COMP 261 Assignment 5 Report

In this assignment I finished core completion parts and I did part of the challenge questions.

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

The two different algorithms are KMP and brute force search.

With default text, the performance difference is not very obviously. KMP is only n*1e-3 seconds faster than brute force search. But I tried to build a text data myself, which looks like ("aaaaaaaaaaa……..aaaaaabc") and searched abcd, it turns out KMP is much faster than brute force search. Because brute force search always shifts the window by exactly 1 position to the right, the matching cost for brute force search is O(n*m). For KMP the searching cost is O (m+n) and the pre-processing phase is O(m) which is much faster than using brute force search. So if we have a huge text and we are searching a long pattern better use KMP over brute force search.

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

: 111010

: 111001

: 110

!: 1110000111

": 11111010

': 111000010

(: 1111101111111

): 011000111000

*: 11111011010010

,: 1111111

-: 100101001

.: 1110001

/: 011000111001010111110

0: 111110110100001

1: 11111011010001

2: 111110110100000

3: 0110001110010111

4: 01100011100101010

5: 0110001110010100

6: 0110001110010110

7: 01100011100111110

8: 01100011100100

9: 01100011100111101

:: 111000001001

;: 111110110101

=: 0110001110010101111111

?: 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: 01100011100111100

Y: 111110111110

Z: 011000111001110

à: 0110001110010101110

a: 1000

b: 1111100

c: 101111

d: 10110

ä: 0110001110010101111010

e: 000

f: 100110

g: 100100

h: 0011

é: 0110001110010101111011

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

: 011000111001010111100

input length: 3258246 bytes

output length: 1848598 bytes

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-peace	Apollo	Lenna	Pi	taisho
Input length	3258246 bytes	6815380 bytes	306296 bytes	1010003 bytes	12341256 bytes
Output length	:1848598 bytes	3135673 bytes	155037 bytes	443632 bytes	1542656 bytes
Ratio between	1.7625	2.1735	1.9756	2.2767	8

From the table shown above, taisho.text achieves the best compression. But I think pi should achieves the best compression in theory, because in the pi text should only have numbers from 0-9 which repeated frequently. The most frequent character gets the smallest **code** and the least frequent character gets the largest **code**. I think the reason taisho has the best performance on the ratio, because there are lots Chinese characters, Chinese characters gonna take more space than English letters when we compress it to binary it can save more space.

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?

With window size of 100

Data using : war peace

Input length: 3258227 characters

Output length: 12501166 characters

Window size of 10

Input length: 3258227 characters

Output length: 20992179 characters

With large window size, the size of the output is smaller.

With bigger window size we can find more patterns which can reduce the number of tupples thus it can reduce the size of the output.

Question 5: What happens if you Huffman encode War and Peace before applying LempelZiv compression to it? Do you get a smaller file size (in characters) overall?

Without using Huffman encode

Input length: 3258227 characters

Output length: 12501166 characters

Apply after using Huffman encode

Input length: 14788789 characters

Output length: 18293816 characters

The file size become bigger than without using Huffman encode