

SWE3 Übung 8

Name: Christoph Gassner (S2110458010)
Zeitaufwand: 15-20h
Datum: 16.02.2022
Punkte:
Anmerkung: Diese Übung ist mein „Joker“ = verspätet Abgabe

Lösungsidee:

Die Klasse `text_coder` ist zuständig für die Kodierung und Dekodierung eines Textes. In der Klasse werden verschiedene Informationen, wie z.B. die Kompressionsrate, gespeichert. Im Konstruktor wird der Dateiname als String übergeben, der Text aus der übergebenen Datei eingelesen und gespeichert. Die Dekodierung des Textes erfolgt über die Methode `decode()`, die Kodierung über die Methode `encode()` diese delegieren an die Klasse `coding_token`. Mit den Methoden `get_compression_rate()`, `get_original_length()` und `get_encoded_length()` können Informationen über die Größe des Textes ausgelesen werden. `get_text()`, `get_encoded_text()` und `get_decoded_text()` geben Pointer auf die jeweiligen Texte zurück.

Die `coding_token` Klasse erstellt ein `coding_table` Objekt, das zur Kodierung und Dekodierung eines Textes verwendet wird. Über die Methoden `encode()` und `decode()` werden Texte entsprechend kodiert bzw. dekodiert unter zuhilfenahme des Trees, welcher in der Liste als einziges Element gespeichert ist.

Die `coding_table` Klasse speichert das `huffman_tree` Objekt. Es kann über die Methoden `get_code_for_char()` und `get_char_for_code()` ein entsprechendes `bit_code` Objekt oder der dazugehörige Key abgerufen werden. Die Methode `print()` gibt eine Tabelle aller Einträge auf dem Bildschirm aus.

Die `frequency_table` Klasse speichert die Vorkommen der chars in %, welche über die Methode `generate()` aus einem übergebenen Text erstellt wird. Über die Methode `get_frequency()` kann auf die Häufigkeit des Vorkommens eines Zeichens zugegriffen werden. Die Methode `print()` gibt eine Tabelle aller Einträge auf dem Bildschirm aus.

Die Klasse `huffman_list` repräsentiert eine Liste von Knoten, die für die Erstellung eines Huffman-Baums verwendet wird. Jeder Knoten hat einen Buchstabenwert und eine Frequenz, die die Häufigkeit des Buchstabens im Text angibt. Die Klasse bietet Methoden zum Hinzufügen von Knoten zur Liste, zum Sortieren der Liste nach Frequenz und zum Erstellen eines Huffman-Baums aus der sortierten Liste.

Die Klasse `huffman_tree` repräsentiert den Huffman-Baum. Der Baum besteht aus Knoten, die entweder Blätter oder innere Knoten sein können. Ein Blattknoten enthält einen Buchstabenwert, während ein innerer Knoten zwei untergeordnete Knoten hat und eine Summe ihrer Häufigkeiten als Frequenzwert hat. Die Klasse bietet Methoden zum Erstellen des Baums aus einer sortierten Liste von Knoten, zum Codieren und Decodieren von Text basierend auf dem Baum, zum Durchlaufen des Baums zur Generierung von Codewörtern für jeden Buchstaben und zum Drucken des Baums zur Überprüfung.

Source Code: siehe VS-Solution

Testfälle:

Leider, sprengt der gesamte Output, mancher Testfälle den rahmen, weshalb ich nur screenshots der relevanten Passagen einfüge.

Testfall 1: Testet die Frequency-Table und ihrer Methoden.

```
Testing Frequency Table:  
Generating table..  
Printing table..  
-----  
Frequency Table  
-----  
|l| 0.315789|  
|H| 0.157895|  
|a| 0.157895|  
|o| 0.157895|  
|.| 0.105263|  
|,| 0.105263|  
-----  
Test size of table..  
get_size: 6  
Test get frequency ...  
get_frequency: 0.157895 should be 0.157895  
Frequency Table Exception: 'G' not found in table.  
Program ended with exit code: 0
```

Testfall2: Gesamte Funktionalitäten des Moduls mit dem text „LoremIpsum“

```
Encoded text length: 32bits  
Compression rate: 60%  
-----  
Frequency Table  
-----  
|m| 0.2|  
|I| 0.1|  
|L| 0.1|  
|e| 0.1|  
|o| 0.1|  
|p| 0.1|  
|r| 0.1|  
|s| 0.1|  
|u| 0.1|  
-----  
Coding Table  
-----  
|I| 1100|  
|L| 1101|  
|e| 1110|  
|m| 10|  
|o| 1111|  
|p| 000|  
|r| 001|  
|s| 010|  
|u| 011|  
-----  
Text we got: LoremIpsum  
Encoded Text: 11011111001111010110000001001110  
Decoded Text: LoremIpsum  
Is Good?: true  
Original text length: 80bits  
Encoded text length: 32bits  
Compression rate: 60%  
Frequency Table
```

Testfall 3: Gesamte Funktionalität mit Beispieltext aus file: test3.txt

Frequency Table		Coding Table	
	0.167751	U	0.000723066
e	0.0889371	W	0.000723066
n	0.076645	q	0.000723066
i	0.0715835	x	0.000723066
a	0.0667375		
t	0.0592914		
o	0.0585683		
s	0.0542299		
r	0.0477223		
l	0.0368764		
d	0.0347872		
c	0.0310918		
f	0.0231381		
u	0.0231381		
g	0.021692		
h	0.0202458		
y	0.0180766		
p	0.0159874		
m	0.0137383		
w	0.0130152		
.	0.0115691		

Testfall 4: Gesamte Funktionalität mit Proteinsequenzen aus file test4.txt

Testfall 5: Gesamte Funktionalität mit DNA Sequenzen aus file test5.txt

CAATGGCCGAGCTAAT		0110011001101000011010110
GTAGAAATAATTAGGTGAAT	I	0..000194818
TATCTCGCATCTTGTCTC	f	0..000194818
Is Good?	-----	
Original text length:		
Encoded text length:		
Compression rate:		
Coding Table		
Frequency Table		
A		1100
T	(11111111
G)	11111110
C	..	11110100
.	0	11001100
.	1	11001101
s	3	1110111100
g	4	1110111101
e	5	1110111110
n	6	11100100
t	7	1110011111
a	8	111000100
i	9	111000101
u	;	111000110
(>	111000111
)	A	10
,	C	1101
H	F	11001110
d	G	00
h	H	11100101
c	J	11100101
l	T	01
o	X	11100110
p	Y	111001111
6	Z	11100011
J	a	110000
	b	11100000
	c	11101100

Testfall 6: Leeres File

```
Coder Exception: empty.txt is empty file..  
Program ended with exit code: 0
```

Testfall 7: nichtexistierendes File

```
Coder Exception: unable to open file! (not_there.txt)
Program ended with exit code: 0
```

Testfall 8: kein txt file text_coder.cpp