Data Structures BCS-4F
FAST-NU, Lahore, Spring 2021

Homework 5

File Compression using Huffman Encoding

Due: Monday June 21 11:59PM Marked out of 200 points.

In this homework, you will write two programs: compression and decompression, following the algorithms described in class 25. I'll not explain the algorithms in this document. Here, I'm going to provide other information about the two programs: their inputs/outputs and how they should operate.

1. Compression program and the .8b file

You program will accept a text file wit the extension **.8b.** This is a simple text file. You can open it in notepad. You will see that it contains 0's and 1's. These 0's and 1's are simple ASCII based character. For example, an 8b file, called abc.8b, may contain the following content:

0000101100101101000000000000101100001011000000000000000000001011

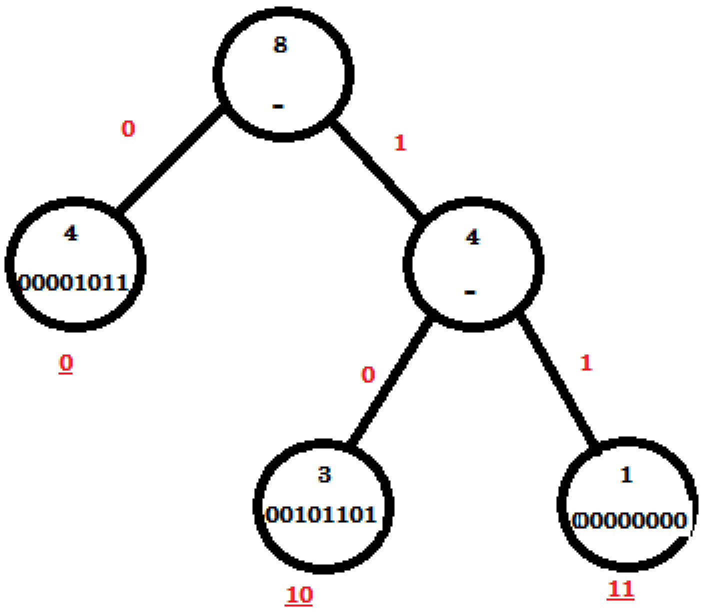
In this case the total number of characters is N = 64. Each group of 8 characters beginning from the start of the file is considered as a single **symbol.** So this file contains 8 symbols. But when you see carefully, you will notice that only three of these symbols are unique. These unique symbols are: 00001011, 00101101 and 00000000, as shown below in red, blue and orange respectively.

0000101100101101000000000000101100001011000000000000000000001011

So this example 8b file contains only 3 unique symbols: namely, 00001011, 00101101 and 00000000, which occur 4, 1 and 3 times respectively. This information should be stored in a **frequency table** by the compression program.

After that, the compression program should run the Huffman algorithm and produce the **encoding tree.** Using the encoding tree, it should create an **encoding table** (this is explained in class 25).

For the example given above, you may end up with a tree and table like this:



The encoding tree for our example

Symbol	Code
00000000	11
00101101	10
00001011	0

The encoding table for the tree given above

The compression program should then produce a compress file called **abc.cmp**. This is also a plain text file which you can open to inspect in notepad. But its size will be much smaller than the 8b file, as the 8-character long symbols have been given much shorter codes.

The compressed file **abc.cmp** will also contain the ending tree in the first line. You may store it in a format that makes it easy to load it during decompression.

In the case of our example, in the abc.cmp file the encoding tree will be followed by the following content (color codes here show which code is for which symbol):

010110011110

As you can see, the original file was N = 64 characters long.

But this compressed like is only n = 12 characters long.

2. **Compression Percentage**

   The compression percentage, p, is computed by the formula: $p = \left(1 - \frac{n}{N}\right) * 100$
   For the example above, $p = \left(1 - \frac{12}{64}\right) * 100$ = 81.25%
   You compression program should print this percentage on the screen, as well as save the abc.cmp file in the current directory.

3. **Decompression program**
   i. Read the compressed file of type .cmp, for example, abc.cmp: read the tree from the first line of this file and reconstruct it.
   ii. Read the compressed sequence of 0's and 1's and using the algorithm described in class 25, produce the decompressed abc.8b file.
   iii. Save this output file in the current directory.

4. **Program interface**

   We should be able to run your programs directly from the console by writing lines like the following and pressing enter:

   ```
   > compress abc.8b
   ```

   Expected Output on screen: compression complete, p = 81.25%

   ```
   >decompress abc.cmp
   ```

   Expected Output on screen: decompression complete

5. **Testing your programs**
   To make it easier for you to test your programs, I am attaching 5 sample .8b files with this assignment statement. Your programs should be able to compress then decompress them. Also, you should note the compression ratios they give you.

--END--