Data (Text File) Compression

ECM1414 Coursework

Term 2, 2021
Submission Date: March 12, 2021

Compression Coursework

ECM1414 Coursework

Introduction

Data compression is part of almost every activity we do in computer science. We compress data before transmitting, we compress video files, we compress music files, etc. Many of the file formats you use such as jpeg, mp3, tiff, mp4, to name a few. The principle is quite simple and despite the many ways compression is implemented, the general goal is to look at repetitions within the data that would allow you to replace a large piece of repeated data with a shorter version and hence save space. Figure 1 shows the general idea of compression.

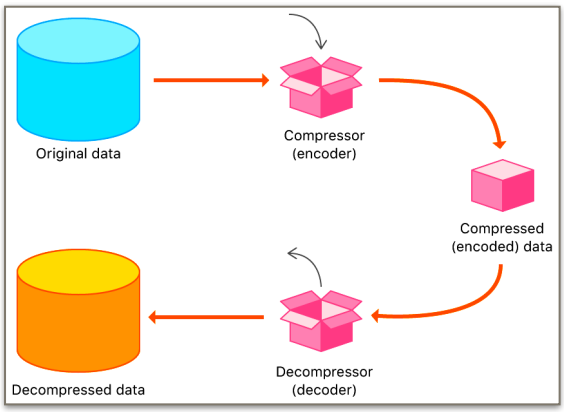


Figure 1: The general schematics of a compression algorithm. (1) The original data is run through an encoder and a (2) compressed file is generated. The compressed data should be smaller than the original data and it includes enough information to allow for the data to be (3) run through a decoder. In a lossless algorithm, the decompressed data (in orange) should be the same as the original data (in blue).

Your coursework this term consists of implementing any lossless compression method and demonstrate its efficiency with given datasets. In this coursework, you are free to implement any compression method of your choice and the coursework consist of researching a method, implementing it in Python or Java (you are not allowed to use compression libraries), test your compression using several datasets, run statistics regarding the level of compression you can achieve, and generating a 5-page report. The minimum requirement would be the implementation of compression based on Huffman coding (explained below).

Huffman coding

Huffman coding was the state of the art in compression in the 1950s and at the time it was considered a breakthrough. It's a greedy algorithm and interesting

because it's simple and easy to implement. The "code" itself consists of a tree that contains the encoding of the characters present in the data (text) to be compressed (see Figure 2).

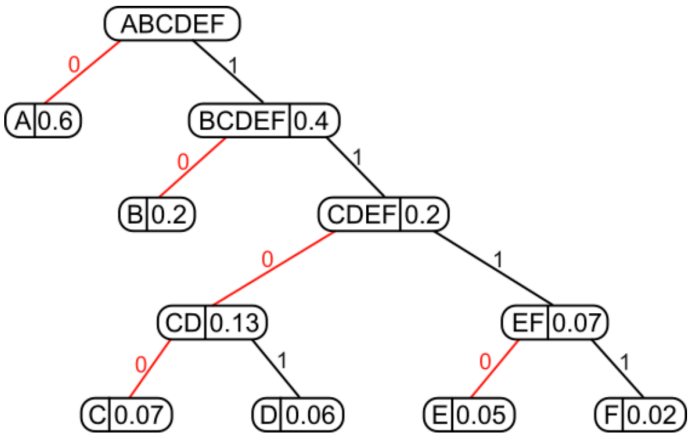


Figure 2: Encoding of characters based on their frequency. More frequent characters are encoded with smaller codes. For instance, the tree below encodes A with 1 bit (0) while it encodes F with 4 bits (1111). (picture from Manning Free Content Center)

In our project, you may assume the data you have to compress is a text file. Hence the input will be a text file such as a book, emails, etc. Later I recommend a few datasets for you to test your implementation.

Let's assume that the only characters in your text are A, B, C, D, E and F. If we want to propose a binary representation for each letter we'd need 3 bits (2 bits can only represent 4 letters). If the file has 1000 characters, we'd need 3000 bits. Can we gain some space? The answer is yes but it all depends on the frequency of each letter. If the frequencies are A=450, B=130, C=120, D=160, E=90, F=50 we could use the following encoding A=0, B=101, C=100, D=111, E=1101, F=1100, which would result in a compressed file of 2240 bits instead of the original 3000 bits. The encoding above works because no code is a prefix of another code. So the representation 1000101 can be easily decoded into CAB.

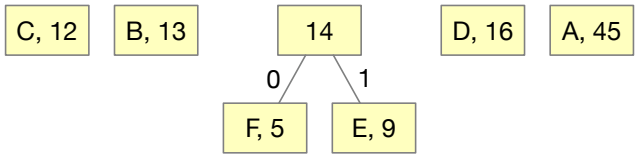
How can one generate this encoding? This is what Huffman has proposed and the algorithm has the following steps:

1. Find the number of occurrences (frequency) for each character in the text.
2. For each (distinct) character create a node. The frequency of that character should be stored in that node (as below)

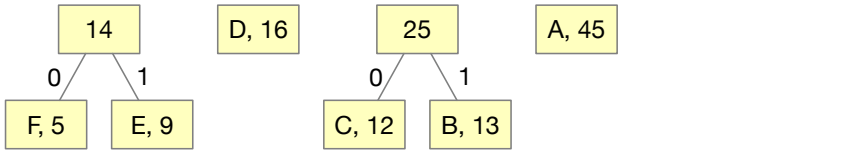
F, 5	E, 9	C, 12	B, 13	D, 16	A, 45

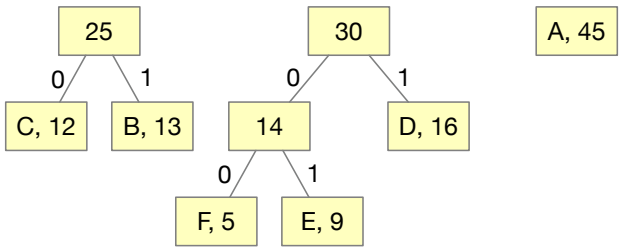
3. Find the two least frequent nodes, *x* and *y*

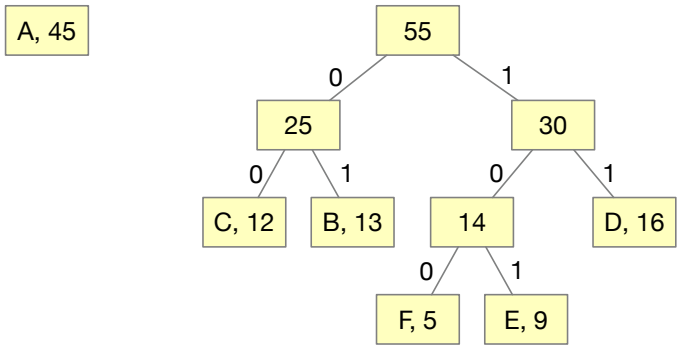
4. Create a new node z and make x and y its children in the tree. The frequency of z is the sum of the frequencies of x and y. Given we're using a binary encoding, you can assign 0 and 1 for the code for F and E respectively.

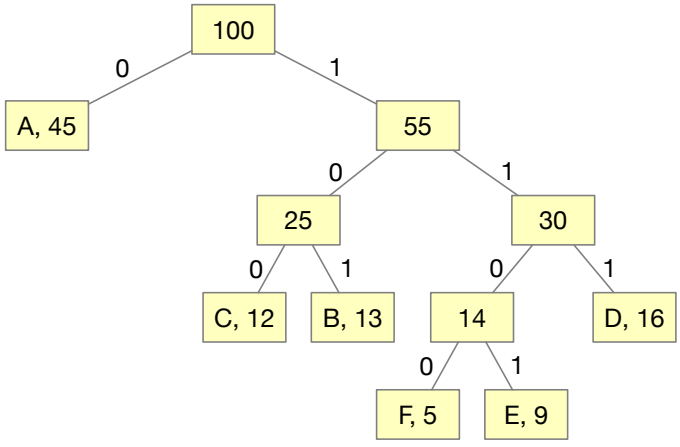


5. Repeat step 4 until there is only one node left. This node is the root of your Huffman tree (see steps below)









This example has been taken from the Algorithms book by Cormen et al. I recommend you to use the book (Section 16.3) during your implementation.

Note that you can implement other compression algorithms if you choose to do so. Huffman is the minimum you can submit. There are other more interesting approaches which involve Arithmetic Coding, Relative Lempel-Ziv, Dictionary-based Compression, Fixed-Block Compression, etc.

What Should be Submitted and How?

You are required to submit your course work by the deadline using BART and E-Submit. The following is required:

- All the source code (Python or Java) (50%)
- A 5-page PDF specification (±1 page) of your work which should include the following sections (40%). This should be written in Times New Roman 11pt with 1-inch margins (top, bottom, left, and right).
    - Literature review of compression algorithms (2-3 pages). This should describe at least 2 additional approaches. That is, in addition to the compression approach you implemented you have to describe 2 others.
    - List of all data structures and algorithms used in your implementation. This refers to data structures and algorithms that we have studied in the module. It is a requirement for you to explain how these were used.
    - Weekly log of your progress (0.5 pages). What has been implemented and when.
    - Performance analysis. This is not a theoretical analysis using asymptotic notations. Instead, this refers to an analysis of different text datasets. Compression algorithms tend to do well in certain scenarios and not so well in others. For instance, the frequency of letters is completely different in different languages. Hence you should test your compression with
        - At least three languages: English, French and Portuguese. This allows you to verify the effectiveness of a encoding with another language

        - In order for the analysis to be effective you need to use the encoding of one language with another. For instance. You could take a book in English, say David Copperfield, and generate the encoding for it. Then you can take the same book in French and Portuguese and see the level of compression you achieve with the English encoding.
    - At least 2 real books in each language. Look at project Gutenberg (https://www.gutenberg.org)
    - At least 1 dataset from the repetitive corpus dataset (http://pizzachili.dcc.uchile.cl/repcorpus.html). 1 from "Artificial", 1 from "Pseudo-real", 1 from "Real".
  - To support the performance analysis, you should include charts that support your performance claims. The charts should clearly show the performance of your compression in comparison with the original file.
  - The size of the compressed file should also account for the size of the Huffman encoding (the tree itself).
  - List of references used to implement your compression algorithm
- A video recording (screen capture video with a demo of your system). You may also provide a link if you decide to place it on your website. Note that this should not exceed 2 minutes. (10%)

More Information

The coursework, if done well, should be a great resource for you to present to employers and discussing what you did in the university. Design it in a way that it becomes part of your e-portfolio. This is not required but it is strongly encouraged. You should think of having a site, and code repositories available (e.g. GitHub).

Marking Scheme for Code Submission

The following will be considered when marking your code:

- Coding Style (10%)
- Commenting (10%)
- The program compiles/run correctly (10%)

- The program works with all test cases (15%)
- Correct output of compressed files (15%)
- Correct use of data structures (30%)
- Flexibility (10%)
- Advanced features (extra 10%)
  - Advanced features include (but not limited to) the implementation of a compression algorithm different from Huffman coding, advanced interface for the program, organisation of code in GitHub, etc.