Compressing Files

CS3411 Fall 2018
Program Three
Due: November 1, 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 program 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 13 bit quantity such that the leading five bits are '11111' and the next 8 bits indicate the byte value as is.

A frequent symbol is a five bit quantity, other than '11111' or '11110'. A 30 entry table supplies the byte value.

A binary value of '11110' indicates end of file.

The compression algorithm is outlined below:

1. Count the frequency of symbols and sort them from the highest frequency down to lowest. Top 30 characters make-up the frequent characters, and they are assigned codes 0-29. This is the *dictionary*.
2. Output the 30-byte dictionary.
3. Seek the input file to the beginning (if the input is not seekable, give an error message and exit).
4. Read the next character.
5. Check if the current character is in the dictionary. If it is, output the dictionary index as a five bit value.
6. If the current character is not in the dictionary, output five one bits followed by the current character.
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:

```
Read the first 30 characters from the file into your dictionary.

repeat
   Read five bits from the file into  c. If  c is the end-of-file symbol,
        close the output stream and exit.
   If  c is the quote character, read 8 more bits and output that as the symbol.
   Otherwise, output Dictionary[c].
until false;
```

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 `prog3.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 prog3.tgz; make` against your submission, two executables named `encode` and `decode` should be created in the current directory.

Clarifications

1. Our char encoding uses 5 bits. This means there are 32 unique encodings. 2 of these are reserved. 11111 to indicate an infrequent character (which is the next byte in the encoded file), and 11110 to indicate end of the file.
2. 11110 is only used by the algorithm to determine where the contents of the encoded file end. Do not write those bits to the decoded file.
3. The dictionary therefore contains 30 entries for frequent chars. Do not write the entries for 11111 and 11110 to the encoded file.
4. The 30 byte dictionary MUST be the first thing in the encoded file.
5. Dictionary index 0 should contain the most frequent character, index 29 should contain the least frequent character.
6. If 2 chars have the same frequency count in the input file, break ties using the ascii value. For example if the frequency of 'a' and 'b' is the same, a should precede b in the dictionary.
7. If there are less than 30 chars in the input, and thus less than 30 entries in the dictionary, you MUST set unused entries to 0.
8. Your encoded file MUST NOT have any filler or padding in between character encodings.
9. Your make file MUST have targets for encode, decode and all where all generates both encode and decode.
10. When creating the encoded file, you must pad the end of it (following the 11110) with 0s to the next byte boundary. These padded zeros will never be used, but do not pad past the byte boundary.