Aufgabe 1: Schmucknachrichten

Teilnahme-ID: 73796

Bearbeiter/-in dieser Aufgabe:
Constantin Reinhold

10. February 2025

Inhaltsverzeichnis

1. Lösungsidee

Das Ziel ist es möglichst kurze, Präfixcodes zu erstellen. Also Codes bei denen kein Trennzeichen benötigt wird, weil sie eindeutig eingelesen werden können. Dafür werden wird Huffman-Kodierung[1] verwendet. Die Huffman Kodierung wurde von David Huffman entwickelt und verwendet standardmäßig einen Binärbaum um einen eindeutigen Pfad für jedes Symbol zu schaffen, welcher durch die Indeces der Teilbäume einen Präfixcode baut. Da in diesem Fall das Codealphabet größer als zwei ist wird eine Variation der Huffman-Codes mit n-ary Bäumen[2] und erweiterndem Codealphabet eingesetzt. Diese sorgt für eine effizientere Datenspeicherung und ermöglicht uns mit mehr als 2 Perlen zu kodieren.

streichen
Komma
Indices
Komma

Für den zweiten Teil der Aufgabe, in der präfixfrei Codes für Perlen gefunden werden sollen, die nicht alle gleich groß sind wird eine andere Herangehensweise benötigt. Statt einen Huffmanbaum zu erstellen, der die n-kleinsten Knoten zusammenfasst erstellen wir einen n-ary Baum wobei der Knotenwert immer das Gewicht des n-ten Symbols im Codealphabet + das der Knotenwert des Mutterknoten ist. Jede dieser Erweiterungen erweitert den Baum somit um n-1 Blätter. Wenn wir davon ausgehen, dass wir mit einem einzelnen Wurzelknoten mit dem Wert 0 starten und das Blatt mit dem kleinsten Wert erweitern können die benötigten Erweiterungen mit der Formel: $\frac{j-1}{n-1}$, wobei j die benötigte Anzahl an Blätterknoten bzw. Die Anzahl an benötigten eindeutigen Kombination ist. Wir erhalten einen Baum in dem der Pfad in zu jedem Blatt einen eindeutigen, präfixfreien Code repräsentiert. Dieser Ansatz erzeugt eine sub-optimale, jedoch funktionale und schnelle Lösung. Im Fall von schmuck5 ist die Lösung ca. 0.7% von der optimalen Lösung erfernt. Im Fall von schmuck9 sind es ca. 15%. Dies spricht dafür, dass der Lösungsansatz eine exponential sub-optimale Lösung generiert. Ein optimaler Ansatz würde nicht jedes mal n neue Blätter erzeugen sondern entweder durch einen Vorhersagealgorithmus verwenden um die Anzahl an Blätter zu bestimmen,um die erweitert wird oder einen rekursiven Charakter haben bei dem später Pfade wieder gelöscht, und durch eine kürzere Alternative ersetzt werden. Denn jedesmal um n Blätter zu erweitern sorgt dafür, dass wir Blätter mit einem Gewicht von k erzeugen, statt einen Pfad zu mit mehreren Perlen zu erstellen der ein Gewicht < k hat. Stellen wir uns ein Codealphabet C = {A, B,

Knotenwert immer die Summe des Gewichts ... Codealphabet und des Knotenwertes des Mutterknotens ist.
hier fehlt ein Verb
die
Kombinationen
"in" streichen
entfernt
Komma
exponentiell
Komma löschen
Ausdruck
Komma löschen

1 https://de.wikipedia.org/wiki/Huffman-Kodierung

2 https://en.wikipedia.org/wiki/Huffman_coding#n-ary_Huffman_coding

C} mit Gewichten {1, 2, 3} vor. Ein Code AAA hat das gleiche Gewicht wie C. Haben wir also die Entscheidungen ob wir einen Knoten B zu BAAA oder BAC erweitern sollten schein BAC zwar kürzer hat aber tatsächlich ein Gewicht von 6 anstatt BAAA mit einem Gewicht von 5.

Möglichkeit, ob
scheint
"sollten" streichen, danach Komma

2. Laufzeitanalyse

Insgesamt hat die Implementierung eine Laufzeit von O(L+N+TlogT) wobei L die Anzahl an Zeichen in der Nachricht (len(message)), N die Anzahl an unterschiedlichen Perlen also klein n im Programm, und T die Anzahl der verschiedenen Zeichen in der Nachricht ist.

Hyphen setzen
Hyphen setzen, ohne Komma

Best-Case: O(L), falls wenige verschiedene Zeichen vorkommen.

Worst-Case: O(L log L), falls die Nachricht aus komplett einzigartigen Zeichen besteht.

3. Implementierung

...es aus, im Ordner...

Um das Programm auszuführen reicht im Ordner mit der Quelldatei folgenden Befehl auszuführen.

```
Python aufgabe1.py “<beispiel>.txt”
```

z.B. *python aufgabe1.py “schmuck0.txt”* für Beispiel schmuck0.

Das Programm funktioniert folgendermaßen:

1. Beispieldatei einlesen → read_file()

2. Entscheiden ob das Beispiel Teilaufgabe a) oder b) ist.

3. Häufigkeiten zählen → count_frequencies()

4. Baum erstellen → TreeWithHeap

5.1 Teilaufgabe a): Baum fusionieren → collapse_tree()

5.2 Teilaufgabe b): Baum erweitern bis genug Blattknoten vorhanden sind → tree.find_min_leaf(), tree.transform_and_add_children()

6. Alle möglichen Pfade finden → find_all_paths()

7. Codes nach Häufigkeit und Gewichtung zuordnen → assign_codes()

8. Nachricht kodieren und Ergebnis ausgeben → encode_message()

Zusätzlich hat die TreeWithHeap Klasse eine Methode print_tree() welche mit tree.print_tree() aufgerufen werden kann und den Baum (tree) in einem nutzerfreundlichen Weg mit Hilfe von Einzügen und ASCII-Charaktern darstellt.

3.1 Node Klasse

3.1.1 __init__

```
def __init__(self, value, is_leaf=True):
        self.value = value
        self.is_leaf = is_leaf
        self.children = []
        self.parent = None
```

Jeder Knoten (Node) hat genau vier Eigenschaften. Value speichert einen Integer, der den Wert des Knotens wiederspiegelt. is_leaf ist ein boolischer Wert, der bestimmt ob ein Knoten ein Blatt ist. Children ist eine Liste, die andere Knoten enthält. Die Unterknoten in children sind die Kinder des Knotens. Sollte ein Knoten ein Blatt sein, ist diese Liste immer leer. Parent speichert den Mutterknoten.

3.1.2 __lt__

```
def __lt__(self, other):
        return self.value < other.value
```

__lt__ ist eine Methode, die bestimmt, wie die Werte von zwei Knoten verglichen werden.

3.2 TreeWithHeap Klasse

3.2.1 __init__

```
def __init__(self, is_weighted, frequencies= {}):
        if is_weighted:
            self.root = Node(0)
            self.leaf_heap = [(self.root.value, self.root)]# Min-heap of leaf nodes
        else:
            self.root = Node(0)
            self.top_layer = []
            for item in frequencies.items():
                self.node = Node(item[1])
                self.top_layer.append(self.node)
```

Diese Methode bestimmt den Startwert des Baumes. Sollten wir einen Baum mit gewichteten Knoten erstellen wollen wird der Baum mit einem einzelnen Wurzelknoten erstellet. Andernseits, werden n Kinder, mit den Frequenzen als Werte unter dem Wurzelknoten erstellt.

3.2.2 add_parent

```
def add_parent(self, children, value):
        new_parent = Node(value, is_leaf=False)
        for child in children:
            if child.parent is not None:
                child.parent.children.remove(child)
            child.parent = new_parent
            new_parent.children.append(child)
            if child in self.top_layer:
                self.top_layer.remove(child)
        self.top_layer.append(new_parent)
        return new_parent
```

Erstellt einen neuen Knoten mit value als Wert. Dieser Knoten wird dann als Mutterknoten für alle Knoten in children eingesetzt. Sollte ein Knoten in children bereits einen Mutterknoten haben, wird

diese Verbindung entfernt und durch den neuen Knoten ersetzt. Die Methode gibt außerdem den neuen Mutterknoten zurück.

3.2.3 add_child

```
def add_child(self, parent, value):
        """
        Adds a new child to the given parent.
        """
        new_node = Node(value)
        new_node.parent = parent
        parent.children.append(new_node)
        return new_node
```

Erstellt einen neuen Knoten und setzt in als Kind in children eines angegeben Mutterknotens ein. Gibt außerdem den neuen Knoten zurück.

3.2.4 find_n_smallest

```
def find_n_smallest(self, n):
        if n <= 0:
            return []

        return heapq.nsmallest(n, self.top_layer, key=lambda node: node)
```

Gibt die N kleinsten Knoten auf der höchsten Eben zurück, wobei höchste Ebene alle Knoten bezeichnet die keine Kinder haben. Dies wird mit Hilfe der heapq Klasse implementiert die Implementation für die Heap Datenstruktur umfasst.

3.2.5 find_min_leaf

```
def find_min_leaf(self):
        if not self.leaf_heap:
            return None
        return self.leaf_heap[0][1]
```

Gibt das Blatt mit dem kleinsten Wert zurück.

3.2.6 transform_and_add_children

```
def transform_and_add_children(self, node, new_values):
        # Transform the leaf node into an internal node
        node.is_leaf = False
        # Remove it from the heap
        self.leaf_heap = [(val, n) for val, n in self.leaf_heap if n != node]
        heapq.heapify(self.leaf_heap)

        # Add new children
        for value in new_values:
            # New node's value = parent's value + added value
            new_node_value = node.value + value
            new_node = Node(new_node_value)
            node.children.append(new_node)
            heapq.heappush(self.leaf_heap, (new_node.value, new_node))
```

Diese Methode ist der Grundbaustein für die Generation eines Baumes mit n Blattknoten. Node ist dabei immer der Blattknoten mit dem kleinsten Wert. Dieser wird zu einem internen Knoten umgewandelt und es werden len(new_values) neue Kinderknoten erstellt. Der Wert jedes neuen Knotes ist dabei der Wert aus new_values + der Wert des ursprünglichen Knotens / Mutterknoten. Somit repräsentiert jedes Blatt direkt die Kosten für seinen Pfad und diese müssen später nicht erneut beim Pfadfinden berechnet werden. Dadurch, dass immer nur die höchste Ebene als Heap gespeichert wird, wird der ursprüngliche Knoten aus diesem Heap gelöscht und die Kinder neu hinzugefügt.

3.2.7 find_all_paths

```
def find_all_paths(self, is_weighted=False):
        paths = []

        def dfs(node, current_path):
            if node.is_leaf:
                if is_weighted:
                    paths.append((current_path, node.value))
                else:
                    if len(current_path) == 0:
                        current_path = "0"
                        path_length = 1
                    else:
                        path_length = len(current_path)
                    paths.append((current_path, path_length))
                return

            for index, child in enumerate(node.children):
                dfs(child, current_path + str(index_mapping(index)))

        dfs(self.root, "")
        return paths
```

find_all_paths ist besonders, dadurch, dass sie sowohl die gewichteten als auch ungewichteten Bäume durchlaufen, und ihre entsprechenden Pfad finden kann. Die Implementierung in diesem Fall ist eine rekursive Funktion, die entsprechenden alle Pfad durchläuft und am Ende zurückgibt. Dabei wird immer ein Tuple zurückgegeben. Sollte ein gewichteter Baum vorliegen ist, dass ein Paar aus Pfad und Blattwert (Pfadkosten). In einem ungewichteten Baum ist es der Pfad und die Länge des Pfades, da jede Kante Kosten von 1 hat.

3.3 read_file

```
def read_file():
    with open(directory+filename, encoding='utf-8') as file:
        content = file.readlines()
        content = [line.strip() for line in content]
    n = int(content[0])
    if n == 1:
        with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
            file.write("Es muss mindestens 2 Perlen geben!")
        raise ValueError('Es muss mindestens 2 Perlen geben!')
    pearls = [int(number) for number in content[1].split()]
    message = content[2]

    return n, pearls, message
```

Liest die Beispieldatei ein und gibt die Anzahl an verschiedenen Perlen (N) als Integer, die Größen der Perlen (pearls) als Liste und die zu kodierende Nachricht (message) als String zurück. Außerdem behandelt sie Edge Case 1. Sollte nur eine Perle vorhanden sein, gibt die Funktion eine Fehlermeldung aus und stop somit die Ausführung des Programmes.

3.4 count_frequencies

```
def count_frequencies(frequencies={}):
    for char in message:
        if char not in frequencies:
            frequencies[char] = 1
        else:
            frequencies[char] += 1
    return frequencies, len(frequencies)
```

Zählt die absolute Häufigkeit jedes Charakters in der zu kodierenden Nachricht und speichert sie ein einem Dictionary, für einfache Aufrufe, ab. Der Schlüssel ist dabei der Charakter und der Wert ist die Häufigkeit.

3.5 collapse_tree

```
def collapse_tree(tree, n):
    n_smallest = tree.find_n_smallest(n)
    if len(n_smallest) == 1:
        return tree
    tree.add_parent(n_smallest, sum([node.value for node in n_smallest]))
    collapse_tree(tree, n)
    return tree
```

Diese Funktion ist der Grundbaustein für die Erstellung von ungewichteten Huffman Bäumen mit erweitertem Codealphabet. Die Funktion ist rekursiv und fusioniert in jedem Durchlauf die N kleinsten Knoten auf der höchsten Ebene zu einem Unterbaum und erstellt einen neuen Mutterknoten für sie. Somit wird die Anzahl an Knoten auf der höchsten Ebene in jedem Durchlauf um n-1 Knoten reduziert bis nur noch ein Knoten vorhanden ist. Diese Knoten ist der Wurzelknoten des Baumes und die Fusionierung stoppt, wobei der fusionierte Baum zurückgegeben wird.

3.6 index_mapping

```
def index_mapping(index):
    if index < 10:
        return index
    elif index >= 10 and index <= 36:
        return chr(index+55)
    elif index >= 37 and index <= 62:
        return chr(index+60)
    else:
        raise ValueError("Wie kann das sein?")
```

Diese Funktion ist dafür zuständig Integer in base62 zu übersetzen, was dafür sorgt, dass mit insgesamt 62 verschiedenen Perlen, statt 10, kodiert werden kann. Das liegt daran, dass Zahlen in

jedem base-System ab 2 Stellen nicht mehr eindeutig dekodiert werden können. Zum Beispiel könnte die base10 Zahl 928 folgendermaßen interpretiert werden:

1. 928

2. 9 und 28

3. 92 und 8

4. 9 und 2 und 8

3.7 assign_codes

```
def assign_codes(paths, frequencies, total_characters):
    paths = sorted(paths, key=lambda x: x[1])[:total_characters]
    frequencies = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)
    codes = {}
    for i in range(len(frequencies)):
        codes[frequencies[i][0]] = paths[i]
    return codes
```

Die assign_codes Funktion hat drei Parameter. Paths, eine Liste von allen möglichen Pfaden in einem Baum und ihren Gewichtungen; Frequencies, ein Dictionary mit allen Symbolen und ihren absoluten Häufigkeiten; total_characters, ein Integer der angibt wie viele eindeutige Codes benötigt werden. Dabei gilt total_characters = len(frequencies). Als erstes löscht assign_codes die n Pfade mit den höchsten Kosten wobei gilt: n = len(paths)-total_characters. Die Funktion weist jedem Symbol in frequencies einen eindeutigen Code aus paths zu. Dabei erhalten die Symbole mit der höchsten Häufigkeit die Pfade mit dem kleinsten Gewicht. Somit wird die Länge der kodierten Nachricht minimiert. Diese Zuweisungen werden in einem Dictionary gespeichert und zurückgegeben.

3.8 encode_message

```
def encode_message(codes):
    encoded = ""
    encoded_length = 0
    for char in message:
        encoded += codes[char][0]
        encoded_length += codes[char][1]
    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write(str(codes)+"\n"+f"Kodierte Textlänge {encoded_length}")
    print(codes)
    print(f"Kodierte Textlänge {encoded_length}")
```

Kodiert die Nachricht und speichert die Länge der kodierten Nachricht und die Codetabelle in einer Ausgabedatei ab.

3.9 n_ary_huffman

```
def n_ary_huffman(n):
    frequencies, total_characters = count_frequencies()
    tree = TreeWithHeap(False, frequencies)
    collapse_tree(tree, n)
    tree.root = tree.top_layer[0]
    paths = tree.find_all_paths()
```

```
    codes = assign_codes(paths, frequencies, total_characters)
    encode_message(codes)
```

Ruft alle Funktionen die für die Teilaufgabe a mit n-ary Huffmanbäumen nötig sind in der richtigen Reihenfolge auf.

3.10 weighted_huffman

```
def weighted_huffman(n, pearls, message):
    frequencies, total_characters = count_frequencies()
    needed_expansions = (total_characters-1)/(n-1)
    tree = TreeWithHeap(True)
    for i in range(math.ceil(needed_expansions)):
        min_leaf_node = tree.find_min_leaf()
        tree.transform_and_add_children(min_leaf_node, pearls)
    paths = tree.find_all_paths(True)
    codes = assign_codes(paths, frequencies, total_characters)
    encode_message(codes)
```

Ruft alle Funktionen die für die Teilaufgabe b mit gewichteten Bäumen nötig sind in der richtigen Reihenfolge auf.

3.11 Funktionsaufruf

```
n, pearls, message = read_file()
n_not_1 = len([x for x in pearls if x != 1])
if n > 62: #Es wird maximal mit 62 Perlen kodiert
        n = 62
if n_not_1 == 0:
    n_ary_huffman(n)
else:
    weighted_huffman(n, pearls, message)
```

Hier wird Edge Case 2 behandelt. Sollte die Anzahl n an Perlen größer als 62, sein wird diese Zahl auf 62 reduziert, so dass eine valide Kodierung erfolgen kann. Danach wird die für die Beispieldatei entsprechende Funktion n_ary_huffman oder weighted_huffman aufgerufen.

3.12 Edge cases

1. Es ist nur 1 Perle vorhanden:

→ Die Nachricht kann nicht kodiert werden. Eine Fehlermeldung wird ausgegeben.

2. Es sind mehr als 62 Perlen vorhanden:

→ Die Nachricht wird mit maximal 62 Perlen kodiert.

3. Die Nachricht hat nur ist nur 1 Charakter lang oder enthält nur 1 Art von Charakter:

→ Der Code ist für diesen Charakter ist immer “0”

4. Beispiele

Für alle Beispiele konnte eine Lösung generiert werden auf wenn die Lösungen für Teilaufgabe b) mit unterschiedlich großen Perlen nicht optimal gelöst werden konnten.

Alle Tests wurden auf einem Laptop mit den folgenden Spezifikationen durchgeführt:

OS: Windows 11

CPU: Intel Pentium Silver N600 @1.10GhZ

RAM: 4 GB

GPU: Intel UHD Graphics

Python: 3.9.13

Alle Ausgabe folgenden dem selben Aufbau.

Die erste Zeile ist immer die Codetabelle in Form eines Dictionaries mit dem Unicode-Symbol als Schlüssel und einem Tuple als Wert. Der Tuple enthält an erster Stelle den Code für das Symbol und an zweiter Stelle die Kosten für diesen Code. Die Codes sind in base62 angegeben: [0, 1, …, 9, A, B, …, Z, a, b, …, z]. Da die Codetabelle in diesem Format nicht nutzerfreundlich lesbar ist wird sie in den folgenden Beispielen tabelarisch dargestellt. Um die Ausgaben in kurz zu halten werden immer nur die ersten 20 Zeichen dargestellt. Die gesamte Ausgabe ist im Anhang auffindbar.

In der zweiten Zeile ist die Länge der kodierten Nachricht auffindbar.

4.1 schmuck0

Symbol	**Code**	**Kosten**
E	000	3
‘ ‘	001	3
T	010	3
N	110	3
S	111	3
D	0110	4
O	0111	4
L	1000	4
R	1001	4
M	1010	4
C	10110	5

H	10111	5

Kodierte Textlänge: 113 Runtime: 88.567 milli seconds

4.2 schmuck00

Symbol	Code	Kosten
‘ ‘	00	2
e	01	2
i	10	2
t	021	3
n	022	3
s	110	3
h	111	3
c	120	3
I	0200	4
a	0201	4
r	0202	4
d	1120	4
D	1121	4
u	1212	4
w	1220	4
m	1222	4
G	12220	5
g	11221	5
E	11222	5
k	12100	5

Kodierte Textlänge: 420 Runtime: 79.019 milli seconds

4.3 schmuck01

Symbol	Code	Kosten
‘ ‘	0	1
e	2	1
n	10	2

r	12	2
i	13	2
s	30	2
l	31	2
a	32	2
t	33	2
h	40	2
c	41	2
o	43	2
g	110	3
m	111	3
u	112	3
.	113	3
d	114	3
k	140	3
E	141	3
,	142	3

Kodierte Textlänge: 1150 Runtime: 110.384 milli seconds

4.4 schmuck1

Symbol	**Code**	**Kosten**
e	001	3
t	011	3
‘ ‘	100	3
r	101	3
i	110	3
n	111	3
w	20	3
B	21	3
T	0000	4
s	0001	4
f	002	4
“	0100	4

b	0101	4
W	012	4
N	020	4
F	021	4
h	102	4
ü	112	4
D	120	4
u	121	4

Kodierte Textlänge: 194 Runtime: 64.155 milli seconds

4.5 schmuck2

Symbol	Code	Kosten
a	000000	6
‘ ‘	01	6
b	001	7
c	100	7
d	0001	8
e	00001	9
f	000001	10
g	11	10
h	101	11

Kodierte Textlänge: 266 Runtime: 66.028 milli seconds

4.6 schmuck3

Symbol	Code	Kosten
a	000	3
b	01	3
c	10	3
‘ ‘	2	3
d	001	4
e	02	4
f	11	4
g	002	5
h	12	5

Kodierte Textlänge: 337 Runtime: 68.169 milli seconds

4.7 schmuck4

Symbol	Code	Kosten
a	001	7
b	00000000	8
c	0001	8
d	0100	8
e	1000	8
f	00001	9
g	000001	10
h	11	10
i	0000001	11
j	011	11
k	101	11
l	00000001	12
m	0101	12
n	1001	12

Kodierte Textlänge: 137 Runtime: 68.306 milli seconds

4.8 schmuck5

Symbol	Code	Kosten
‘ ‘	00	2
e	010	3
t	001	3
i	02	3
o	100	3
s	101	3
a	110	3
n	111	3
r	12	3
c	20	3
d	21	3
l	3	3

m	012	4
h	03	4
u	102	4
p	112	4
f	13	4
b	22	4
y	4	4
g	013	5

Kodierte Textlänge: 3185 Runtime: 64.569 milli seconds

4.9 schmuck6

Symbol	**Code**	**Kosten**
'	00000	5
文	0001	5
馬	0010	5
。	002	5
有	0100	5
古	011	5
英	1000	5
雄	101	5
未	12	5
遇	200	5
時	21	5
都	00001	6
無	0002	6
大	0011	6
志	0101	6
非	012	6
止	0200	6

鄧	021	6
禹	1001	6
希	102	6

Kodierte Textlänge: 234 Runtime: 89.268 milli seconds

4.10 schmuck7

Symbol	Code	Kosten
‘ ‘	00	2
e	01	2
n	02	2
i	03	2
r	04	2
s	05	2
a	06	2
d	10	2
h	11	2
t	12	2
u	13	2
l	14	2
c	15	2
g	16	2
m	20	2
o	21	2
b	22	2
,	23	2
f	24	2
w	25	2

Kodierte Textlänge: 165728 Runtime: 103.466 milli seconds

4.11 schmuck8

Symbol	Code	Kosten

，	00000	5
。	00001	5
「	00010	5
」	00011	5
》	0003	5
:	00100	5
《	00101	5
之	00110	5
格	00111	5
不	0012	5
有	0013	5
為	0020	5
云	0021	5
時	004	5
其	01000	5
一	01001	5
武	01010	5
相	01011	5
人	0102	5
詩	0103	5

Kodierte Textlänge: 3428 Runtime: 79.832 milli seconds

4.12 schmuck9

Symbol	Code	Kosten
の	000000000	9
た	00000001	9
に	00000010	9
い	0000002	9

し	00000100	9
と	0000011	9
を	0000020	9
て	000003	9
は	00001000	9
‘ ‘	0000101	9
な	0000110	9
。	000012	9
が	0000200	9
る	000021	9
で	00010000	9
こ	0001001	9
か	0001010	9
っ	000102	9
ら	0001100	9
う	000111	9

Kodierte Textlänge: 42224 Runtime: 146.457 milli seconds

4.13 long_message_a

Input:

36

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Das hier ist meine erste BWINF-Teilnahme. Ich hätte nie gedacht, dass ich soweit kommen würde. Jetzt sind wir schon in der zweiten Runde; verrückt oder? Danke für alles BWINF Team <3

Symbol	**Code**	**Kosten**
‘ ‘	0	1
e	1	1

i	2	1
t	30	2
n	31	2
s	32	2
r	33	2
a	34	2
h	35	2
d	36	2
m	37	2
c	38	2
o	39	2
w	3A	2
I	3B	2
l	3C	2
k	3D	2
ü	3E	2
D	3F	2
B	3G	2

Kodierte Textlänge: 300 Runtime: 67.049 milli seconds

4.14 long_message_b

Input:

36

1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 5 6 6 6 6 6 6 7

Das hier ist meine erste BWINF-Teilnahme. Ich hätte nie gedacht, dass ich soweit kommen würde. Jetzt sind wir schon in der zweiten Runde; verrückt oder? Danke für alles BWINF Team <3

Symbol	**Code**	**Kosten**
‘ ‘	1	1
e	2	1
i	3	1
t	4	1
n	5	1

s	6	1
r	7	1
a	8	1
h	9	1
d	A	1
m	B	1
c	00	2
o	01	2
w	02	2
I	03	2
l	04	2
k	05	2
ü	06	2
D	07	2
B	08	2

Kodierte Textlänge: 252 Runtime: 61.677 milli seconds

4.15 short_message

Input:

2

1 1

a

Symbol	**Code**	**Kosten**
a	0	1

Kodierte Textlänge: 1 Runtime: 63.906 milli seconds

4.16 single

Input:

1

1

Single

ValueError: Es muss mindestens 2 Perlen geben!

Quellcode

```
class Node:
    def __init__(self, value, is_leaf=True):
        self.value = value
        self.is_leaf = is_leaf
        self.children = []
        self.parent = None
    def __lt__(self, other):
        return self.value < other.value

class TreeWithHeap:
    def __init__(self, is_weighted, frequencies= {}):
        if is_weighted:
            self.root = Node(0)
            self.leaf_heap = [(self.root.value, self.root)]# Min-heap of leaf nodes
        else:
            self.root = Node(0)
            self.top_layer = []
            for item in frequencies.items():
                self.node = Node(item[1])
                self.top_layer.append(self.node)

    def add_parent(self, children, value):
        new_parent = Node(value, is_leaf=False)
        for child in children:
            if child.parent is not None:
                child.parent.children.remove(child)
            child.parent = new_parent
            new_parent.children.append(child)
            if child in self.top_layer:
                self.top_layer.remove(child)
        self.top_layer.append(new_parent)
        return new_parent

    def add_child(self, parent, value):
        new_node = Node(value)
        new_node.parent = parent
        parent.children.append(new_node)
        return new_node

    def find_n_smallest(self, n):
        if n <= 0:
            return []
        return heapq.nsmallest(n, self.top_layer, key=lambda node: node)

    def find_min_leaf(self):
        if not self.leaf_heap:
            return None
        return self.leaf_heap[0][1]

    def transform_and_add_children(self, node, new_values):
        node.is_leaf = False
        self.leaf_heap = [(val, n) for val, n in self.leaf_heap if n != node]
        heapq.heapify(self.leaf_heap)
        for value in new_values:
            new_node_value = node.value + value
            new_node = Node(new_node_value)
            node.children.append(new_node)
```

```
            heapq.heappush(self.leaf_heap, (new_node.value, new_node))

    def find_all_paths(self, is_weighted=False):
        paths = []

        def dfs(node, current_path):
            if node.is_leaf:
                if is_weighted:
                    paths.append((current_path, node.value))
                else:
                    if len(current_path) == 0:
                        current_path = "0"
                        path_length = 1
                    else:
                        path_length = len(current_path)
                    paths.append((current_path, path_length))
                return
            for index, child in enumerate(node.children):
                dfs(child, current_path + str(index_mapping(index)))
        dfs(self.root, "")
        return paths

def read_file():
    with open(directory+filename, encoding='utf-8') as file:
        content = file.readlines()
        content = [line.strip() for line in content]
    n = int(content[0])
    if n == 1:
        with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
            file.write("Es muss mindestens 2 Perlen geben!")
        raise ValueError('Es muss mindestens 2 Perlen geben!')
    pearls = [int(number) for number in content[1].split()]
    message = content[2]
    return n, pearls, message

def count_frequencies(frequencies={}):
    for char in message:
        if char not in frequencies:
            frequencies[char] = 1
        else:
            frequencies[char] += 1
    return frequencies, len(frequencies)

def collapse_tree(tree, n):
    n_smallest = tree.find_n_smallest(n)
    if len(n_smallest) == 1:
        return tree
    tree.add_parent(n_smallest, sum([node.value for node in n_smallest]))
    collapse_tree(tree, n)
    return tree

def index_mapping(index):
    if index < 10:
        return index
    elif index >= 10 and index <= 36:
        return chr(index+55)
    elif index >= 37 and index <= 62:
        return chr(index+60)
    else:
        raise ValueError("Wie kann das sein?")

def assign_codes(paths, frequencies, total_characters):
    paths = sorted(paths, key=lambda x: x[1])[:total_characters]
    frequencies = sorted(frequencies.items(), key=lambda x: x[1], reverse=True)
    codes = {}
```

```
    for i in range(len(frequencies)):
        codes[frequencies[i][0]] = paths[i]
    return codes

def encode_message(codes):
    encoded = ""
    encoded_length = 0
    for char in message:
        encoded += codes[char][0]
        encoded_length += codes[char][1]
    with open("./ausgaben/"+filename+".out", "w", encoding='utf-8') as file:
        file.write(str(codes)+"\n"+f"Kodierte Textlänge {encoded_length}")
    print(codes)
    print(f"Kodierte Textlänge {encoded_length}")

def n_ary_huffman(n):
    frequencies, total_characters = count_frequencies()
    tree = TreeWithHeap(False, frequencies)
    collapse_tree(tree, n)
    tree.root = tree.top_layer[0]
    paths = tree.find_all_paths()
    codes = assign_codes(paths, frequencies, total_characters)
    encode_message(codes)

def weighted_huffman(n, pearls, message):
    frequencies, total_characters = count_frequencies()
    needed_expansions = (total_characters-1)/(n-1)
    tree = TreeWithHeap(True)
    for i in range(math.ceil(needed_expansions)):
        min_leaf_node = tree.find_min_leaf()
        tree.transform_and_add_children(min_leaf_node, pearls)
    paths = tree.find_all_paths(True)
    codes = assign_codes(paths, frequencies, total_characters)
    encode_message(codes)

n, pearls, message = read_file()
n_not_1 = len([x for x in pearls if x != 1])
if n > 62: #Es wird maximal mit 62 Perlen kodiert
        n = 62
if n_not_1 == 0:
    n_ary_huffman(n)
else:
    weighted_huffman(n, pearls, message)
```

Anhang

Schmuck0

{'E': ('000', 3), ' ': ('001', 3), 'I': ('010', 3), 'N': ('110', 3), 'S': ('111', 3), 'D': ('0110', 4), 'O': ('0111', 4), 'L': ('1000', 4), 'R': ('1001', 4), 'M': ('1010', 4), 'C': ('10110', 5), 'H': ('10111', 5)}

Kodierte Textlänge: 113

schmuck00

{' ': ('00', 2), 'e': ('01', 2), 'i': ('10', 2), 't': ('021', 3), 'n': ('022', 3), 's': ('110', 3), 'h': ('111', 3), 'c': ('120', 3), 'l': ('0200', 4), 'a': ('0201', 4), 'r': ('0202', 4), 'd': ('1120', 4), 'D': ('1121', 4), 'u': ('1212', 4), 'w': ('1220', 4), 'm': ('1222', 4), 'G': ('11220', 5), 'g': ('11221', 5), 'E': ('11222', 5), 'k': ('12100', 5), 'f':

('12101', 5), 'W': ('12102', 5), 'Z': ('12110', 5), 'P': ('12111', 5), 'o': ('12112', 5), '?': ('12210', 5), 'A': ('12211', 5), 'b': ('12212', 5)}

Kodierte Textlänge: 420

schmuck01

{' ': ('0', 1), 'e': ('2', 1), 'n': ('10', 2), 'r': ('12', 2), 'i': ('13', 2), 's': ('30', 2), 'l': ('31', 2), 'a': ('32', 2), 't': ('33', 2), 'h': ('40', 2), 'c': ('41', 2), 'o': ('43', 2), 'g': ('110', 3), 'm': ('111', 3), 'u': ('112', 3), '.': ('113', 3), 'd': ('114', 3), 'k': ('140', 3), 'E': ('141', 3), ',': ('142', 3), 'D': ('143', 3), 'b': ('144', 3), 'w': ('340', 3), 'ü': ('342', 3), 'f': ('343', 3), 'S': ('344', 3), 'v': ('420', 3), 'I': ('422', 3), 'p': ('423', 3), 'O': ('424', 3), 'V': ('440', 3), 'ö': ('441', 3), 'z': ('442', 3), 'F': ('443', 3), 'B': ('444', 3), 'ä': ('3410', 4), 'G': ('3411', 4), 'K': ('3412', 4), 'R': ('3413', 4), 'M': ('3414', 4), 'ß': ('4210', 4), 'H': ('4211', 4), 'N': ('4212', 4), 'W': ('4213', 4), '…': ('4214', 4)}

Kodierte Textlänge: 1150

schmuck1

{'e': ('001', 3), 't': ('011', 3), ' ': ('100', 3), 'r': ('101', 3), 'i': ('110', 3), 'n': ('111', 3), 'w': ('20', 3), 'B': ('21', 3), 'I': ('0000', 4), 's': ('0001', 4), 'f': ('002', 4), '"': ('0100', 4), 'b': ('0101', 4), 'W': ('012', 4), 'N': ('020', 4), 'F': ('021', 4), 'h': ('102', 4), 'ü': ('112', 4), 'D': ('120', 4), 'u': ('121', 4), 'd': ('22', 4), 'o': ('0002', 5), 'm': ('0102', 5), 'a': ('022', 5), 'k': ('122', 5)}

Kodierte Textlänge: 194

schmuck2

{'a': ('000000', 6), ' ': ('01', 6), 'b': ('001', 7), 'c': ('100', 7), 'd': ('0001', 8), 'e': ('00001', 9), 'f': ('000001', 10), 'g': ('11', 10), 'h': ('101', 11)}

Kodierte Textlänge: 266

schmuck3

{'a': ('000', 3), 'b': ('01', 3), 'c': ('10', 3), ' ': ('2', 3), 'd': ('001', 4), 'e': ('02', 4), 'f': ('11', 4), 'g': ('002', 5), 'h': ('12', 5)}

Kodierte Textlänge: 337

schmuck4

{'a': ('001', 7), 'b': ('00000000', 8), 'c': ('0001', 8), 'd': ('0100', 8), 'e': ('1000', 8), 'f': ('00001', 9), 'g': ('000001', 10), 'h': ('11', 10), 'i': ('0000001', 11), 'j': ('011', 11), 'k': ('101', 11), 'l': ('00000001', 12), 'm': ('0101', 12), 'n': ('1001', 12)}

Kodierte Textlänge: 137

schmuck5

{' ': ('00', 2), 'e': ('010', 3), 't': ('011', 3), 'i': ('02', 3), 'o': ('100', 3), 's': ('101', 3), 'a': ('110', 3), 'n': ('111', 3), 'r': ('12', 3), 'c': ('20', 3), 'd': ('21', 3), 'l': ('3', 3), 'm': ('012', 4), 'h': ('03', 4), 'u': ('102', 4), 'p': ('112',

4), 'f': ('13', 4), 'b': ('22', 4), 'y': ('4', 4), 'g': ('013', 5), '.': ('04', 5), 'v': ('103', 5), 'x': ('113', 5), 'q': ('14', 5), ',': ('23', 5), 'w': ('5', 5), 'T': ('014', 6), 'F': ('05', 6), 'S': ('104', 6), '-': ('114', 6), 'A': ('15', 6), 'H': ('24', 6), '1': ('6', 6), '9': ('015', 7), '5': ('06', 7), '2': ('105', 7), 'z': ('115', 7), 'G': ('16', 7), '’': ('25', 7), 'j': ('016', 8), ';': ('106', 8)}

Kodierte Textlänge: 3185

schmuck6

{'､': ('00000', 5), '文': ('0001', 5), '馬': ('0010', 5), '。': ('002', 5), '有': ('0100', 5), '古': ('011', 5), '英': ('1000', 5), '雄': ('101', 5), '未': ('12', 5), '遇': ('200', 5), '時': ('21', 5), '都': ('00001', 6), '無': ('0002', 6), '大': ('0011', 6), '志': ('0101', 6), '非': ('012', 6), '止': ('0200', 6), '鄧': ('021', 6), '禹': ('1001', 6), '希': ('102', 6), '學': ('1100', 6), '、': ('111', 6), '武': ('201', 6), '望': ('22', 6), '督': ('00002', 7), '郵': ('0012', 7), '也': ('0102', 7), '晉': ('0201', 7), '公': ('022', 7), '妻': ('1002', 7), '不': ('1101', 7), '肯': ('112', 7), '去': ('202', 7), '齊': ('0202', 8)}

Kodierte Textlänge: 234

schmuck7

{' ': ('00', 2), 'e': ('01', 2), 'n': ('02', 2), 'i': ('03', 2), 'r': ('04', 2), 's': ('05', 2), 'a': ('06', 2), 'd': ('10', 2), 'h': ('11', 2), 't': ('12', 2), 'u': ('13', 2), 'l': ('14', 2), 'c': ('15', 2), 'g': ('16', 2), 'm': ('20', 2), 'o': ('21', 2), 'b': ('22', 2), ',': ('23', 2), 'f': ('24', 2), 'w': ('25', 2), '.': ('26', 2), 'k': ('30', 2), 'z': ('31', 2), 'S': ('32', 2), 'ä': ('33', 2), 'ü': ('34', 2), 'ö': ('35', 2), 'v': ('40', 2), 'p': ('41', 2), 'F': ('42', 2), 'ß': ('43', 2), 'D': ('44', 2), 'E': ('45', 2), 'V': ('46', 2), 'W': ('50', 2), 'N': ('51', 2), 'K': ('52', 2), 'B': ('53', 2), 'A': ('54', 2), 'M': ('55', 2), 'H': ('56', 2), 'I': ('60', 2), 'L': ('61', 2), 'R': ('62', 2), 'G': ('63', 2), 'T': ('64', 2), 'Z': ('65', 2), 'j': ('66', 2), 'J': ('7', 2), '-': ('07', 3), ';': ('17', 3), '(': ('27', 3), ')': ('360', 3), ':': ('361', 3), '_': ('362', 3), 'O': ('363', 3), 'U': ('364', 3), '[': ('365', 3), ']': ('366', 3), '„': ('37', 3), '“': ('47', 3), 'P': ('57', 3), '1': ('67', 3), '4': ('8', 3), '3': ('08', 4), '?': ('18', 4), '2': ('28', 4), '!': ('367', 4), '5': ('38', 4), '6': ('48', 4), '0': ('58', 4), 'q': ('68', 4), 'Ä': ('9', 4), '8': ('09', 5), '7': ('19', 5), '9': ('29', 5), 'Q': ('368', 5), '’': ('39', 5), 'Ö': ('49', 5), 'x': ('59', 5), 'y': ('69', 5), 'C': ('369', 6)}

Kodierte Textlänge: 165728

schmuck8

{'､': ('00000', 5), '。': ('00001', 5), '「': ('00010', 5), '」': ('00011', 5), '》': ('0003', 5), '：': ('00100', 5), '《': ('00101', 5), '之': ('00110', 5), '格': ('00111', 5), '不': ('0012', 5), '有': ('0013', 5), '為': ('0020', 5), '云': ('0021', 5), '時': ('004', 5), '其': ('01000', 5), '一': ('01001', 5), '武': ('01010', 5), '相': ('01011', 5), '人': ('0102', 5), '詩': ('0103', 5), '風': ('01100', 5), '也': ('01101', 5), '公': ('01110', 5), '是': ('01111', 5), '知': ('0112', 5), '在': ('0113', 5), '誰': ('0120', 5), '；': ('0121', 5), '未': ('0130', 5), '都': ('0131', 5), '、': ('014', 5), '馬': ('0200', 5), '嚴': ('0201', 5), '而': ('0210', 5), '意': ('0211', 5), '以': ('022', 5), '言': ('023', 5), '日': ('0300', 5), '十': ('0301', 5), '百': ('0310', 5), '皆': ('0311', 5), '作': ('032', 5), '者': ('033', 5), '花': ('040', 5), '天': ('041', 5), '調': ('10000', 5), '性': ('10001', 5), '情': ('10010', 5), '律': ('10011', 5), '骨': ('1002', 5), '無': ('1003', 5), '大': ('10100', 5), '志': ('10101', 5), '非': ('10110', 5), '禹': ('10111', 5), '文': ('1013', 5), '光': ('1020', 5), '與': ('1030', 5), '李': ('1031', 5), '通': ('104', 5), '尤': ('11000', 5), '目': ('11001', 5), '曰': ('11010', 5), '君': ('11011', 5), '得': ('1102', 5), '韓': ('1103', 5), '王': ('11100', 5), '後': ('11101', 5),

'見': ('11110', 5), '解': ('11111', 5), '林': ('1112', 5), '將': ('1113', 5), '開': ('1120', 5), '看': ('1121', 5), '來': ('1130', 5), '四': ('1131', 5), '此': ('114', 5), '便': ('1200', 5), '年': ('1201', 5), '已': ('1210', 5), '中': ('1211', 5), '出': ('122', 5), '七': ('123', 5), '上': ('1300', 5), '問': ('1301', 5), '濟': ('1310', 5), '才': ('1311', 5), '何': ('132', 5), '滿': ('133', 5), '自': ('140', 5), '水': ('141', 5), '外': ('2001', 5), '多': ('2010', 5), '枝': ('2011', 5), '繩': ('202', 5), '深': ('203', 5), ' ': ('2100', 5), '好': ('2101', 5), '談': ('2110', 5), '趣': ('2111', 5), '三': ('212', 5), '篇': ('213', 5), '同': ('220', 5), '乎': ('221', 5), '吟': ('230', 5), '古': ('231', 5), '英': ('24', 5), '雄': ('3000', 5), '遇': ('3001', 5), '止': ('3010', 5), '鄧': ('3011', 5), '希': ('302', 5), '學': ('303', 5), '望': ('3100', 5), '督': ('3101', 5), '郵': ('3110', 5), '晉': ('3111', 5), '妻': ('312', 5), '肯': ('313', 5), '去': ('320', 5), '齊': ('321', 5), '貧': ('330', 5), '訟': ('331', 5), '逋': ('34', 5), '租': ('400', 5), '於': ('401', 5), '奇': ('410', 5), '歸': ('411', 5), '謂': ('42', 5), '寧': ('43', 5), '耶': ('00002', 6), '窺': ('00003', 6), '盼': ('00012', 6), '榮': ('00013', 6), '蘄': ('00020', 6), '小': ('00021', 6), '卒': ('0004', 6), '士': ('00102', 6), '封': ('00103', 6), '怒': ('00112', 6), '悔': ('00113', 6), '己': ('0014', 6), '奮': ('0022', 6), '拳': ('0023', 6), '毆': ('00300', 6), '般': ('00301', 6), '鄂': ('00310', 6), '西': ('00311', 6), '辛': ('0032', 6), '丑': ('0033', 6), '元': ('01002', 6), '攬': ('01003', 6), '鏡': ('01012', 6), '老': ('01013', 6), '門': ('0104', 6), '草': ('01102', 6), '生': ('01103', 6), '詠': ('01112', 6), '懷': ('01113', 6), '猶': ('0114', 6), '如': ('0122', 6), '到': ('0123', 6), '可': ('0132', 6), '郎': ('0133', 6), '玩': ('0202', 6), '詞': ('0203', 6), '若': ('0212', 6), '料': ('0213', 6), '入': ('024', 6), '及': ('0302', 6), '省': ('0303', 6), '經': ('0312', 6), '略': ('0313', 6), '金': ('034', 6), '丞': ('042', 6), '席': ('043', 6), '心': ('10002', 6), '酬': ('10003', 6), '恩': ('10012', 6), '客': ('10013', 6), '屈': ('1004', 6), '指': ('10102', 6), '世': ('10103', 6), '登': ('10112', 6), '甲': ('10113', 6), '秀': ('10120', 6), '樓': ('10121', 6), '絕': ('1014', 6), '句': ('10210', 6), '炊': ('10211', 6), '煙': ('1022', 6), '卓': ('1023', 6), '午': ('1032', 6), '散': ('1033', 6), '輕': ('11002', 6), '絲': ('11003', 6), '萬': ('11012', 6), '家': ('11013', 6), '飯': ('1104', 6), '熟': ('11102', 6), '訊': ('11103', 6), '招': ('11112', 6), '火': ('11113', 6), '斜': ('1114', 6), '陽': ('1122', 6), '樹': ('1123', 6), '鄉': ('1132', 6), '祠': ('1133', 6), '居': ('1202', 6), '然': ('1203', 6), '侯': ('1212', 6), '命': ('1213', 6), '氣': ('124', 6), '象': ('1302', 6), '迴': ('1303', 6), '異': ('1312', 6), '張': ('1313', 6), '桐': ('134', 6), '城': ('142', 6), '則': ('143', 6), '翰': ('20000', 6), '至': ('20001', 6), '首': ('2002', 6), '最': ('2003', 6), '清': ('2012', 6), '妙': ('2013', 6), '柳': ('204', 6), '陰': ('2102', 6), '春': ('2103', 6), '曲': ('2112', 6), '暮': ('2113', 6), '山': ('214', 6), '葉': ('222', 6), '底': ('223', 6), '雙': ('232', 6), '蝴': ('233', 6), '蝶': ('3002', 6), '先': ('3003', 6), '臨': ('3012', 6), '種': ('3013', 6), '化': ('304', 6), '兩': ('3102', 6), '扈': ('3103', 6), '蹕': ('3112', 6), '憐': ('3113', 6), '龍': ('314', 6), '鐘': ('322', 6), '叟': ('323', 6), '騎': ('332', 6), '踏': ('333', 6), '冰': ('402', 6), '星': ('403', 6), '和': ('412', 6), '皇': ('413', 6), '〈': ('44', 6), '箏': ('00004', 7), '九': ('00014', 7), '霄': ('00022', 7), '近': ('00023', 7), '增': ('00104', 7), '華': ('00114', 7), '色': ('0024', 7), '野': ('00302', 7), '仗': ('00303', 7), '寶': ('00312', 7), '押': ('00313', 7), '字': ('0034', 7), '韻': ('01004', 7), '寄': ('01014', 7), '托': ('01104', 7), '遙': ('01114', 7), '二': ('0124', 7), '楊': ('0134', 7), '誠': ('0204', 7), '齋': ('0214', 7), '從': ('0304', 7), '分': ('0314', 7), '低': ('044', 7), '拙': ('10004', 7), '空': ('10014', 7), '架': ('10104', 7), '孑': ('10114', 7), '腔': ('10122', 7), '口': ('10123', 7), '易': ('10212', 7), '描': ('10213', 7), '專': ('1024', 7), '寫': ('1034', 7), '靈': ('11004', 7), '辦': ('11014', 7), '餘': ('11104', 7), '愛': ('11114', 7), '須': ('1124', 7), '半': ('1134', 7), '勞': ('1204', 7), '思': ('1214', 7), '婦': ('1304', 7), '率': ('1314', 7), '事': ('144', 7), '今': ('20002', 7), '能': ('20003', 7), '範': ('2004', 7), '圍': ('2014', 7), '否': ('2104', 7), '況': ('2114', 7), '皋': ('224', 7), '歌': ('234', 7), '國': ('3004', 7), '雅': ('3014', 7), '頌': ('3104', 7), '豈': ('3114', 7), '定': ('324', 7), '哉': ('334', 7), '許': ('404', 7), '渾': ('414', 7), '似': ('00024', 8), '成': ('00304', 8), '仙': ('00314', 8), '里': ('10124', 8), '莫': ('10214', 8), '浪': ('20004', 8)}

Kodierte Textlänge: 3428

schmuck9

{'の': ('000000000', 9), 'た': ('00000001', 9), 'に': ('00000010', 9), 'い': ('0000002', 9), 'し': ('00000100', 9), 'と': ('0000011', 9), 'を': ('0000020', 9), 'て': ('000003', 9), 'は': ('00001000', 9), '、': ('0000101', 9), 'な': ('0000110', 9), '。': ('000012', 9), 'が': ('0000200', 9), 'る': ('000021', 9), 'で': ('00010000', 9), 'こ': ('0001001', 9), 'か': ('0001010', 9), 'っ': ('000102', 9), 'ら': ('0001100', 9), 'う': ('000111', 9), 'れ': ('000120', 9), 'す': ('00013', 9), 'ー': ('0002000', 9), 'り': ('000201', 9), 'き': ('000210', 9), 'ま': ('00022', 9), 'そ': ('000300', 9), 'ン': ('00031', 9), 'く': ('00100000', 9), 'よ': ('0010001', 9), 'だ': ('0010010', 9), 'あ': ('001002', 9), 'さ': ('0010100', 9), 'も': ('001011', 9), 'わ': ('001020', 9), '・': ('00103', 9), ' ': ('0011000', 9), '\u3000': ('001101', 9), 'お': ('001110', 9), 'け': ('00112', 9), '「': ('001200', 9), '」': ('00121', 9), 'ル': ('00130', 9), 'え': ('0020000', 9), 'ち': ('002001', 9), 'ん': ('002010', 9), 'ラ': ('00202', 9), 'つ': ('002100', 9), 'ス': ('00211', 9), 'カ': ('00220', 9), '人': ('0023', 9), 'イ': ('003000', 9), '見': ('00301', 9), 'ト': ('00310', 9), 'ウ': ('0032', 9), 'レ': ('01000000', 9), 'ど': ('0100001', 9), 'や': ('0100010', 9), 'ジ': ('010002', 9), '大': ('0100100', 9), '聖': ('010011', 9), '彼': ('010020', 9), 'め': ('01003', 9), '上': ('0101000', 9), 'エ': ('010101', 9), '者': ('010110', 9), 'せ': ('01012', 9), 'ア': ('010200', 9), '思': ('01021', 9), 'フ': ('01030', 9), '出': ('0110000', 9), 'ば': ('011001', 9), 'サ': ('011010', 9), '車': ('01102', 9), '堂': ('011100', 9), '建': ('01111', 9), '海': ('01120', 9), '中': ('0113', 9), 'む': ('012000', 9), '道': ('01201', 9), 'ョ': ('01210', 9), '事': ('0122', 9), '物': ('01300', 9), 'み': ('0131', 9), 'チ': ('0200000', 9), '言': ('020001', 9), 'タ': ('020010', 9), '手': ('02002', 9), '地': ('020100', 9), '一': ('02011', 9), 'ロ': ('02020', 9), '会': ('0203', 9), 'べ': ('021000', 9), 'ク': ('02101', 9), 'げ': ('02110', 9), '部': ('0212', 9), 'マ': ('02200', 9), 'ほ': ('0221', 9), 'オ': ('0230', 9), 'ろ': ('030000', 9), 'び': ('03001', 9), '立': ('03010', 9), '水': ('0302', 9), '築': ('03100', 9), '乗': ('0311', 9), '雨': ('0320', 9), '通': ('033', 9), 'ょ': ('10000000', 9), '入': ('1000001', 9), 'ね': ('1000010', 9), '主': ('100002', 9), '任': ('1000100', 9), '司': ('100011', 9), '場': ('100020', 9), 'ル': ('10003', 9), '決': ('1001000', 9), '自': ('100101', 9), '馬': ('100110', 9), '員': ('10012', 9), 'ァ': ('100200', 9), 'ド': ('10021', 9), 'ぶ': ('10030', 9), '家': ('1010000', 9), '合': ('101001', 9), '—': ('101010', 9), '分': ('10102', 9), '気': ('101100', 9), '祭': ('10111', 9), '線': ('10120', 9), '内': ('1013', 9), '送': ('102000', 9), '後': ('10201', 9), '港': ('10210', 9), 'ざ': ('1022', 9), '市': ('10300', 9), '取': ('1031', 9), 'ッ': ('1100000', 9), '利': ('110001', 9), '明': ('110010', 9), '行': ('11002', 9), 'ご': ('110100', 9), '目': ('11011', 9), '駅': ('11020', 9), '降': ('1103', 9), '前': ('111000', 9), 'ぎ': ('11101', 9), '友': ('11110', 9), '奏': ('1112', 9), '重': ('11200', 9), '不': ('1121', 9), '陸': ('1130', 9), '四': ('120000', 9), '得': ('12001', 9), '身': ('12010', 9), '防': ('1202', 9), 'ぐ': ('12100', 9), '振': ('1211', 9), '方': ('1220', 9), '広': ('123', 9), '鉄': ('13000', 9), '外': ('1301', 9), '交': ('1310', 9), '年': ('132', 9), '運': ('2000000', 9), '屋': ('200001', 9), '式': ('200010', 9), '敷': ('20002', 9), '過': ('200100', 9), 'ワ': ('20011', 9), '着': ('20020', 9), '用': ('2003', 9), '数': ('201000', 9), '実': ('20101', 9), '側': ('20110', 9), 'ぼ': ('2012', 9), 'ひ': ('20200', 9), '若': ('2021', 9), '意': ('2030', 9), '紹': ('210000', 9), '介': ('21001', 9), '教': ('21010', 9), 'ミ': ('2102', 9), 'ノ': ('21100', 9), 'ガ': ('2111', 9), '医': ('2120', 9), 'リ': ('213', 9), '差': ('22000', 9), '安': ('2201', 9), '素': ('2210', 9), '塔': ('222', 9), '量': ('2300', 9), '無': ('231', 9), '船': ('300000', 9), 'ダ': ('30001', 9), '撃': ('30010', 9), '名': ('3002', 9), '河': ('30100', 9), '他': ('3011', 9), '民': ('3020', 9), '生': ('303', 9), '代': ('31000', 9), '考': ('3101', 9), '石': ('3110', 9), '堤': ('312', 9), '牧': ('3200', 9), '草': ('321', 9), '味': ('330', 9), '南': ('000000001', 10), '越': ('00000002', 10), '舞': ('00000011', 10), '町': ('0000003', 10), '々': ('00000101', 10), '小': ('0000012', 10), '北': ('0000021', 10), '走': ('00001001', 10), '復': ('0000102', 10), '保': ('0000111', 10), '古': ('000013', 10), '説': ('0000201', 10), '修': ('000022', 10), '商': ('0000300', 10), 'ず': ('000031', 10), '到': ('00010001', 10), '荷': ('0001002', 10), '足': ('0001011', 10), '音': ('000103', 10), 'パ': ('0001101', 10), 'じ': ('000112', 10),

'歩': ('000121', 10), 'ぞ': ('0002001', 10), '少': ('000202', 10), '散': ('000211', 10), '巨': ('00023', 10), '空': ('000301', 10), '話': ('00032', 10), '格': ('00100001', 10), '助': ('0010002', 10), '仕': ('0010011', 10), '示': ('001003', 10), 'づ': ('0010101', 10), '指': ('001012', 10), 'ゃ': ('001021', 10), '特': ('0011001', 10), '廊': ('001102', 10), '晴': ('001111', 10), '眠': ('00113', 10), '図': ('001201', 10), 'ォ': ('00122', 10), '元': ('00131', 10), '呼': ('0020001', 10), '所': ('002002', 10), '今': ('002011', 10), '岸': ('00203', 10), '二': ('002101', 10), '寄': ('00212', 10), '隊': ('00221', 10), '隻': ('003001', 10), '世': ('00302', 10), '由': ('00311', 10), '砂': ('0033', 10), '流': ('01000001', 10), '細': ('0100002', 10), '横': ('0100011', 10), '伸': ('010003', 10), '知': ('0100101', 10), '放': ('010012', 10), '羊': ('010021', 10), '権': ('0101001', 10), '持': ('010102', 10), '負': ('010111', 10), '美': ('01013', 10), '風': ('010201', 10), '波': ('01022', 10), '儀': ('01031', 10), '日': ('0110001', 10), 'ゆ': ('011002', 10), '窓': ('011011', 10), '動': ('01103', 10), '境': ('011101', 10), 'グ': ('01112', 10), '輪': ('01121', 10), '街': ('012001', 10), '治': ('01202', 10), '都': ('01211', 10), '有': ('0123', 10), '能': ('01301', 10), '信': ('0132', 10), '望': ('0200001', 10), '集': ('020002', 10), '団': ('020011', 10), '便': ('02003', 10), '社': ('020101', 10), '支': ('02012', 10), 'エ': ('02021', 10), '委': ('021001', 10), '時': ('02102', 10), '汽': ('02111', 10), '午': ('0213', 10), '三': ('02201', 10), '等': ('0222', 10), '券': ('0231', 10), '買': ('030001', 10), '厳': ('03002', 10), '心': ('03011', 10), '老': ('0303', 10), 'ブ': ('03101', 10), 'ム': ('0312', 10), 'ホ': ('0321', 10), 'テ': ('10000001', 10), '床': ('1000002', 10), '覆': ('1000011', 10), '十': ('100003', 10), '全': ('1000101', 10), '頭': ('100012', 10), '面': ('100021', 10), '感': ('1001001', 10), 'ピ': ('100102', 10), 'ヴ': ('100111', 10), '進': ('10013', 10), '根': ('100201', 10), '神': ('10022', 10), 'ポ': ('10031', 10), '深': ('1010001', 10), '套': ('101002', 10), '暗': ('101011', 10), '職': ('10103', 10), '直': ('101101', 10), '多': ('10112', 10), '以': ('10121', 10), '頷': ('102001', 10), '参': ('10202', 10), 'キ': ('10211', 10), '尊': ('1023', 10), '区': ('10301', 10), '下': ('1032', 10), 'シ': ('1100001', 10), '楽': ('110002', 10), 'ぬ': ('110011', 10), '度': ('11003', 10), '対': ('110101', 10), '達': ('11012', 10), '何': ('11021', 10), '役': ('111001', 10), '濡': ('11102', 10), '匂': ('11111', 10), '注': ('1113', 10), 'べ': ('11201', 10), '丈': ('1122', 10), '様': ('1131', 10), '積': ('120001', 10), '山': ('12002', 10), '英': ('12011', 10), '国': ('1203', 10), '測': ('12101', 10), '制': ('1212', 10), '作': ('1221', 10), '単': ('13001', 10), '敵': ('1302', 10), '艦': ('1311', 10), '戦': ('133', 10), '六': ('2000001', 10), '紀': ('200002', 10), '攻': ('200011', 10), '迎': ('20003', 10), '緒': ('200101', 10), '歴': ('20012', 10), '史': ('20021', 10), '輝': ('201001', 10), '残': ('20102', 10), '域': ('20111', 10), '沈': ('2013', 10), '泥': ('20201', 10), 'ふ': ('2022', 10), '州': ('2031', 10), '貿': ('210001', 10), '易': ('21002', 10), '探': ('21011', 10), 'へ': ('2103', 10), '縦': ('21101', 10), '縮': ('2112', 10), '変': ('2121', 10), '貌': ('22001', 10), '類': ('2202', 10), '計': ('2211', 10), '塩': ('223', 10), '沢': ('2301', 10), '埋': ('232', 10), '償': ('300001', 10), '真': ('30002', 10), '路': ('30011', 10), '造': ('3003', 10), '低': ('30101', 10), '峡': ('3012', 10), 'プ': ('3021', 10), '産': ('31001', 10), '抵': ('3102', 10), '抗': ('3111', 10), '渡': ('313', 10), '東': ('3201', 10), '潮': ('322', 10), '共': ('331', 10), '忘': ('000000002', 11), '昔': ('00000003', 11), '拘': ('00000012', 11), '束': ('00000102', 11), '断': ('0000013', 11), '切': ('0000022', 11), '暴': ('00001002', 11), '階': ('0000103', 11), '眺': ('0000112', 11), '誰': ('0000202', 11), '再': ('000023', 11), '移': ('0000301', 11), '浸': ('000032', 11), '湖': ('00010002', 11), '界': ('0001003', 11), '幅': ('0001012', 11), 'ザ': ('0001102', 11), '幹': ('000113', 11), '捨': ('000122', 11), '七': ('0002002', 11), '長': ('000203', 11), '鄙': ('000212', 11), '往': ('000302', 11), '選': ('00033', 11), '議': ('00100002', 11), 'ゼ': ('0010003', 11), '体': ('0010012', 11), '表': ('0010102', 11), '正': ('001013', 11), '組': ('001022', 11), '織': ('0011002', 11), '改': ('001103', 11), '善': ('001112', 11), '必': ('001202', 11), '要': ('00123', 11), '設': ('00132', 11), '性': ('0020002', 11), '去': ('002003', 11), '比': ('002012', 11), '粗': ('002102', 11), '末': ('00213', 11), '託': ('00222', 11), '当': ('003002', 11), '珍': ('00303', 11), '消': ('00312', 11), '毎':

('01000002', 11), '激': ('0100003', 11), 'ロ': ('0100012', 11), '旅': ('0100102', 11), '費': ('010013', 11), '節': ('010022', 11), '約': ('0101002', 11), '換': ('010103', 11), '威': ('010112', 11), '苦': ('010202', 11), '労': ('01023', 11), '終': ('01032', 11), '養': ('0110002', 11), '院': ('011003', 11), '配': ('011012', 11), '属': ('011102', 11), 'ズ': ('01113', 11), '族': ('01122', 11), '喜': ('012002', 11), '好': ('01203', 11), '客': ('01212', 11), '余': ('01302', 11), '裕': ('0133', 11), 'ぷ': ('0200002', 11), '藁': ('020003', 11), '突': ('020012', 11), '間': ('020102', 11), '耐': ('02013', 11), '完': ('02022', 11), '駆': ('021002', 11), '角': ('02103', 11), '聳': ('02112', 11), 'セ': ('02202', 11), '嘆': ('0223', 11), '声': ('0232', 11), '抑': ('030002', 11), '篠': ('03003', 11), '追': ('03012', 11), '払': ('03102', 11), '緑': ('0313', 11), '幾': ('0322', 11), 'デ': ('10000002', 11), 'ィ': ('1000003', 11), 'ゴ': ('1000012', 11), '待': ('1000102', 11), '沛': ('100013', 11), '然': ('100022', 11), '砕': ('1001002', 11), '霧': ('100103', 11), '昇': ('100112', 11), '蒸': ('100202', 11), 'ェ': ('10023', 11), '被': ('10032', 11), '台': ('1010002', 11), '紗': ('101003', 11), '幕': ('101012', 11), '郭': ('101102', 11), '浮': ('10113', 11), '想': ('10122', 11), '像': ('102002', 11), '描': ('10203', 11), '姿': ('10212', 11), '遥': ('10302', 11), '秘': ('1033', 11), '的': ('1100002', 11), '荘': ('110003', 11), '門': ('110012', 11), '停': ('110102', 11), '抜': ('11013', 11), '板': ('11022', 11), '御': ('111002', 11), '開': ('11103', 11), '届': ('11112', 11), '帽': ('11202', 11), '子': ('1123', 11), '襟': ('1132', 11), '飛': ('120002', 11), '墓': ('12003', 11), '急': ('12012', 11), '服': ('12102', 11), '撥': ('1213', 11), '扉': ('1222', 11), '戸': ('13002', 11), '革': ('1303', 11), '帳': ('1312', 11), '《': ('2000002', 11), '》': ('200003', 11), '押': ('200012', 11), '雲': ('200102', 11), '閉': ('20013', 11), '薄': ('20022', 11), '歌': ('201002', 11), '席': ('20103', 11), '男': ('20112', 11), '返': ('20202', 11), '領': ('2023', 11), '袖': ('2032', 11), '首': ('210002', 11), '巻': ('21003', 11), '挨': ('21012', 11), '拶': ('21102', 11), '彩': ('2113', 11), '使': ('2122', 11), '同': ('22002', 11), '己': ('2203', 11), '申': ('2212', 11), '聞': ('2302', 11), '別': ('233', 11), '付': ('300002', 11), '泊': ('30003', 11), '業': ('30012', 11), '誇': ('30102', 11), '回': ('3013', 11), '短': ('3022', 11), '敬': ('31002', 11), '調': ('3103', 11), '相': ('3112', 11), '扱': ('3202', 11), '貶': ('323', 11), '疑': ('332', 11), '問': ('000000003', 12), 'ヤ': ('00000013', 12), '礼': ('00000103', 12), '拝': ('0000023', 12), '演': ('00001003', 12), 'ニ': ('0000113', 12), '優': ('0000203', 12), '秀': ('0000302', 12), '売': ('000033', 12), '管': ('00010003', 12), '理': ('0001013', 12), '伝': ('0001103', 12), '受': ('000123', 12), '肩': ('0002003', 12), '態': ('000213', 12), '侮': ('000303', 12), '蔑': ('00100003', 12), '万': ('0010013', 12), '逆': ('0010103', 12), '充': ('001023', 12), '認': ('0011003', 12), '識': ('001113', 12), '恭': ('001203', 12), '揖': ('00133', 12), '丁': ('0020003', 12), '寧': ('002013', 12), '謙': ('002103', 12), '虚': ('00223', 12), '惜': ('003003', 12), '護': ('00313', 12), '務': ('01000003', 12), '料': ('0100013', 12), '機': ('0100103', 12), '嫌': ('010023', 12), '徴': ('0101003', 12), '摘': ('010113', 12), '折': ('010203', 12), '快': ('01033', 12), '褒': ('0110003', 12), '逃': ('011013', 12), '観': ('011103', 12), '念': ('01123', 12), '傘': ('012003', 12), '独': ('01213', 12), '雰': ('01303', 12), '囲': ('0200003', 12), '醸': ('020013', 12), '冷': ('020103', 12), '漏': ('02023', 12), '雄': ('021003', 12), '弁': ('02113', 12), '案': ('02203', 12), '千': ('0233', 12), '百': ('030003', 12), '五': ('03013', 12), '経': ('03103', 12), '験': ('0323', 12), '脇': ('10000003', 12), '尋': ('1000013', 12), '導': ('1000103', 12), '夫': ('100023', 12), '答': ('1001003', 12), '眉': ('100113', 12), '毛': ('100203', 12), '崇': ('10033', 12), '高': ('1010003', 12), '簡': ('101013', 12), 'ケ': ('101103', 12), '吟': ('10123', 12), '値': ('102003', 12), '軽': ('10213', 12), '岩': ('10303', 12), '頑': ('1100003', 12), '架': ('110013', 12), '構': ('110103', 12), '莫': ('11023', 12), '抱': ('111003', 12), '亀': ('11113', 12), '裂': ('11203', 12), '煉': ('1133', 12), '瓦': ('120003', 12), '詰': ('12013', 12), '跡': ('12103', 12), '筋': ('1223', 12), '枠': ('13003', 12), '割': ('1313', 12), '稲': ('2000003', 12), '妻': ('200013', 12), '刻': ('200103', 12), '印': ('20023', 12), '諺': ('201003', 12), '半': ('20113', 12), '円': ('20203', 12), '形': ('2033', 12), '背':

('210003', 12), '載': ('21013', 12), ' ！ ': ('21103', 12), '胆': ('2123', 12), '連': ('22003', 12), '応': ('2213', 12), 'ぜ': ('2303', 12), '定': ('300003', 12), '関': ('30013', 12), '天': ('30103', 12), '登': ('3023', 12), 'ギ': ('31003', 12), 'ペ': ('3113', 12), '拭': ('3203', 12)}

Kodierte Textlänge: 42224

long_message_a

{' ': ('0', 1), 'e': ('1', 1), 'i': ('2', 1), 't': ('30', 2), 'n': ('31', 2), 's': ('32', 2), 'r': ('33', 2), 'a': ('34', 2), 'h': ('35', 2), 'd': ('36', 2), 'm': ('37', 2), 'c': ('38', 2), 'o': ('39', 2), 'w': ('3A', 2), 'I': ('3B', 2), 'l': ('3C', 2), 'k': ('3D', 2), 'ü': ('3E', 2), 'D': ('3F', 2), 'B': ('3G', 2), 'W': ('3H', 2), 'N': ('3I', 2), 'F': ('3J', 2), 'T': ('3K', 2), '.': ('3L', 2), 'z': ('3M', 2), '-': ('3N', 2), 'ä': ('3O', 2), 'g': ('3P', 2), ',': ('3Q', 2), 'J': ('3R', 2), 'R': ('3S', 2), 'u': ('3T', 2), ';': ('3U', 2), 'v': ('3V', 2), '?': ('3W', 2), 'f': ('3X', 2), '<': ('3Y', 2), '3': ('3Z', 2)}

Kodierte Textlänge: 300

long_message_b

{' ': ('1', 1), 'e': ('2', 1), 'i': ('3', 1), 't': ('4', 1), 'n': ('5', 1), 's': ('6', 1), 'r': ('7', 1), 'a': ('8', 1), 'h': ('9', 1), 'd': ('A', 1), 'm': ('B', 1), 'c': ('00', 2), 'o': ('01', 2), 'w': ('02', 2), 'I': ('03', 2), 'l': ('04', 2), 'k': ('05', 2), 'ü': ('06', 2), 'D': ('07', 2), 'B': ('08', 2), 'W': ('09', 2), 'N': ('0A', 2), 'F': ('0B', 2), 'T': ('C', 2), '.': ('D', 2), 'z': ('E', 2), '-': ('F', 2), 'ä': ('0C', 3), 'g': ('0D', 3), ',': ('0E', 3), 'J': ('0F', 3), 'R': ('G', 3), 'u': ('H', 3), ';': ('I', 3), 'v': ('J', 3), '?': ('0G', 4), 'f': ('0H', 4), '<': ('0I', 4), '3': ('0J', 4)}

Kodierte Textlänge: 252

short_message

{'a': ('0', 1)}

Kodierte Textlänge: 1

single

Es muss mindestens 2 Perlen geben!