Übung 08

6h 35min

Beispiel

Lösungsidee

Huffman-Codierung ist ein Ansatz zur Komprimierung und spart Speicherplatz indem häufige Zeichen durch weniger Bit kodiert, werden als seltene.

Da wir in den Übungen aber nichts zu Bitstreams gemacht haben, werden einfach Nuller bzw. Einser als Zeichen in ein File geschrieben, so kann man zusätzlich die Codierung besser ansehen und es ist einfacher sie wieder einzulesen.

TextCoder

Stellt nach außen die beiden Methoden *encode* und *decode* zur Verfügung.

void encode(string const& from, string const& to)
Liest und setzt über die *read_original*-Methode den Text des Files 'from' ein. Danach wird mit Hilfe der Klasse CodingToken ein CodingTable-Objekt erstellt. Nun wird für jedes Zeichen des Originaltextes der passende Code aus dem CodingTable-Objekt gesucht und dieser in einem String gespeichert. Danach wird der nun codierte Text ausgegeben und in das übergebene to-File gespeichert

void decode(string const& from, string const& to)
Der als Text gespeicherte Code wird eingelesen und es wird überprüft ob der Zeiger zum CodingTable-Objekt kein Nullpointer ist. Ohne einen CodingTable ist die Dekodierung nämlich nicht möglich.
Ist ein CodingTable vorhanden wird überprüft, ob die Zeichen des eingelesenen Codes dort gespeichert sind. Wenn nicht wird, dass nächste Zeichen zum aktuell temporär gespeicherten Code hinzugefügt und es wird überprüft, ob es sich nun um einen gültigen Code handelt. Dass wird so lange fortgeführt, bis man am Ende des eingelesenen Texts ankommt. Handelt es sich um einen Code wird dass Zeichen für welches er steht zu Ergebnis-String hinzugefügt und es wird mit dem nächsten Zeichen als neuer aktueller Code begonnen.
Danach werden sowohl der Originaltext als auch der dekodierte Text ausgegeben und miteinander verglichen. Der dekodierte Text wird dann in der übergebene ‚to'-File gespeichert.

CodingToken

Stellt nach außen die Methode generateCodingTable und set_Text zur Verfügung und enthält eine HuffmanListe, eine Map für die Häufigkeiten der Zeichen und den Text für welchen der CodingTable generiert werden soll.

CodingTable* generateCodingTable()
Füllt zuerst die Häufigkeits-Map mit der Hilfsmethode fill_frequency_map und printed diese auf die Konsole. Danach erzeugt die Hilfmethode create_hl eine neue HuffmanListe.
So lange die Anzahl an Einträge in dieser List größer als 1 ist, werden die ersten beiden Elemente (die mit den niedrigsten Prozentanteilen) entfernt und Bäume bzw. die Wurzeln der Bäume, die in diesen Elementen gespeichert werden, werden zu einem neuen Baum zusammen gefügt und mit einem neuen Knoten erneut sortiert in die List eingefügt.
Ist nur noch ein Listenelement übrig so handelt es sich bei dem Baum in diesem Element um den fertigen Huffman-Baum für welchen ein Objekt der Klasse CodingTable die Codes für die einzelnen Zeichen findet und ausgibt. Dieser CodingTable wird dann zurückgegeben und später in der Klasse TextCoder verwendet.

CodingTable

Stellt nach außen die Methoden find_codes, get_code, get_sign, print_codes und get_entries zur Verfügung. Wobei get_entries nur der Getter für die private Entry-Map ist.

void find_codes(HTNode const* root)
Überprüft ob der übergebene HuffmanTree-Knoten kein Nullpointer ist und holt sich, falls dies nicht der Fall ist, dessen Kinder. Handelt es sich dabei um Nullpointer gibt es im gesamten Baum nur ein Element und dieses bekommt den Code 1.
Sind die Kinder keine Nullpointer werden mit ihnen jeweils die Methode find_codes_rec aufgerufen. Für den linken Kindknoten wird der Code 0 übergeben, für den rechten 1.

void CodingTable::find_codes_rec(HTNode const* node, string code)
Überprüft ob es sich bei dem übergebenen Knoten um ein Blatt handelt (dies ist der Fall wenn seine beiden Kindknoten Nullpointer sind) und fügt den Knoten dann gegebenen Falls in die Entry-Map hinzu mit dem übergebenen Code als Value.
Handelt es sich nicht um ein Blatt wird find_codes_rec erneut mit dem linken bzw. rechten Kindknoten aufgerufen, wobei der Code für den linken Knoten um eine 0 am Ende ergänzt wird, der für den rechten wird um eine 1 erweitert.

string const& get_code(char const c)
Sucht in der Entry-Map nach Einträgen mit c als Schlüssel und gibt gegebenen Falls den Wert dieses Eintrags (also den Code für das Zeichen c) zurück oder einen leeren String falls es zu dem Zeichen keinen Eintrag gibt.

char const& get_sign(string const& code) const
Sucht in der Entry-Map nach Einträgen die ‘code’ als Code hinterlegt haben, wird keiner gefunden wird -1 retourniert, ansonsten der Schlüssel des gesuchten Eintrags (also das Zeichen das durch ‚code‘ kodiert wird).

void print_codes() const
Schreibt jeden Eintrag der Entry-Map auf die Konsole

HuffmanList

Enthält einen Pointer auf einen HuffmanListNode, sowie die aktuelle Größe der Liste. Die Methode add fügt einen neuen Knoten sortiert nach dessen Wert in die Liste ein. Die Methode pop_front entfernt und retourniert den Wurzelknoten.

HLNode

Steht für HuffmanListNode und enthält einen Pointer auf das nächste Listenelement sowie einen Pointer auf einen HuffmanTree. Für Beide gibt es Setter und Getter.

HuffmanTree

Enthält einen Pointer auf einen HuffmanTreeNode. Für welchen es einen Setter und einen Getter gibt. Die Methode print ruf die rekursive Hilfsmethode print_rec auf und gibt so den gesamten Baum mit passenden Einrückungen aus. Die Methode get_value retourniert den Wert des Wurzelknotens.

HTNode

Steht für HuffmanTreeNode und enthält Point auf sein linkes bzw. rechtes Kind, sowie das Zeichen für welches dieser Knoten steht und den Wert. Für alle vier Datenkomponenten gibt es Getter, Setter gibt es nur für die beiden Kindknoten.
Die Methode calc_value berechnet den Wert eines Knotens, indem er die calc_value-Methoden der Kinder aufruft und die Werte zusammenzählt. Hat ein Knoten keine Kinder handelt es sich um ein Blatt und es wird einfach der Wert der in ‚value‘ gespeichert wird zurückgegeben.

Quelltext

Siehe SWE3_Waser_UE08

Testfälle

```
Test with empty file:
    c|    p
------------

Huffman tree:
can't decode without Coding Table!

Test with non existant file:
    c|    p
------------

Huffman tree:
can't decode without Coding Table!

Test decoding without coding table (should not work):
can't decode without Coding Table!
```

Ungültige Testfälle, welche den Text nicht kodieren und/oder beim Anschließenden decodieren scheitern sollen.

Test mit dem Beispiel aus der Angabe:

```
Test decoding without coding table (should not work):
can't decode without Coding Table!

Test with example from instructions:
    c|    p
------------
  "A"| 0.33
  "C"| 0.32
  "D"| 0.12
  "N"| 0.08
  "R"| 0.15

Huffman tree:
"-" 1
        "-" 0.35
                "R" 0.15
                "-" 0.2
                        "N" 0.08
                        "D" 0.12
        "-" 0.65
                "C" 0.32
                "A" 0.33
    c|    p
------------
  "A"|   11
  "C"|   10
  "D"|  011
  "N"|  010
  "R"|   00

Compression rate of 27.5%

Encoded:
11111010100100100110110111111101010101000001111101010100100100101101101101101110101011111110100101
11111111101001101101101100001111111101010101000001111111110000001010101011110000111111111111111010100
0000000010101010

Decoded:
AACCCNNDDDAACCCCCRRAACCCCNNNADDDDDCCCAAACCNAAAAACCDDDDRRAAAANCCCRRAAAAARRRCCCCAARRAAAAAAACCCRRRRNCCC
Original:
AACCCNNDDDAACCCCCRRAACCCCNNNADDDDDCCCAAACCNAAAAACCDDDDRRAAAANCCCRRAAAAARRRCCCCAARRAAAAAAACCCRRRRNCCC
The decoded version it is the same as the original!
```

Test mit Textstelle aus der Erklärung (da sehr viele verschiedene Zeichen vorkommen nur Ausschnitte der Ausgabe): Beginnt mit einem Teil der Table der Wahrscheinlichkeiten

```
 "a"|0.0418919
 "b"|0.0148649
 "c"|0.00945946
 "d"|0.0337838
 "e"|0.143243
 "f"|0.0256757
 "g"|0.0189189
 "h"|0.0202703
 "i"|0.0594595
 "k"|0.00945946
 "l"|0.0378378
 "m"|0.0243243
 "n"|0.0756757
 "o"|0.0216216
 "p"|0.00540541
 "r"|0.0594595
 "s"|0.0621622
 "t"|0.0527027
 "u"|0.0364865
 "v"|0.00675676
 "w"|0.00945946
 "x"|0.0027027
 "z"|0.00405405

Huffman tree:
"-" 1
        "-" 0.427027
                "-" 0.187838
                        "-" 0.0851351
                                "a" 0.0418919
                                "-" 0.0432432
                                        "-" 0.0216216
                                                "-" 0.0108108
                                                        "D" 0.00540541
                                                        "A" 0.00540541
                                                "-" 0.0108108
```

Ausschnitt der Codierungstabelle, man sieht, dass „e“ sehr oft vorkommt und deshalb nur 3 Bit bräuchte:

```
 "d"|10100
 "e"|  110
 "f"|00101
 "g"|101111
 "h"|111110
 "i"| 0110
 "k"|1011101
 "l"|11110
 "m"|00100
 "n"| 1110
 "o"|00011
 "p"|0001011
 "r"| 0101
 "s"| 0111
 "t"| 0011
 "u"|10110
 "v"|0100010
 "w"|1011100
 "x"|00010100
 "z"|10101010

Compression rate of 56.25%

Encoded:
11111101011111001011000100001100100
00101100111011110000001011000101100
10011010100001011101001101100001111
```

Beim Kodieren das Beispiel aus der Angabe verwendet, beim Dekodieren dass kodierte File des Textes aus der Erklärung:

```
Decoded:
AAANAACNARNRDRCRNNRNNRANCCANCRACAANCRRNRNNRNNARACAARRRCARNARCCDRCARRDNNARCCACDADCCRRCNCNDCDNCRNACCDDRRAAC
CDNADNACCDCARCCADRCCDCRAACDDCAAARNRCRRCCCNAARNNARACDADACRACRRNARAARCCDDCCRCDAACRRDCAACACCRDARRCCRDAAAANAR
DNACANNADCCCNDACCARNCNCNRRDCAAAAACDRACDCCAARRNRDDNRCDAACRRDCAACCRDARRCCRDAAAANARRACCDRCARNNDDRDARDCRRCCAC
RACAACAANADCCAADCNDRDAACRDCCDDCCRRDCCNADCCCNDRCAACANCRANAARACRADACNACCRRCDDRNADARRNNANRNRARDCNCCRDDARNACC
RADCCAAAAACDRDCACACACCCAARRNRDDNRCDAACRRDCAACCDACCARNCNCNRRDCAAACDNACANRCACRNANCRDRAACACRADDRCCDCDRADCADC
ADCCCNNACRARCANNRADACNACNCAACDRDDARNRCRDACCNCDRCARNDAACRRDCAACCDACCARNCNCNRRDCAAAAACDRACDCRCRCARCADCDDCCC
ADNNCRADNNACDRARDARCACCRCDRDNADNACCDCARCARAAANAACNACDARCCCANCRACAANCRRNRNRAARNRRDACAARCNDCDNRAARRRCCADACR
RNARNARARADCANRACRNARCNARCACCRCRDANACACACCADCCAACNCDRCARDDCCCRARCRCDNCNADDADADDCCDAARCRCDDAADCCCDDCAAARAR
RDCAACACCCAARRNCACNADCDDCCCDDCCRCDRADNNCRRRAARCCCANCADCCCCCDRRCACNCDRCARRNRRRRAADCCACCRRRCADACAANADCCAACR
RANCADACNRAARDRNRRCAAADRNDDCRADDRCACACCRCRRNDRADNNACAACARADCANNCCRDDARNCAADCAACRRDCAACACCNACRADNNCCCCCAAR
ACRRRCNCAARNDRADDACANRRCNCNNNDDANCNADCCAACRCRDCCRNADDRCARARADCADCCRRCRARACANAAACNRCRDNCAANADACAARRNAAAAAR
Original:
AACCCNNDDDAACCCCCRRAACCCCNNNADDDDDCCCAAACCNAAAAACCDDDDRRAAAANCCCRRAAAAAARRRCCCCAARRAAAAAAACCCRRRRNCCC
The decoded version it is not the same as the original!
```