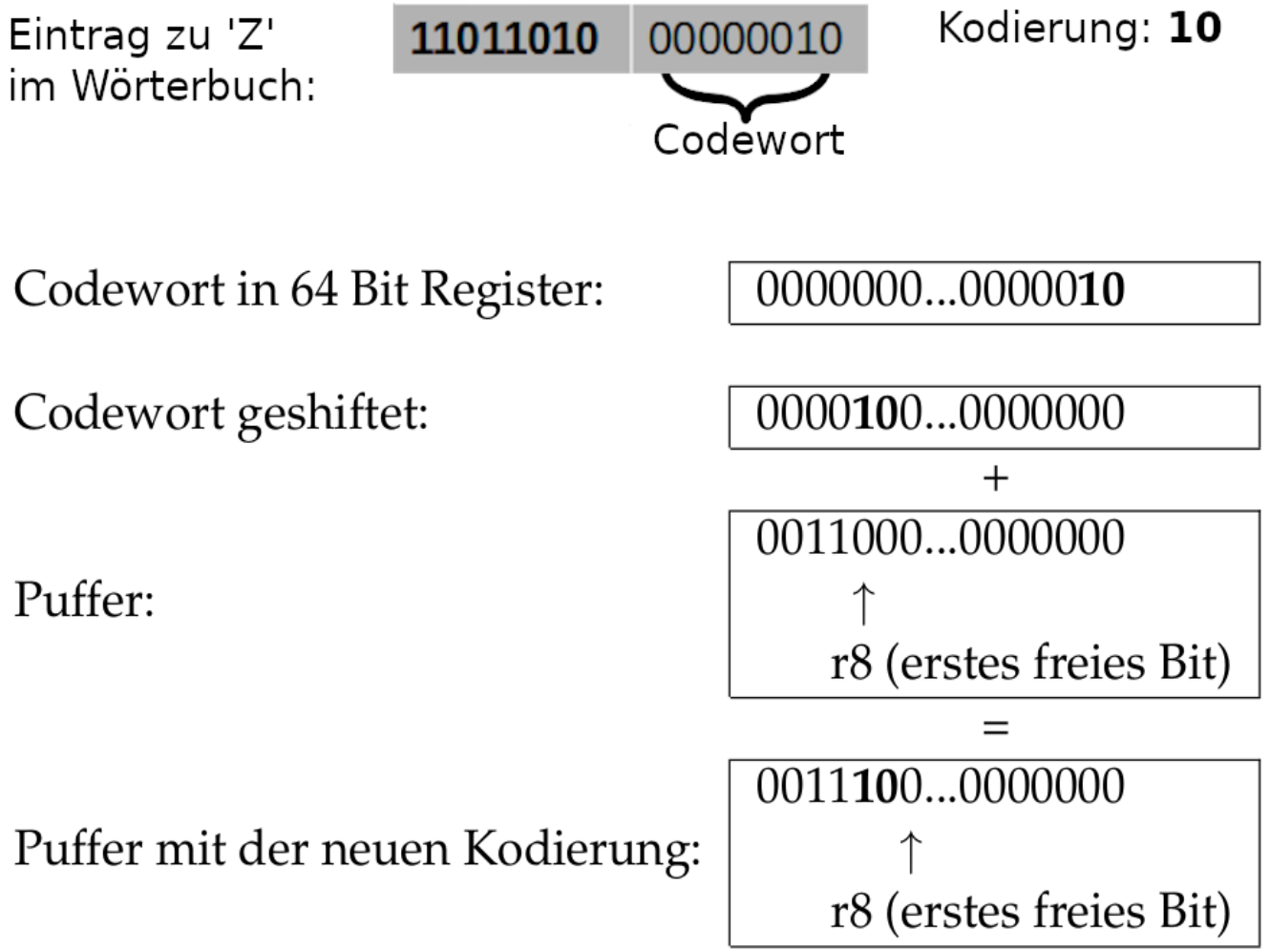
Aktuelles Quellsymbol: ’*Z*’ ≙ *0**1011010***

Modifiziertes Quellsymbol: ’$\tilde{Z}$’ ≙ ***1**1011010*

=>passenden Eintrag gefunden

2.3 Übersetzung des Eingabeworts

3,4. Das Codewort wird auf 64 Bit erweitert, angepasst und in den Puffer eingefügt.



2.3 Übersetzung des Eingabeworts

Die Schritte 1-4 werden solange wiederholt, bis das nächste Quellsymbol *’\0’* ≙ *00000000* ist.

2.3 Übersetzung des Eingabeworts

5. Puffer in das result-Array kopieren

- Kann mitten in der Schleife und nach der Beendigung der Schleife passieren

result-Array:

2.3 Übersetzung des Eingabeworts

6. Kodierungsende markieren und Rückgabewert bestimmen.

1.Fall: Gesamte Kodierung endet mit 1

0000001**1**	00111011	00111110	00001101	01011**000**

↑

Kodierung endet hier

2.Fall: Gesamte Kodierung endet mit 0

↓

0000000**0**1	00111011	00111110	00001101	01010**111**

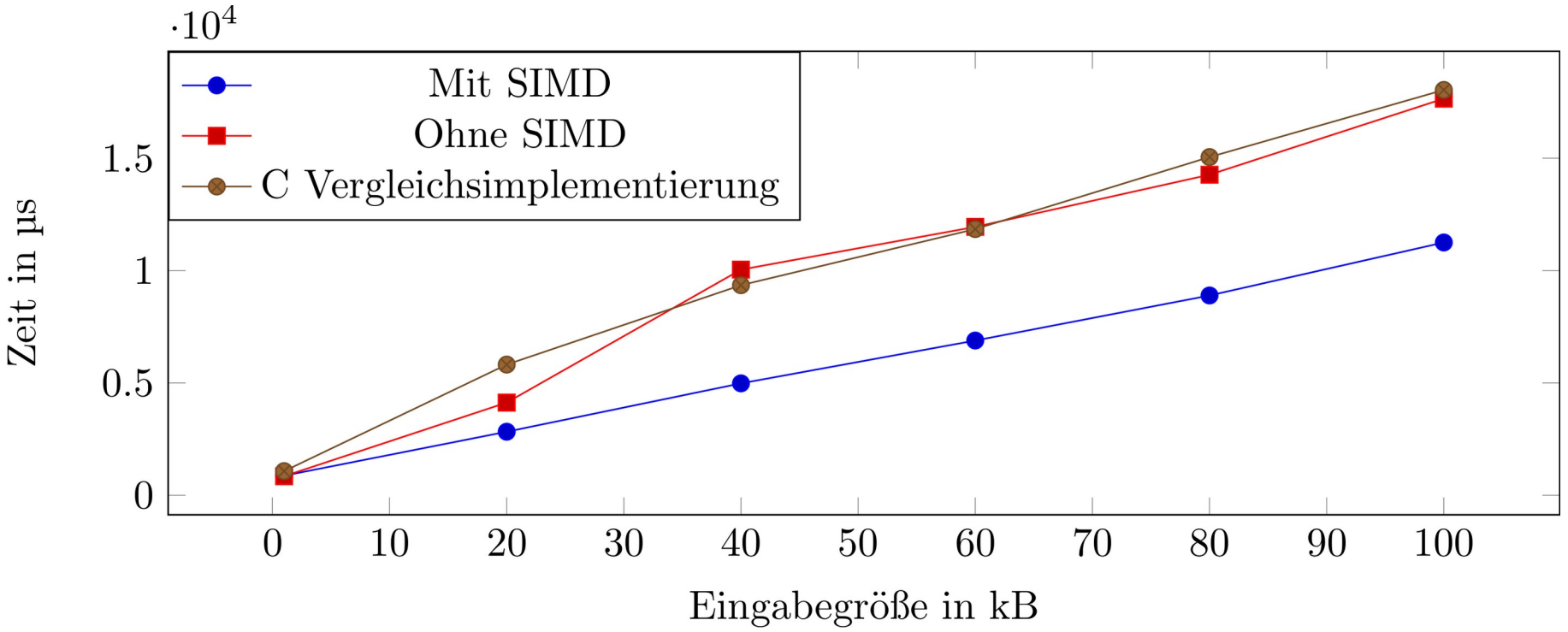
Rückgabewert beim Erfolgsfall: Länge der gesamten Kodierung in Bytes

Rückgabewert bei anderen Fällen (z.B result-Array zu klein): **-1**

3. Korrektheit: Testfälle

- Beispiel aus der Aufgabenstellung: *ABRAKADABRAB*
- Überprüfung von Randfällen und der Fehlerbehandlung
  - Eingabewort der Länge 0, Eingabewort der Länge 1, ein Zeichen mehrmals wiederholt
  - mindestens ein Zeichen im Eingabewort ist nicht in der 7-Bit-ASCII-Variante enthalten
  - ein Zeichen kommt im Eingabewort mehr als 65535 mal vor
- Eingaben in natürlicher Sprache zur Überprüfung des Wörterbuchs und der Kodierung

4. Performanzanalyse



5.1 Zusammenfassung

- Kodierung von Daten in 7-Bit-ASCII-Darstellung mit dem Huffmanalgorithmus
- automatisierte Tests und Zeitmessungen für jeweils vordefinierte Eingaben
- Huffmankodierung lohnt sich nicht immer
- wichtig aus der Performanzanalyse:
  - SIMD lohnt sich erst bei großen Eingaben
  - deutlicher Performanzvorteil von Assemblerimplementierung gegenüber der Implementierung in einer Hochsprache

5.2 Ausblick

- Erweiterung der Kodierungsfunktion: Funktion zur Analyse der Eingaben
  - Größe des result-Arrays vorberechnen
  - entscheiden ob die Huffmankodierung sinnvoll ist
- Weitere Kodierungsverfahren, die gegensätzlich für Huffmankodierung gut funktionieren:
  - Bsp.: "ABCDEFGHIJKLMMMMMMMMMMMM…"
    → Lauflängenkodierung
- Sortierung der Zeichen mit gleichen Häufigkeiten

6. Quellen

- Thomas Borys. Codierung und Kryptologie. Vieweg + Teubner Research. Wissenschaft. Vieweg & Teubner, Wiesbaden, 1. aufl. edition, 2011.
- Wilfried Dankmeier. Grundkurs Codierung. Springer Fachmedien Wiesbaden, Wiesbaden, 4. aufl. 2017 edition, 2017.
- Joachim Goll. C als erste Programmiersprache. Springer Vieweg, Wiesbaden, 8., überarb. und erw. aufl. edition, 2014.
- D. A. Huffman. A method for the construction of minimum-redundancy codes. Proceedings of the IRE, 40(9):1098–1101, 1952.
- Ralph-Hardo Schulz. Codierungstheorie. Vieweg+Teubner Verlag, Wiesbaden, 1991.