CIS 22C
Data Abstractions and Structures
Honors Project

A Variation of The Huffman Encoding Algorithm

200 Points

Basic Requirements

Write a program that finds the frequency of each character in a text and builds the corresponding Huffman tree. After the tree has been built, demonstrate encoding and decoding text using the Huffman tree in a menu-driven program as explained below.

Write a menu-driven program with the following options:

- Print the character weights (frequencies)
- Print Tree (Right-Root-Left – indented format)
- Print the Huffman codes as strings for all characters in the list
- Enter one character – print its Huffman code as a string
- Enter a word – print its ASCII binary representation (as strings of 0s and 1s), then its Huffman code (as strings of 0s and 1s)
- Enter an encoded word, decode it, then display it to the screen
- Enter the name of a text file, encode it, and save it to another file.
- Enter the name of the encoded file, decode it, then save it to another file.
- Quit

Data Structures

- Use a hash table to determine the frequency of each character and store the Huffman codes.
- Each letter and its frequency are stored in a binary tree that has only one node. A leaf node in a binary tree has as data the character and its weight. The other nodes have just the accumulated weight.
- We use a singly linked list to keep track of all these one-node binary trees. The linked list is sorted in ascending order by frequency. A node in the list has as data a pointer to a root of a binary tree.

Example

Let’s consider as input just one word: “`Mississippi`”. The table below shows the frequency of each character, its ASCII value in decimal and one-byte binary.

Character	Frequency	ASCII(10)	ASCII(2)
M	1	77	`01001101`
i	4	105	`01101001`
s	4	115	`01110011`
p	2	112	`01110000`

If we replace each letter in Mississippi with its ASCII code get:

0100110101101001011100110111001101101001011100110111001101101001011100000111000001101001

Linked List sorted by frequency:

```
head --->(M,1) --->(p, 2) ---> (i, 4) ---> (s, 4)
```

Building the Huffman Tree

Build new trees by combining two existing trees at a time. Eventually we will obtain only one tree, the Huffman tree. Combine two existing trees by merging the two nodes with minimum weight from the linked list into one; the new linked list is to contain a pointer to a new tree node; the weight of the new tree node is to be the sum of the original weights; its left child is one of the two nodes with the minimum weight, and its right child is the other one.
The process continues until the linked list is reduced to only one node that is containing a pointer to the Huffman tree.

```
head --->(M,1) --->(p, 2) ---> (i, 4) ---> (s, 4)
```

1st merge: (M, 1) and (p, 2), the smallest and second smallest frequencies: 1 + 2 = 3

```
       ( , 3)
       /    \
   (M, 1)   (p, 2)
```

After removing the first two nodes, insert the new node (3) into the sorted linked list:

```
head --->( , 3)---> (i, 4) ---> (s, 4)
         /    \
     (M, 1) (p, 2)
```

2nd merge: (, 3) and (i, 4), the smallest and second smallest frequencies: 3 + 4 = 7

```
             ( , 7)
            /      \
         ( , 3)   (i, 4)
         /    \
     (M, 1) (p, 2)
```

After removing the first two nodes, insert the new node (7) into the sorted linked list:

```
head --->(s, 4) ---> ( , 7)
                     /      \
                  ( , 3)   (i, 4)
                  /    \
              (M, 1) (p, 2)
```

3rd and last merge: (s, 4) and (, 7), the smallest and second smallest frequencies: 4 + 7 = 11.

After removing the first two nodes, insert the new node (, 11) into the sorted linked list. The linked list has only one node that contains a pointer to the root of the Huffman tree.

```
head --->( , 11)
           0/     \1
      (s, 4)    ( , 7)
                0/     \1
             ( , 3)   (i, 4)
             0/    \1
         (M, 1) (p, 2)
```

We can now traverse the tree to get the Huffman codes: left branch is 0, right branch is 1.

Character	Frequency	ASCII(10)	ASCII(2)	Huffman
M	1	77	01001101	100
i	4	105	01101001	11
s	4	115	01110011	0
p	2	112	01110000	101

See below the Huffman encoded "Mississippi".

```
1001100110011101101111
0100110101101001011100110111001101101001011100110111001101101001011100000111000001101001
```

See above the ASCII encoded "Mississippi".

Weekly Reports

Beginning with the fourth week of the quarter submit to Canvas discussion a weekly report showing the status of the Honors Project (design, implementation, debugging and testing, etc.)

Suggested Team Assignments

Student 1:

- Data Structure Design
- main(), menu() and other related functions
- Linked List algorithms
- Integration, testing, and project documentation

Student 2:

- Data Structure Design
- Binary Tree algorithms
- Hash Table algorithms
- Integration, testing, and project documentation

Documentation

One of the students in the team will create a folder named 22C_Honors_Project_x, where x is the group number. Compress it and submit the .zip file (one submission per team). This folder includes two sub-folders:

1. Program Files:
    a. Source Files and Header File(s). For each programmer, clearly indicate the assignment on the project.
    b. Input data files.
    c. Output data files.
    d. Sample screen output.
2. Presentation: Power Point presentation, a Word document, or a .PDF file containing the following sections:
    a. Group number, Project title, team members' names.
    b. Introduction – a short summary describing the project application.
    c. Data Structure Design (diagram) with typical data, showing the relationships among the data (you may use the example on pages 2&3).
    d. Class Diagrams (UML: Linked List, Binary Tree, Hash Table, etc.).
    e. Structure Charts (A Structure Chart is a tree; you may use either the general tree representation, or the indented representation).
    f. A short description of each person's assignment.
    g. Sample screen output.
    h. Anything else you consider relevant such as your thoughts about team collaboration, one of the major debug problems and how did you solve it, etc.
    i. A list of specific applications of Huffman encoding with a brief description.
    j. Conclusion.

Project Score

1. Completeness (80Points - 40%). The project contains all of the required units and functionality along with the required documentation.
2. Accuracy (60Points - 30%). The program runs without errors and provides correct results.
3. Documentation (40Points - 20%). Documentation is complete and readable.
4. Canvas discussion participation and weekly reports (20Points - 10%).

Note

In the Advanced C++ class, after you learn about bitwise operators and binary files you will be able to finish the implementation of the Huffman encoding algorithm.

Test Data

In addition to your own test data, test your program using the following text (in.txt).

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes"[1].

The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted [2]. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all
compression methods - it is replaced with arithmetic coding [3] or asymmetric numeral systems [4] if a better compression ratio is required.

Source: https://en.wikipedia.org/wiki/Huffman_coding