CS 32 (Data Structures)
Machine Problem 2: Huffman Codes

10 October 2018

1 General Instructions

1. For the machine problem, you are to write your implementation using the **C programming language**. You must use **gcc 5.2.1** as a reference compiler.

2. The name assigned to the source code file that will contain the main method (assuming multiple source code files will be implemented) is **cs32_mp2_<surname>.c** (example: cs32_mp2_yoo.c). Assume that the input file is **on the same folder** as the source code.

3. In the event that multiple source code files were created, place them all in a compressed folder named **cs32_mp2_<surname>.zip**.

4. Place a comment at the beginning of your source code your full name, student number, and section.

5. Add a comment before each function/method in your code, save for the main method, detailing the following: (1) input, if any, (2) output, if any, and (3) what the method does.

6. If you have consulted references (books, journal articles, online materials, other people) *apart from the one used as reference for the specification*, cite them as references by adding a comment detailing the references after your name. Use the MLA format for listing references. (For lines requiring italics, just place them inside quotation marks.)

7. **You are only to submit the source code of your implementation.** Submission of the machine problem should be done via e-mail. Attach the source code file/s, and write as the subject header of the e-mail: **[CS 32] <*Section*> – <*Student Number*> – <*Last Name, First Name*> – MP 2**. For example, [CS 32] WFQR – 202099969 – Yoo, Jeongyeon – MP 2. Send your answers to `john_justine.villar@up.edu.ph`.

8. Submit the signed honor pledge document (uploaded on SCL website) at my pigeonhole on or before the second submission deadline. Note that your submission will not be checked without this document.

9. **You should receive a confirmation e-mail from me stating receipt of your deliverable within 24 hours upon your submission of the MP.** If you have not received any, forward your previous submission using the same subject header once more.

10. If your program does not compile and/or does not run properly for at least one of the test inputs even after minor edits to the source code, then your submission is **deemed invalid and will not be accepted**. You will be asked to submit a working deliverable. This provision will also hold for subsequent (repeat) submissions. Date of submission is considered to be the **date when an acceptable deliverable was submitted**.

11. If you have any questions regarding an item (EXCEPT the answer and solution) in the machine problem, do not hesitate to e-mail me to ask them. However, I will no longer entertain questions that were sent or asked **after 11:59pm of 16 October 2018**.

12. Carefully follow the instructions as mentioned above, especially those pertaining to actual submission of the deliverable. Lack of or erroneous implementation of any part of the instructions **may be considered a deficient submission, and thus may not be accepted**.

13. The deadline for the submission of this machine problem is on **23 October 2018, 11:59pm.**

2 Criteria for Grading

- **Program effectivity (50%)**
  The program should be able to perform the algorithm according to specification on a number of test cases.
- **Adherence to specification (30%)**
  The implementation should, to the fullest extent possible, adhere to the specifications set for the machine problem. Much focus is given to the required feature/s as mentioned.
- **Program documentation and readability (20%)**
  The program should have proper internal documentation (i.e. comments detailing functionalities), and that the documentation details the mechanisms of each function and some code blocks clearly. The code should also be readable in the sense that function and variable names are as intuitive as possible, and that proper indentations/blocking of statements is observed (e.g. statements on the same level/block should be properly aligned, statements inside a for loop should be indented, etc.).

3 Problem Specification

3.1 Problem Introduction

Your goal to create a program that will generate the Huffman encoding of the symbols in a text.

3.2 Problem Background

3.2.1 Data Compression

Data compression (or compression) is the process of representing information as accurate as possible using the less number of bits as the original representation. Data compression is useful because it reduces the usage of resources, such as disk storage and computer memory. As a matter of fact, data compression is used just about everywhere.

To demonstrate compression, consider a 100, 000-character text file which is composed of the symbols `‘a’`,`‘b’`,`‘c’`,`‘d’`,`‘e’`, and `‘f’`. Table 1 shows the frequency of each symbol in the file.

Table 1: Histogram

Symbol	Frequency (in thousands)
a	45
b	13
c	12
d	16
e	9
f	5
TOTAL	100

Initially, our data is represented in ASCII, which means that each symbol is represented using 8 bits. Using this representation, our data can be stored by using a total of $8 \times 100,000 = 800,000$ bits. Table 2 shows the original representation of each symbol. Now if we will represent the data by giving frequent characters less number of bits and infrequent characters more number of bits, we can save a lot of resources. Using the alternative representation of symbols in table 3, our data can be stored by using a total of $[(45 \times 1) + (13 \times 3) + (12 \times 3) + (16 \times 3) + (9 \times 4) + (5 \times 4)] \times 1000 = \mathbf{224,000}$ bits, which is far less than the number of bits used by the original representation.

The alternative representation in Table 3 is called a Huffman code, and it is generated through Huffman coding, an algorithm used for compression.

Given a stream of data composed of different symbols, the goal of this algorithm is to derive a representation of the data according to the frequency of occurrence of the symbols. This algorithm is named after David Huffman.

Table 2: ASCII Representation of the Symbols

Symbol	Code	Binary
a	97	01100001
b	98	01100010
c	99	01100011
d	100	01100100
e	101	01100101
f	102	01100110

Table 3: New Representation

Symbol	Code	Number of Bits
a	0	1
b	101	3
c	100	3
d	111	3
e	1101	4
f	1100	4

3.2.2 Huffman Coding

To construct a Huffman code, the first thing to do is to construct a Huffman tree using Algorithm 1. This algorithm uses a data structure called a priority queue. Unlike the ordinary queue, the behavior of the priority queue in this algorithm is *smallest in, first out.*

Algorithm 1 Constructing a Huffman Tree

```
procedure HUFFMAN($C, F$)
    PQ_INITIALIZE($Q$)
    for $C' \in C$ do
        GETNODE($T$)
        $SYMBOL(T) \leftarrow C'$
        $FREQUENCY(T) \leftarrow F[C']$
        $LSON(T) \leftarrow RSON(T) \leftarrow \Lambda$
        PQ_INSERT($Q, T$)
    end for
    $N \leftarrow SIZE(PQ)$
    for $i \leftarrow 1 : n$ do
        GETNODE($Z$)
        $LSON(Z) \leftarrow PQ\_EXTRACT(Q)$
        $RSON(Z) \leftarrow PQ\_EXTRACT(Q)$
        $FREQUENCY(Z) \leftarrow FREQUENCY(LSON(Z)) + FREQUENCY(RSON(Z))$
        PQ_INSERT($Q, T$)
    end for
    $root \leftarrow$ PQ_EXTRACT($Q$)
    return $root$
end procedure
```

Figure 1 shows the Huffman tree of the given example. By assigning 0s and 1s to the left and right branches, we can generate the Huffman codes by traversing the Huffman tree.

3.3 Problem Description

Implementation Guidelines.

- Implement a heap-based priority queue for the MP.

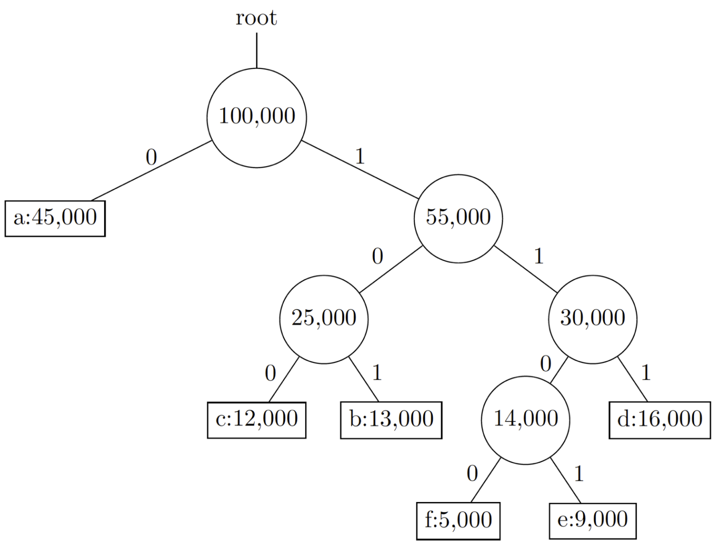

Figure 1: Huffman Tree

- On initializing the contents of the priority queue, insert the tree nodes in increasing order of the symbol's ASCII value. For example, if the symbols in the input are `J, a, T, L, B`, the order of inserting must be `B, J, L, T, a`.
- If at any point there are two or more tree nodes with the same frequency in the priority queue, the FIFO rule will apply on those tree nodes.
- Ignore non-alphabetical characters (numbers, whitespace, punctuation, special symbols, etc.) in generating the Huffman code.

Input Format. The input is plain text. Input will end at the end of file.

Output Format. Output each symbol and their corresponding Huffman codes. Output the symbols in increasing ASCII value. Each code must be followed by a newline (`\n`).

Constraints. There is no limit to the size of the input. Furthermore, you are only allowed to use the following libraries: `stdio.h`, `stdlib.h`, `stdbool.h`, `string.h` and `ctype.h`.

Sample Input. .

```
Bumili ako ng bituka ng butiki sa butika.
Bababa ba? Bababa.
Kakakanan lang sa kangkungan sa may kakahuyan si Ken Ken habang kumakain ng kakaibang
kakanin kahapon.
Makati sa Makati, may pari sa Aparri, mahihilo sa Iloilo at may bagyo sa Baguio.
```

Sample Output. .

```
A 0101100
B 00000
I 0101101
K 010111
M 000010
a 10
b 0110
e 000011
g 0001
```

```
h 01000
i 1110
k 001
l 01001
m 01110
n 1111
o 11001
p 110000
r 110001
s 11010
t 01111
u 11011
y 01010
```

References

[1] Cormen TH, Stein C, Rivest RL, and Leiserson RL. Introduction to Algorithms. McGraw-Hill Higher Education, 2nd ed., 2001.

[2] Wikipedia. Huffman coding – Wikipedia, the free encyclopedia, 2018. [Online; accessed 9 October 2018.]