|  |  |
| --- | --- |
|  |  |
| Huffman Compression Technique |  |
|  |  |
|  | DSAI 325Information Theory |
|  | Mohamed Ehab Yousri202201236 |

**Huffman Compression Technique**

The **Huffman Compression Technique** is a **lossless data compression** algorithm that uses variable-length codes for encoding symbols based on their frequencies of occurrence. More frequent symbols are encoded using shorter codes, while less frequent symbols are given longer codes. This technique ensures that the compressed data size is minimized by exploiting the frequency distribution of characters in the input text.

Implementation Details:

Compression Algorithm:

-Frequency Count: First, the algorithm counts the frequency of each character in the text.

-Building the Huffman Tree: Using these frequencies, a Huffman tree is constructed. The tree is a binary tree where each leaf node represents a character, and the path from the root to the leaf node represents the Huffman code for that character.

-Assigning Codes: Shorter codes are assigned to more frequent characters, and longer codes are assigned to less frequent characters, ensuring that the total size of the encoded data is minimized.

-Encoding the Input Text: Each character in the input text is replaced with its corresponding Huffman code.

-Output: The encoded data is then stored in a binary format, which is smaller in size compared to the original text.

Decompression Algorithm:

-Reading the Encoded Data: The compressed data, which consists of Huffman codes, is read.

-Rebuilding the Huffman Tree: Using the saved Huffman codes, the algorithm reconstructs the Huffman tree.

-Decoding the Text: The encoded data is traversed, and each Huffman code is decoded by following the tree, ultimately recovering the original text.

The Main Components:

-Huffman Tree: The binary tree built from the input symbol frequency. Each leaf node contains a symbol, and each internal node contains the sum of the frequencies of its children.

-Character Codes: The variable-length binary codes assigned to each character, based on their frequencies. More frequent characters are assigned to shorter codes, and less frequent characters are assigned longer codes.

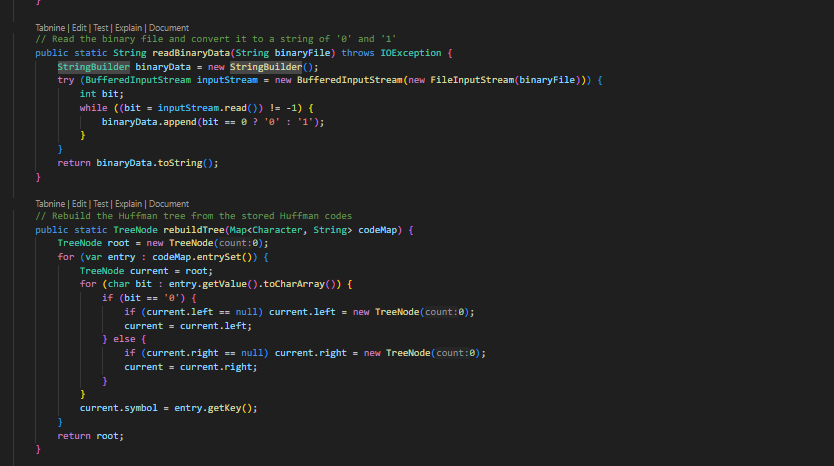
The Algorithm Implementation:

Compression Algorithm:

A screen shot of a computer program

AI-generated content may be incorrect.





Decompression Algorithm:

A screen shot of a computer screen

AI-generated content may be incorrect.

Main :  
A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

Test Cases:

Test Case 1:

Original:

A black background with white text

AI-generated content may be incorrect.

**Original Size:** 16 characters × 8 bits = 128 bits

Compressed:

A screenshot of a computer

AI-generated content may be incorrect.

**Compressed Size:** 8bits (A)+12bits (B)+6bits (C)+6bits (D)= 32bits

Entropy of the text: 1.9056390622295662

Average code length: 2.0 bits/symbol

Test Case 2:

Original:

A black background with white text

AI-generated content may be incorrect.

**Original Size:** 14 characters × 8 bits = 112 bits

Compressed:

A screenshot of a computer program

AI-generated content may be incorrect.

**Compressed Size:** 4bits (X)+4bits (Y)+5bits (Z)+6bits (A)+6bits (B)+6bits (C)+3bits (D)= 34bits

Entropy of the text: 2.6031579868726

Average code length: 2.615384615384615 bits/symbol

The Full Running Process :

A computer screen shot

AI-generated content may be incorrect.

References

<https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/>

https://youtu.be/P7BPI0dP-SU?feature=shared

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/PriorityQueue.html>