Generally, our work can be divided into two parts: compression and uncompression.

* In the first part, we generate a Huffman Tree based on the frequency dictionary. Then, we encode the original text to Huffman-Tree-code text by using the tree we already have. Subsequently, we store the Huffman Tree into a byte type.
* In the second part, we generate the Huffman Tree basing on Huffman-Tree bytes. Then we use the Huffman Tree to decode the Huffman-Tree-code text. Then we get the decoded text.

Compare the compression ratio between Huffman Tree and Mac itself.

|  |  |  |
| --- | --- | --- |
|  | Huffman Tree | Mac |
| music.mp3 | 99.8%(476/477) | 99.4%(474/477) |
| music.wav | 94.6%(831/878) | 94.3%(828/878) |
| dan.bmp | 36.1%(481/1.3\*1024) | 20.1%(267/1.3\*1024) |
| book.txt | 55.7%(684/1.2\*1024) | 36.8%(452/1.2\*1024) |

As the compression ratio shown above, we can find two results. First one is that both Huffman Tree compression and Mac compression perform better on non-music type file (bmp & txt) than music type file. Second one is that Huffman Tree compression ratio is higher than Mac compression ratio. For example, the compression ratio for dan.bmp is 20.1% by using Mac compression and the compression ratio for dan.bmp is 36.1% by using Huffman tree compression.