ECE F344: Information Theory and Coding Assignment 1

Pranav Chandra N. V.
2023AAPS0013P

February 19, 2026

Contents

1 Introduction

The goals of this assignment, as per the brief, are listed below:

- Implement Huffman Encoding and Decoding
- Implement Shanon Type Encoding
- Implement Shannon Encoding
- Implement Shannon-Fano Encoding
- Implement Ternary (or higher) based Huffman Encoding
- Provide a comparitive statement between the different encoding efficiencies

The above goals were stated as the minimum required targets. All set targets were achieved, and passed. The submission is able to perform the following:

- Be able to handle both text streams via stdin and .txt files.
- Huffman encoding and decoding.
- Huffman encoding in D-ary type, with $2 \leq D \leq 9$.
- Shanon Type encoding and decoding.
- Shanon encoding and decoding.
- Shannon-Fano encoding and decoding.

2 Methodology

While implementing these algorithms on paper is generally easy, in code it became slightly harder. My selected language was C. The first aspect of implementation involved generating binary (and later on D-ary) trees that could store the incoming symbols for further processing. A lot of these methods involve adding togehter lower ranked nodes into higher one, and the best way to keep track of these additions was through the use of trees. The structure of the comprising nodes for the tree is below:

```
typedef struct node {
    // Node Properties
    int id;
    float prob;
    char symbol;
    // Tree Creation Properties (generalised to D-ary)
    struct node* children[MAX_D];
    int child_count;  // number of children (0 for leaf nodes)
    // Huffman Code Values
    char code[100];
    int code_len;
    // Shannon-Type Code Values
    char st_code[100];
    int st_code_len;
    // Shannon Code Values
    char sh_code[100];
    int sh_code_len;
    // Fano Code Values
    char fano_code[100];
    int fano_code_len;
} node;
```

The node contained variables for id, probability and the symbol it represented. Further, to act as a part of the tree, it required child nodes. In this case, to allow for D-ary coding, an array of pointers of size `MAX_D` was used.

Below this, the huffman code values, shannon-type code values, shannon code values and fano code values were stored within the node in order to ensure that all the data was portable and always together, preventing data handling exceptions.

The primary flow of the code is as follows:

1. Executable is run (with/without argument).
2. If no argument is passed, the user is prompted for a string.
3. Once input data is entered, the text is broken down into symbols.
4. The probability of each unique symbol is computed.
5. The symbol as well as it's associated probability is stored into the node struct.

6. The nodes are assembled into a D-ary tree, and then traversed appropriately to generate the various codes.

7. The codes are stored back to the nodes at all points, to ensure that data loss doesn't occur.

8. The bitstreams of each type of encoding method are genearted and then printed out.

9. The bitstream is passed into a "decode" function along with the base nodes containing the original symbols.

10. The bitstream undergoes decoding. As all codes are "prefix-free", we don't have the issue of code ambiguity, and the bitstream is decoded to characters.

11. The decoded bitstream is printed out.

12. The average coding length is calculated using $L_{avg} = \sum_X L_X \log_D(P(X))$ for each type. Where $P(X)$ corresponds to the probability of character occurance, L_X corresponds to the number of bits required to encode that symbol.

13. The entropy bound of the message of the message is calculated, using $L_{avg} = \sum_X P(X) \log_D(P(X))$.

14. A comparison table is displayed to the user.

3 Operation

The code utilizes no external libraries, save for the standard C libraries like `stdio.h, string.h, stdlib.h, math.h`.

A simple Makefile has been set up. You may run `make` in your terminal to compile the script with the relevant flags. From there, the `itc` executable will appear. Run the executable, either by itself or passing a .txt file as an argument. If no file is passed as an argument, you will be prompted to enter a string. The output will then be printed directly to stdout.