Compressing Files

CS3411 Fall 2017
Program Two
Due: October 19, AoE

In this project, we will develop a program called *encode* which compresses the data stream given to its standard input and write the compressed stream to its standard output and another one called *decode* which decompresses a compressed stream directed at its standard input and writes decompressed data to its standard output.

Similar to many compression algorithms, our compression algorithm tries to represent bytes(s) by using a fewer number of bits. In order to accomplish this, we assign most frequently occuring bytes a smaller number of bits. If a byte is not frequently occuring in the file, we encode it as is. In order for the algorithm to distinguish infrequent bytes from frequent ones, we have to use additional bits and hence pay a price in file system size. Note that, in this document we use the terms *byte*, *symbol*, or *character* to mean the same thing.

The following are the data formats used for encoding/decoding:

An infrequent symbol is a 9 bit quantity such that the leading bit is zero and the next 8 bits indicate the byte value as is.

A byte of zero value is represented using binary 10, i.e., it uses only two bits.

A bit pattern of 11 indicates the following bits encode a repeating pattern such that the first 2 bits give a repeat count (zero means the byte does not repeat) and the following 4 bits represent a frequent symbol code described below.

Bit pattern 11000000 indicates the end of the file and any remaining bits on the input are ignored. In other words, frequent symbol value zero means the end of the file.

The compression algorithm is outlined below:

1. Count the frequency of symbols and sort them from the highest frequency down to lowest. Top 15 characters make-up the frequent characters, and they are assigned codes 1-15. This is the *dictionary*.
2. Output the 15-byte dictionary.
3. Seek the file to the beginning (if the input is not seekable, give an error message and exit).
4. Read the next character.
5. If the current character is a frequent character, calculate the run-length (i.e., find out how many times it repeats) for a maximum of 4. Consume that many characters from the input. Output the current frequent character encoding using 8 bits (i.e., a 11 followed by two bit count followed by the position of that symbol in the dictionary).
6. If the current character is not a frequent character, form an infrequent character encoding by prefixing the current character with a zero bit and output 9 bits of encoded data.
7. Repeat the above steps until the end of the file is reached.
8. Output the end-of-file symbol by padding it with as many zero bits as necessary to bring it to the next byte boundary.

The decoding algorithm is outlined below:

1. Assign Dictionary[0] to zero and load bytes from the file into the *Dictionary* starting at position 1.
2. Read one bit from the file. If it is zero, read the next 8 bits and output is as the next character.
3. If the bit is one, read the next bit. If it is zero, output eight zero bits (null symbol).

4. If both the first and the second bit are one, read 2 more bits and store these into the repeat-count. Read the next 4 bits from the file. This is the character code **c-code**. If both repeat-count and the c-code are zero, close the output file and exit. Otherwise, Dictionary[c-code] gives the symbol. Output this symbol as many times as indicated by the repeat-count. Note that a repeat count of zero means the character appears once.

In the following example, the diff should not report any differences:

```
encode < input-file  | decode > new.input-file
diff input-file new.input-file.
```

Once you make sure your program is working correctly, use your program to compress its own source and its own object, i.e., encode.c and encode. Compress the same programs with gzip as well. Write a README file and indicate the compression ratio's you obtained in this README file.

Submission Requirements

Your submission must be written in C.

Use Canvas to submit a tar file named `prog2.tgz` that contains:

- A copy of the source **with comments**.
- A makefile with *all*, *encode*, *decode* and *clean*.
- A README file showing compression ratios.
- A file named TESTS in the main project directory that contains a description of the test cases you executed against the code to verify correct operation.

When I execute the commands: `gtar zxf prog2.tgz; make` against your submission, two executables named `encode` and `decode` should be created in the current directory.