12.03.2021

ECM1414 Continuous Assessment

Alperen Türkmen

at753@exeter.ac.uk

Data compression:

Data compression methods aim to make a data take up less space than its original. There are two types of data compression: Lossy and Lossless. Huffman method is a Lossless data compression method. We do not lose any data if we use it to compress. Lossy method causes a data loss as in its name.

David Huffman:

David Albert Huffman has lived between 1925-1999. His bachelor's degree is electrical engineering from Ohio State University, his master's degree on same discipline was from Ohio State University as well. He earned his Doctor of Science in MIT. [1]

Huffman Coding:

Huffman coding was developed by David A. Huffman in 1952 while he was a Sc.D. student at MIT. This is a definition by National Institute of Standards and Technology:
"Definition: A minimal variable-length character coding based on the frequency of each character. First, each character becomes a one-node binary tree, with the character as the only node. The character's frequency is the tree's frequency. Two trees with the least frequencies are joined as the subtrees of a new root that is assigned the sum of their frequencies. Repeat until all characters are in one tree. One code bit represents each level. Thus more frequent characters are near the root and are coded with few bits, and rare characters are far from the root and are coded with many bits. " [2]

How Huffman Encoding works:

The method works by first finding the frequency of each character in the given input. Then the algorithm should create a node tree according to the frequencies, taking the smallest 2 values, marking them as 1/0, and creating a new node on top of them. When the algorithm iterates over the entire character list, we will see that the algorithm has created a node tree. To reach the characters over nodes, we should record the path to a character. In this way, we can see a code (with binary digits) of the character.

Shannon-Fano Compression:

It's been developed by Claude Shannon and Robert M. Fano in. It is an example of lossless data compression methods.

First, we calculate the frequency of the characters. It works with binary tree logic. We divide the elements, if we are closer to 50%, it will give more efficient results. We give 0 to left, 1 to right, we

divide it equally again (Not equally but the closest to equal.) We give 0s and 1s again. We do this until all elements are tagged with a unique binary code. Then we create a binary file using these keys. To decode the binary, we will use keys again to read and translate the file to its decompressed state. [3]

Shannon-Fano vs. Huffman:

"In general, Shannon-Fano and Huffman coding will always be similar in size. However, Huffman coding will always at least equal the efficiency of the Shannon-Fano method, and thus has become the preferred coding method of its type (Nelson, 38)." [4]

Data compression in real life:

As I experienced, I can tell that the idea of Huffman method is also used in natural languages. I will now give some examples for it. Last summer, I have learned the Korean alphabet just for fun. And I've seen that they use vowels just as straight lines, this saves a lot of time when writing. Because they're used the most. I then realized that this is also the case in Latin and Cyrillic alphabets. I don't know the Arabic alphabet, but I know the logic which they use quite little scratches and points as vowels. They only write the consonants, then they add the vowels as little touches. This saves a lot of space and ink. We can also think in a word-wise way, we can see that some words that are used frequently are named short. But there is a lot of exceptions in world languages, so I will pass this one.

JPEG Compression:

Definition: "JPEG is an acronym for Joint Photographic Experts Group and JFIF is an acronym JPEG File Interchange Format. Normally, the files with the .JPG extension are JFIF files. If a JPEG file is loaded into a text editor such as Window's NotePad , the string JFIF is started in the first line starting at the 7th character. JPEG compression is mainly discards much of the original information so the exact original image cannot be reconstructed from a JPEG file. This is called lossy compression. While this sounds bad, a photograph actually contains considerable information that the human eye cannot detect so this can be safely discarded." [5]

Data structure types:

Strings are arrays. I used them to store the concatenated characters, and used it as a mid-step from raw string to binary file by storing binary numbers in a string.

Dictionary in python is an instance of hash map. It is mapped from characters to binary codes. I used this to pair characters with their binary code values. And I stored it in JSON files to increase flexibility.

My Huffman code tree is an example of Binary Tree. It's an object containing 2 nodes(objects), all the way up to individual characters. I used this structure to create a coding system in an efficient way.

bytearray is an array. It consists of bytes. I used this to transform a string to a bin file, not using 8-bytes for a "1" character, using a bit instead of that.

Algorithm types:

huffman_code_tree(): Recursive algorithm. Because it invokes itself until the result is reached. It does it to create a binary code for each character. Every step, it goes to each children of a node.

huffman_decode(): Brute Force Algorithm. Because it iterates over all text, and on each character, it iterates over entire directory to find a match. It takes the most of the runtime, not very efficient. There should be other ways to do it.

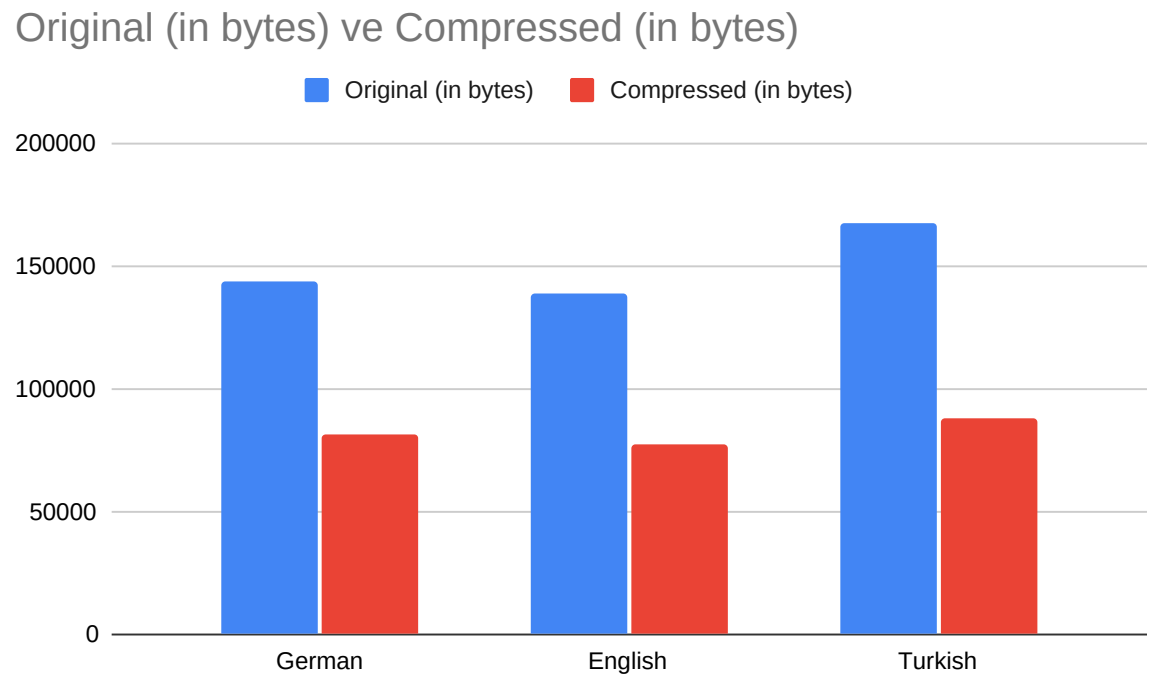
to_bytes(): Divide&Conquer Algorithm. Because it divides the string into 8-byte parts. Then appends the byte array by these 8 characters.

remove_padding(): Brute Force Algorithm. Because it flows one by one.

pad_encoded_text(): Brute Force Algorithm just like remove_padding().

huffman_encode:Also Brute Force Algorithm. Does the reverse of huffman_decode method.

Analysis Chart:



I'd like to add that:

German decompression took 13.53 seconds
English decompression took 13.36 seconds.
Turkish decompression took 14.47 seconds.

Weekly progress:

Week 1: I watched the video on the first day, I made some researches for Huffman coding. The primal idea has started to form in my mind.

I've looked into java and python instances, I've noticed that different infrastructures has been used in them, I've decided to do it in Python, given my fluency and its usefulness.

Weeks between: I tried to do my best in all modules.

Week of 15/02: I draw a sketchy flowchart on how to implement my ideas.

I've drawn a flowchart of the algorithm. I planned to use classes and arrays. I've did some small examples on pen and paper. Created Huffman trees for them.

Week of 8/03: I coded the entire program starting from Saturday, added this pdf, took the video.

I've read the files with python, encoded with Huffman. The program has been successful in tests, debugs. But I couldn't manage to translate the I/O string to binary file. I've researched for it, dealt with it. Committed changes in GitHub.

Tested binary encode and decode on big files, but it went in an infinite loop because of the bug/mistake in the last byte. It costed me a lot of time to solve. I've seen the problem in debug mode, solved it. Because the program filled the last byte's first empty characters with 0s. I've added a padding-remove padding methods to cope with this. I used those respectively in encode-decode.

I added a JSON file last day, wrote and read from it instead of using the same variable.

Bibliography

[1] “IEEE Computer Society,” [Online]. Available: https://www.computer.org/profiles/david-huffman.

[2] "National Institute of Standards and Technology," [Online]. Available: https://xlinux.nist.gov/dads/HTML/huffmanCoding.html.

[3] S. E. Seker, "BilgisayarKavramlari," [Online]. Available: https://www.youtube.com/watch?v=NwyFwGRL3eE.

[4] "Stanford University Computer Science," [Online]. Available: https://cs.stanford.edu/people/eroberts/courses/soco/projects/data-compression/lossless/shannonfano/comparison.htm.

[5] "UCDAVIS Electrical and Computer Engineering," [Online]. Available: https://www.ece.ucdavis.edu/cerl/reliablejpeg/compression/.

Code references

Padding idea:

[6] B. Srivastava. [Online]. Available: https://bhrigu.me/post/huffman-coding-python-implementation/.

Main algorithm idea:

[7] "Programiz," [Online]. Available: https://www.programiz.com/dsa/huffman-coding