DAA Assignment: Compressing Data using Huffman Trees

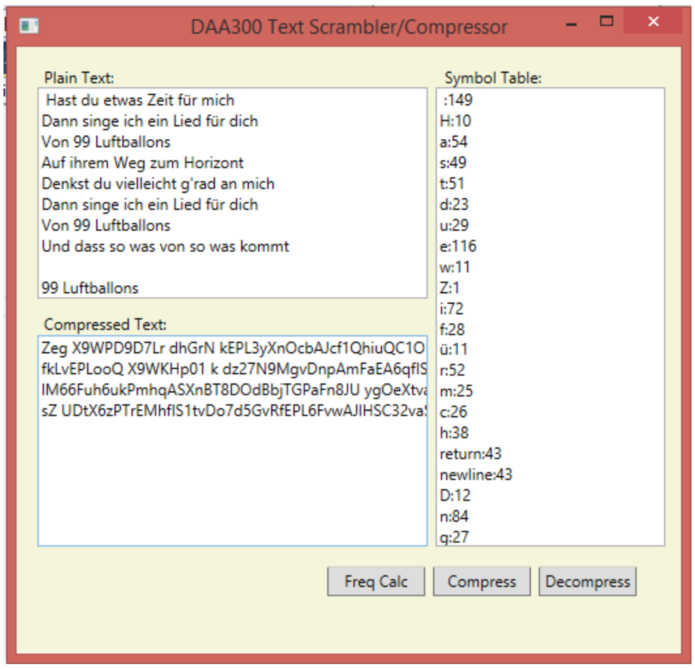
Jason Giancono (16065985)

May 22, 2014

Abstract

For the assignment I implemented a Huffman Coding compression algorithm in C#. The application is working and has no major bugs.

Contents



Plain Text:

```
 Hast du etwas Zeit für mich
Dann singe ich ein Lied für dich
Von 99 Luftballons
Auf ihrem Weg zum Horizont
Denkst du vielleicht g'rad an mich
Dann singe ich ein Lied für dich
Von 99 Luftballons
Und dass so was von so was kommt

99 Luftballons
```

Compressed Text:

```
Zeg X9WPD9D7Lr dhGrN kEPL3yXnOcbAJcf1QhiuQC1O
fkLvEPLooQ X9WKHp01 k dz27N9MgvDnpAmFaEA6qflS
IM66Fuh6ukPmhqASXnBT8DOdBbjTGPaFn8JU ygOeXtva
sZ UDtX6zPTrEMhflS1tvDo7d5GvRfEPL6FvwAJIHSC32va
```

Symbol Table:

```
 :149
H:10
a:54
s:49
t:51
d:23
u:29
e:116
w:11
Z:1
i:72
f:28
ü:11
r:52
m:25
c:26
h:38
return:43
newline:43
D:12
n:84
g:27
```

Figure 1: Huffman Coding can reduce the amount of space data occupies.

1 Introduction

Huffman Coding allows for symbols or sequences of symbols in a data file which are frequent to be represented by smaller symbols in order to reduce the overall file size of the program. This method is only effective when there are disparities between the frequencies of symbols. If all symbols appear the same amount of times there will be no efficiency in compression, assuming there is no redundant data in the representation of the data. Huffman coding is used in many compression algorithms today and this application demonstrates how it can reduce file size visually using text representation.

2 Functionality

2.1 Huffman Tree

The Huffman Tree class is an abstract class with leaf and composite children. This allows them to be treated equally when being sorted in the heap but also allows intuitive recursive encoding and decoding by the use of polymorphism.

2.2 Priority Queue (Heap)

The priority queue I implemented was a heap. It has no default constructor because all the nodes will be generated at it's creation so I just have the Build Max Heap constructor which uses an existing array. The heap is specific to the HuffmanTreeNode class which I did to be able to use the frequency without passing it as a separate argument and having a container class for the heap nodes.

2.3 Frequency Table Generation

My pplication generates the frequency table by iterating through each character in the text field, checking if they are already in the frequency dictionary and if not, iterating through the entire text field incrementing a count by one every time there is a match. It only calculates the frequency of single characters.

2.4 Compression

The compression button generates the Huffman tree from the frequency table and outputs the text into the compressed text box after compressing it. This works fine, the text gets longer if you mess with the optimal frequency table. If the table is incomplete then a message box is shown with an exception.

2.4.1 Compressed Text Display

The text is displayed in the text box after the compress button is pressed.

2.5 Decompression

Decompression works when the same frequency table is used as compression, otherwise it usually won't be able to decompress the text. There may be extra characters on the end of the plain text due to the padding of the compressed text so it can be displayed.

2.6 Multiple String Symbols

This was not implemented because it was worth 0 marks and I'm lazy.

2.7 Exception Handling

Any exceptions which are thrown are displayed to the user as a message box then the program will keep running.

3 Defects

3.1 Decompression Artefacts

As mentioned in the specification, there are extra symbols when decompressing due to the bit padding during compression. This is a feature, not a bug!

3.2 Somewhat vague exceptions

I haven't defined my own exceptions, so sometimes the exceptions the system throws are a bit non-specific. For instance if you have an empty symbol table it says "Object reference not set to an instance of an object" which is not helpful. Or if there is erroneous input in the symbol table it says "An item with the same key has already been added". This is unfortunate but making the system detect the different types of errors would have been too much work.

3.3 Newline and Return frequencies

Due to parsing issues, instead of displaying the \n (newline) and \r (return) characters in the frequency box, it substitues them with the words newline and return and switches them back when parsing. Also windows has newlines as \r\n, which means each newline is actually two characters. This means there is some redundancy in the table, however that is 100% the fault of Windows.

4 How to run

The program was developed with Microsoft Visual Studio 2013 Professional^{TM}. To compile and run just boot it up and press play.

5 Sample Output

5.1 Sample 1

5.1.1 Plain Text

```
The Quick Brown Fox Jumped Over The Lazy Dog.
```

5.1.2 Frequency Table

```
T:2
h:2
e:4
 :8
Q:1
u:2
i:1
c:1
k:1
B:1
r:2
o:3
w:1
n:1
F:1
x:1
J:1
m:1
p:1
d:1
O:1
v:1
L:1
a:1
z:1
y:1
D:1
g:1
.:1
```

5.1.3 Compressed Text

```
nNJ
JSRJvlUhXL3fbhgqG
rHzGF33xYAY7f
```

5.2 Sample 2

5.2.1 Plain Text

```
All work and no play makes Jack a dull boy.
```

5.2.2 Frequency Table

```
A:1
l:5
 :9
w:1
o:3
r:1
k:3
a:5
n:2
d:2
p:1
y:2
m:1
e:1
s:1
J:1
c:1
u:1
b:1
.:1
```

5.2.3 Compressed Text

```
tk1NbaGVL7ALmIvJIkLNM8AIYYjA
```