COMP 261 Assignment 5

Question 1: Write a short summary of the performance you observed using the two search algorithms.

I found that on average, the KMP algorithm performed slightly better than the brute force. However, on some occasions the brute force was slightly faster, I was not able to figure out the reason for this, but I assume it is related to the match table values for particular pattern. Ultimately, both algorithms worked as they both found the search string in the given text.

Question 2: Report the binary tree of codes your algorithm generates, and the final size of *War and Peace* after Huffman coding.

input length: 3258246 bytes
output length: 1848598 bytes

original and decoded texts match.

 = 111010
 = 111001
  = 110
! = 1110000111
" = 11111010
' = 111000010
(= 1111101111111
) = 011000111000
* = 11111011010010
, = 1111111
- = 100101001
. = 1110001
/ = 0110001110010101111110
0 = 111110110100001
1 = 11111011010001
2 = 111110110100000
3 = 0110001110010111
4 = 01100011100101010
5 = 0110001110010100
6 = 0110001110010110
7 = 01100011100111110
8 = 0110001110010 0
9 = 01100011100111101
: = 111000001001
; = 111110110101
= = 011000111001010111111
? = 1001010100
A = 011000110
B = 1110000001
C = 01100010000
D = 11111011000
E = 01100010001

F = 11100000101
G = 111110111101
H = 1110000011
I = 100101011
J = 11111011010011
K = 111110111100
L = 1111101111110
M = 1001010101
N = 1110000000
O = 01100011101
P = 011000101
Q = 01100011100111111
R = 11111011011
S = 0110001111
T = 100101000
U = 01100011100110
V = 111000001000
W = 0110001001
X = 011000111001111100
Y = 111110111110
Z = 011000111001110
à = 01100011100101011110
a = 1000
b = 1111100
c = 101111
d = 10110
ä = 011000111001010111010
e = 000
f = 100110
g = 100100
h = 0011
é = 011000111001010111011
i = 0100
j = 11111011001
ê = 011000111001010110
k = 0110000
l = 01101
m = 101110
n = 0101
o = 0111
p = 1111110
q = 11111011101
r = 11110
s = 0010
t = 1010
u = 111011
v = 1001011
w = 100111
x = 1110000110
y = 011001
z = 11111011100
 = 0110001110010101111100

Question 3: Consider the Huffman coding of war_and_peace.txt, taisho.txt, and pi.txt. Which of these achieves the best compression, i.e. the best reduction in size? What makes some of the encodings better than others?

War and Peace reduction: 1409648 bytes
Taisho reduction: 2107288 bytes
Pi reduction: 566371 bytes

The best reduction in size was taisho.txt. This text had the largest alphabet to code (8318 unique characters). On top of this, it also has a wide frequency distribution within the text. This means that the characters compressed to only very few bits are used relatively often, making the overall compression more efficient. By comparison, War and order also has a wide frequency distribution, but not a large enough character base to make it efficient in this compression system.

Question 4: The Lempel-Ziv algorithm has a parameter: the size of the sliding window. On a text of your choice, how does changing the window size affect the quality of the compression?

I used War and Peace as my test text, I tested this text at three different window sizes, 40, 100 and 250. As the window size increased, the ratio of original to compressed got worse. From each compression to the next it got between 50-200 thousand bytes worse. Because of this, I decided my default window size should be relatively small (hence why I picked 40)

Question 5: What happens if you Huffman encode War and Peace *before* applying Lempel-Ziv compression to it? Do you get a smaller file size (in characters) overall?
Original Size: 3258246 bytes
After Huffman: 1848598 bytes
    Size difference: 1409648 bytes SMALLER
After Lempel-Ziv: 110954684 bytes
    Size difference: 109545036 bytes BIGGER

For comparison:
Original to Lempel-Ziv compression size difference: 19755349 bytes BIGGER

This makes the final compressed file much bigger than simply encoding it with the Huffman algorithm. Alternatively, although Lempel-Ziv encoding only results in a size increase, it is still 89789687 bytes smaller than the combined compression.