CSCI – B505 Applied Algorithms

Programming Assignment – 4

Submitted by – Vishal Singh (singhvis)

Text File used-

http://www.gutenberg.org/ebooks/5200 - Metamorphosis by Franz Kafka

Description-

The Huffman Coding was implemented in following steps-

1. A class 'HeapNode' is defined to construct heap object, a heap object has following features –
    a. Char – the character at the node
    b. Freq – The frequency of the character
    c. Left – The character on the left branch of the node
    d. Right – The character on the right branch of the node
2. We create a frequency dictionary for all the characters in the text of the book using the method 'calculate_frequency'
3. We create a heap node objects for every character using the method 'make_heap'
4. We next combine the characters with lowest frequencies subsequently using the method 'merge_nodes'
5. Codes are assigned to all the characters based on their frequency of occurrence using the method 'char_codes'
6. We remove the characters which do no have ASCII value between 31 and 128
7. Next, we calculate the number of bits taken by doing Huffman Coding by multiplying the frequency of character with the length of codes for that character
8. We compare the number of bits calculated above with the number of bits it would've taken to encode every character in a seven-bit character.
9. The results for the comparison for the given book are as follows-

    Size of encoding using Huffman coding -> 506478 bits

    Number of characters -> 139058

    Size of encoding using a 7-bit fixed length encoding -> 139058 * 7 = 973406 bits

    Number of bits saved -> 973406 – 506478 = 466928

Output-

Character codes:

t 000

m 00100

M 0010100000

N 0010100001

H 001010001

B 0010100100

F 0010100101

j 001010011

' 00101010

; 001010110

ï 0010101110000000

¿ 0010101110000001

0010101110000010

] 0010101110000011

[0010101110000100

» 0010101110000101

Q 001010111000011

K 0010101110001

2 001010111001

Y 00101011101

O 0010101111

, 001011

u 00110

b 001110

p 001111

s 0100

r 0101

e 011

n 1000

i 1001

h 1010

! 10110000000

: 10110000001

U 10110000010

(101100000110

z 101100000111

q 1011000010

C 10110000110

) 101100001110

/ 101100001111

G 10110001

k 1011001

v 1011010

- 1011011000

? 10110110010

1 10110110011

P 1011011010

3 1011011011000

7 10110110110010

% 1011011011001100

X 1011011011001101

@ 10110110110011110

$ 10110110110011111

0 101101101101

D 10110110111

T 101101110

E 1011011110

x 1011011111

y 101110

. 1011110

l 101111100

L 10111110100

R 10111110101

6 10111110110000

J 10111110110001

5 1011111011001

4 10111110110100

V 10111110110101

9 10111110110110

8 10111110110111

W 10111110111

" 101111110

A 1011111110

S 1011111111

a 1100

o 1101

l 11100

d 11101

g 111100

f 111101

c 111110

w 111111

Number of characters which are encoded: 82

Number of characters which are encoded between ASCII values 31 and 128: 79

Character Frequency: {'t': 3, 'm': 5, 'M': 10, 'N': 10, 'H': 9, 'B': 10, 'F': 10, 'j': 9, '"': 8, ';': 9, '#': 16, ']': 16, '[': 16, 'Q': 15, 'K': 13, '2': 12, 'Y': 11, 'O': 10, ',': 6, 'u': 5, 'b': 6, 'p': 6, 's': 4, 'r': 4, 'e': 3, 'n': 4, 'i': 4, 'h': 4, '!': 11, ':': 11, 'U': 11, '(': 12, 'z': 12, 'q': 10, 'C': 11, ')': 12, '/': 12, 'G': 8, 'k': 7, 'v': 7, '-': 10, '?': 11, '1': 11, 'P': 10, '3': 13, '7': 14, '%': 16, 'X': 16, '@': 16, '$': 16, '0': 12, 'D': 11, 'T': 9, 'E': 10, 'x': 10, 'y': 6, '.': 7, 'I': 9, 'L': 11, 'R': 11, '6': 14, 'J': 14, '5': 13, '4': 14, 'V': 14, '9': 14, '8': 14, 'W': 11, '"': 9, 'A': 10, 'S': 10, 'a': 4, 'o': 4, 'l': 5, 'd': 5, 'g': 6, 'f': 6, 'c': 6, 'w': 6}

The text was encoded using 506478 bits

The text had 139058 valid characters

Using a 7-bit fixed length encoding, this would have been 973406 bits long

So we saved 466928 bits!