جامعة
الملك سعود
King Saud University

Design and Analysis of Algorithms

CSC311

Huffman coding and decoding

(Project report)

Prepared by:

- Abdulaziz Alresaini –439102791.
- Nawaf Mustafa dallah–441106022.

1. Introduction

1.1 Purpose

The goal of this project is to build a program that compresses text without losing any data using Huffman trees/Huffman coding, a technique created by David Huffman.

1.2 Problem statement

Characters in the text currently take 8 bits or 1 byte to be represented on computers and compressing text is a challenge because it needs to be compressed to less than 8 bits without any loss of data.

1.3 Main data structures

The main data structures used are: Linked list , Priority queue, Hash Map.

2. Experiments

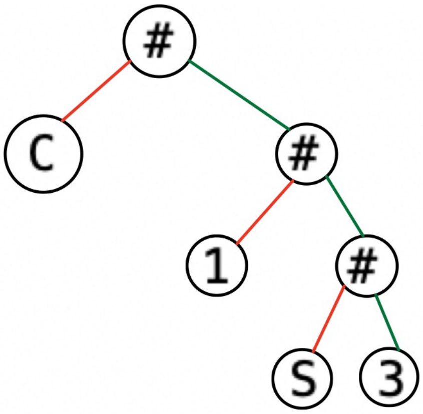
Below are 4 experiments of increasing text (input) size and the resulting output of the program and Huffman tree with average running time when running the experiment 3 times.

Experiment 1: input size: 6 characters

File input = “CSC311”

The output:

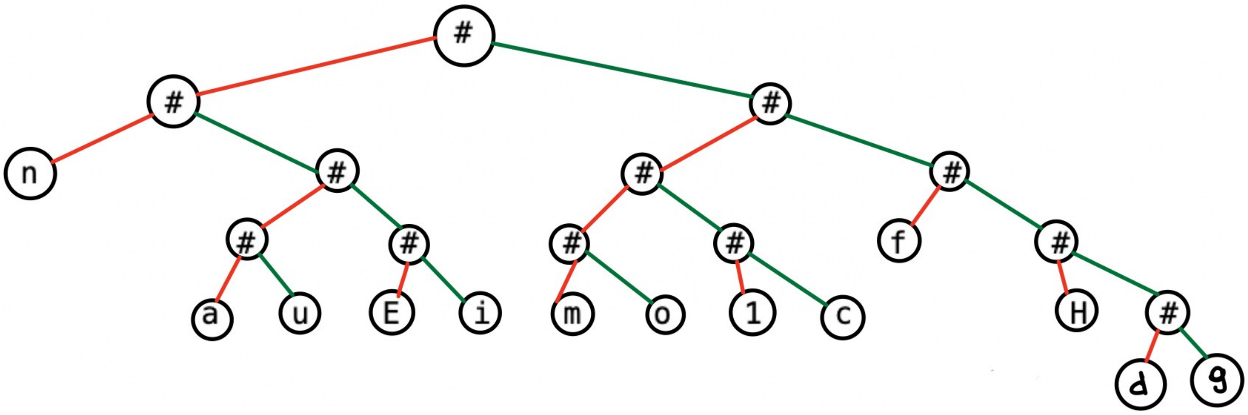
```
Char| Huffman code
-------------------
C   |  0
1   |  10
S   |  110
3   |  111


Encoded text: 011001111010


Decoded text: CSC311

Size before compression: 48 bits
Size after Compression: 12 bits
Data compression rate: 75.00%

The average time= 5.66 ms .
```



Experiment 2: input size 16 characters

File input = “HuffmanEncoding1”.
The output:

```
Char| Huffman code
------------------
n   |  00
a   |  0100
u   |  0101
E   |  0110
i   |  0111
m   |  1000
o   |  1001
1   |  1010
c   |  1011
f   |  110
H   |  1110
d   |  11110
g   |  11111


Encoded text: 1110010111011010000100000110001011100111110011100111111010


Decoded text: HuffmanEncoding1

Size before compression: 128 bits
Size after Compression: 58 bits
Data compression rate: 54.69%

The average time= 13 ms.
```

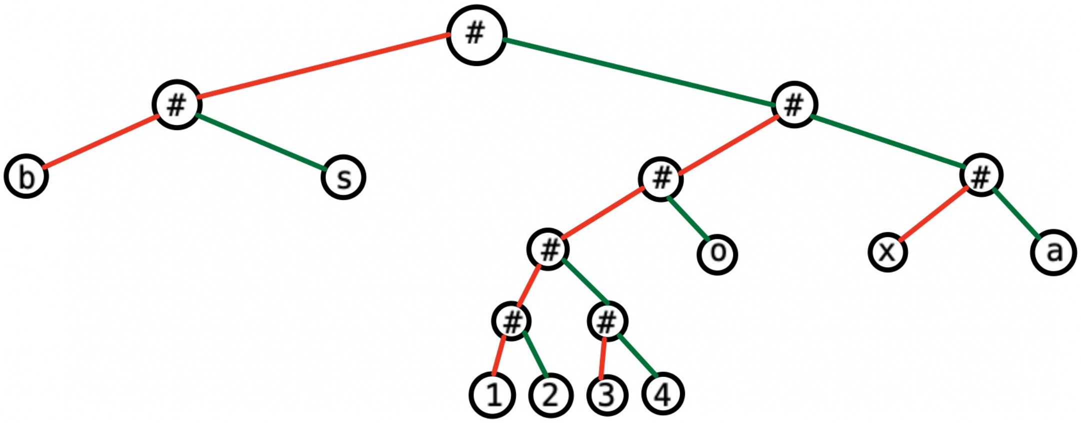


Experiment 3: input size 50 characters

File input = “aaaaaaaaaa1bbbbbbbbbb2oooooooo3ssssssssss4xxxxxxxx”.

The output:

```
Char| Huffman code
------------------
b   |  00
s   |  01
1   |  10000
2   |  10001
3   |  10010
4   |  10011
o   |  101
x   |  110
a   |  111


Encoded text:
11111111111111111111111111111110000000000000000000000000100011011011011010110
1101101101100100101010101010101010101100111101101101101101101101101101110


Decoded text: aaaaaaaaaa1bbbbbbbbbb2oooooooo3ssssssssss4xxxxxxxx

Size before compression: 400 bits
Size after Compression: 138 bits
Data compression rate: 65.50%

The average time= 12.66 ms.
```



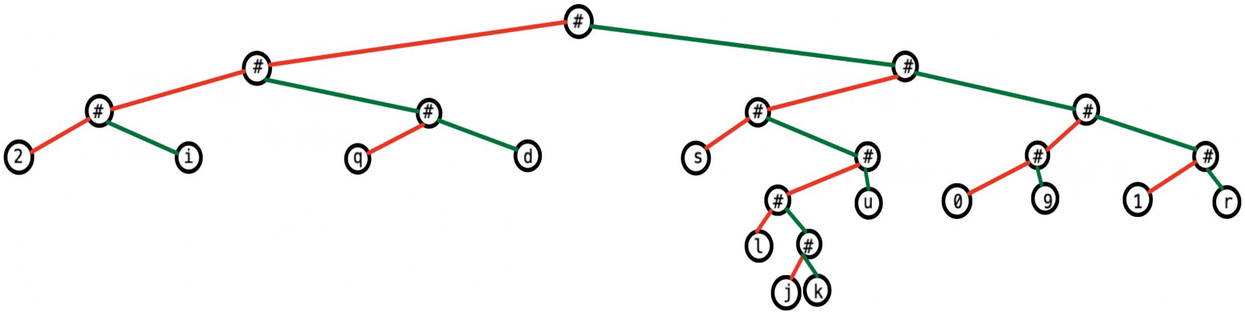
Experiment 4: input size 100 characters

File input = “ssssssssss1111111111dddddddddd2222222222rrrrrrrrrrqqqqqqqqqq000000000iiiiilkjiiiii99999
99999uuuuuuuu”

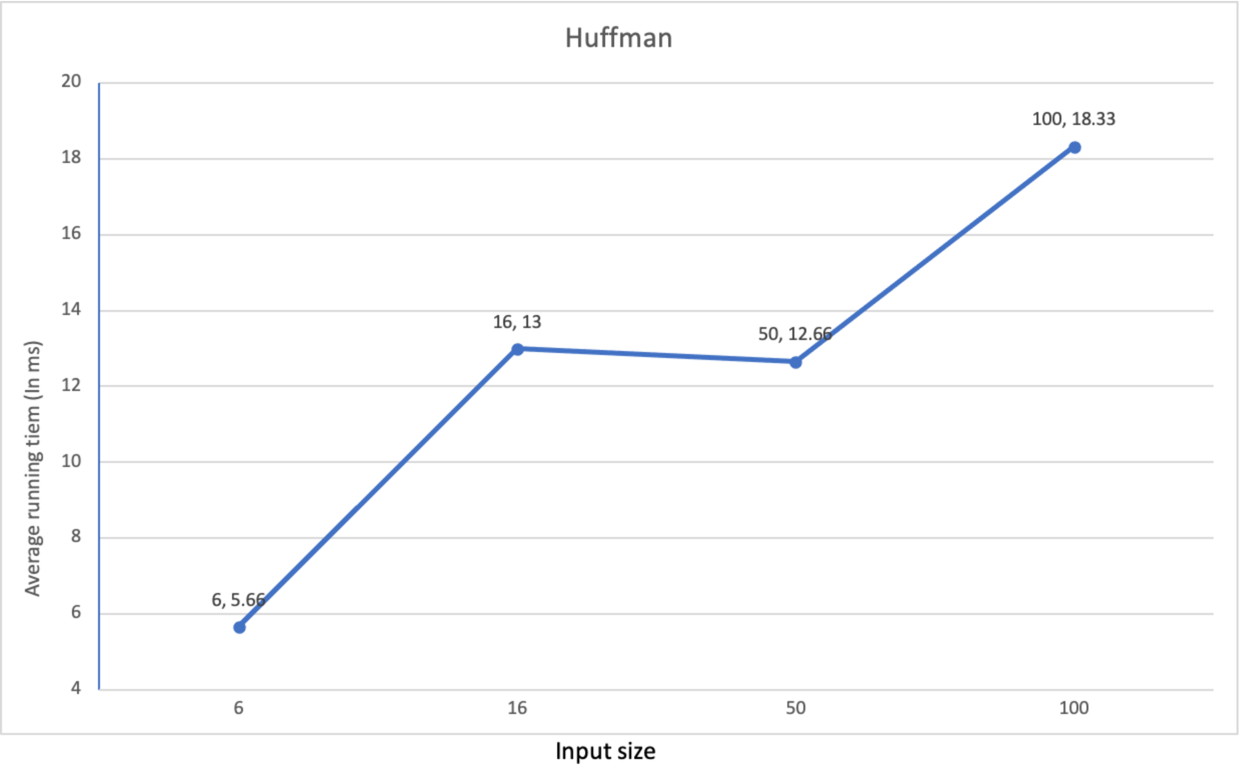
The output:

```
Char| Huffman code
------------------
2    |  000
i    |  001
q    |  010
d    |  011
s    |  100
l    |  10100
j    |  101010
k    |  101011
u    |  1011
0    |  1100
9    |  1101
1    |  1110
r    |  1111

Encoded text:
10010010010010010010010010010011101110111011101110111011101110111011100110
11011011011011011011011011000000000000000000000000000000111111111111111111
11111111111111111111110100100100100100100100100100101100110011001100110011
00110011001100010010010010011010010101110101000100100100100111011101110111
10111011101110111011101110111011101110111011101110111011

Decoded text:
ssssssssss1111111111dddddddddd2222222222rrrrrrrrrrqqqqqqqqqq000000000iiiii
lkjiiiii9999999999uuuuuuuu

Size before compression: 800 bits
Size after Compression: 355 bits
Data compression rate: 55.63%
```

The average time= 18.33 ms.



Average running time versus input size:



3. Conclusion

Huffman coding compresses text without loss and the smaller the text/input size the more effective the compression is, as we see in the experiments, the bigger the text/input size, the lesser the compression rate becomes, also the average time between 3 experiments was taken as each run varies in time as it depends on the CPU's speed and the CPU's task scheduler, and the Huffman coding algorithm has a time complexity of O(nlogn).