|  |  |
| --- | --- |
|  |  |
| Adaptive Huffman |  |
|  |  |
|  | DSAI 325Mini- Project |
|  | Mohamed Ehab Yousri202201236 |

Adaptive Huffman

Adaptive Huffman Coding is a lossless data compression technique that dynamically adjusts its encoding scheme as it processes the input data. Unlike static Huffman coding, which requires a frequency table to be generated before encoding, adaptive Huffman coding builds and updates the tree in real time. This allows for efficient compression in scenarios where the data is not available in advance, making it ideal for streaming data.

**Implementation Details:**

**Compression Algorithm:**

The compression process in Adaptive Huffman Coding works by the following steps:

**-Initialization**: The algorithm starts with an empty Huffman tree, with a special “Not Yet Transmitted” (NYT) node as the root.

**-Encoding Process**: For each character in the input:

If the character has been seen before, the corresponding Huffman code is appended to the encoded string.

If it is a new character, the NYT node is used to encode it. The NYT node is replaced by an internal node, and a new leaf node for the new character is added.

The newly added nodes are assigned a code, and the tree is updated dynamically to maintain the Huffman properties.

**-Tree Update**: After encoding a character, the tree is updated by incrementing the frequency count of the nodes and swapping them to maintain optimal Huffman properties (nodes with higher frequencies are moved closer to the root).

**Decompression Algorithm:**

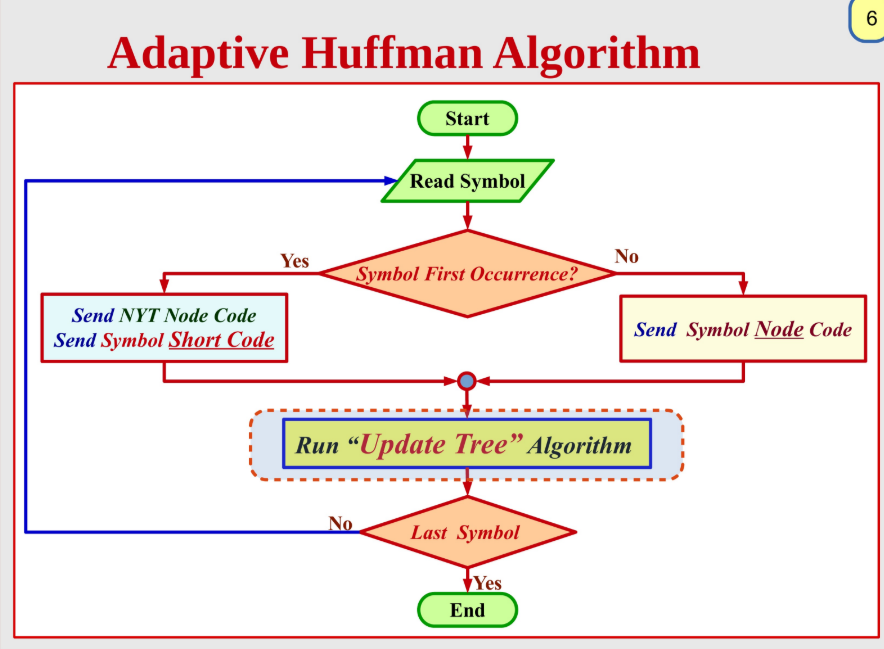
The decompression process works by reading the encoded binary stream and reconstructing the original text using the Huffman tree:

**-Decoding Process**: The algorithm begins by traversing the Huffman tree using the binary stream. It follows the path determined by 0 (left) and 1 (right) bits.

**-Handling New Characters**: If the path leads to the NYT node, the next 8 bits are read to decode the ASCII value of the new character. The NYT node is then updated to encode the new character in subsequent bits.

**-Tree Update**:After decoding each character, the tree is updated similarly to the encoding process, incrementing frequencies and adjusting the tree structure.

**-Reconstructing the Text**: The process continues until all bits are decoded, and the original text is reconstructed.



A diagram of a tree

AI-generated content may be incorrect.

The Algorithm Implementation:

AdaptiveHuffman.java:

A screen shot of a computer program

AI-generated content may be incorrect.

Encoder.java:



Decoder.java:

A screen shot of a computer program

AI-generated content may be incorrect.

Node.java:

A screen shot of a computer program

AI-generated content may be incorrect.

HuffmanTree.java:

A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

AdaptiveHuffmanTest.java:

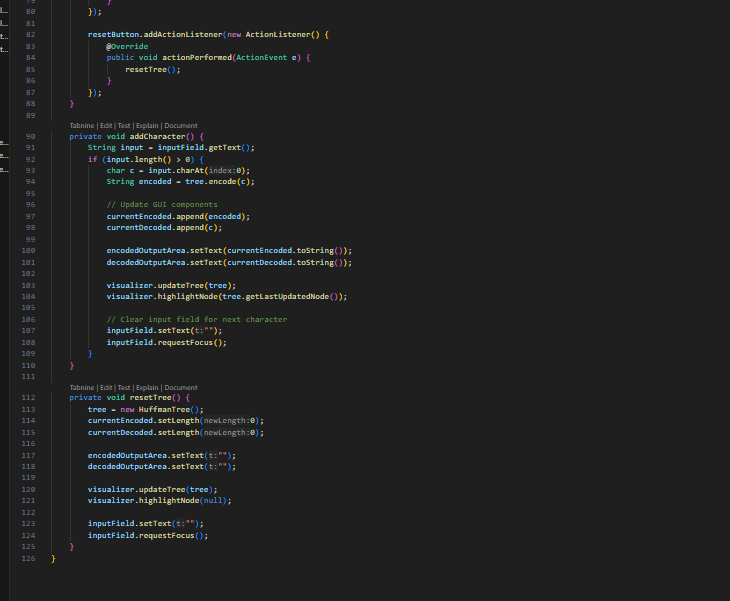
A screen shot of a computer

AI-generated content may be incorrect.

AdaptiveHuffmanGUI.java

A screen shot of a computer program

AI-generated content may be incorrect.



RunVisualizer.java:

A screen shot of a computer code

AI-generated content may be incorrect.

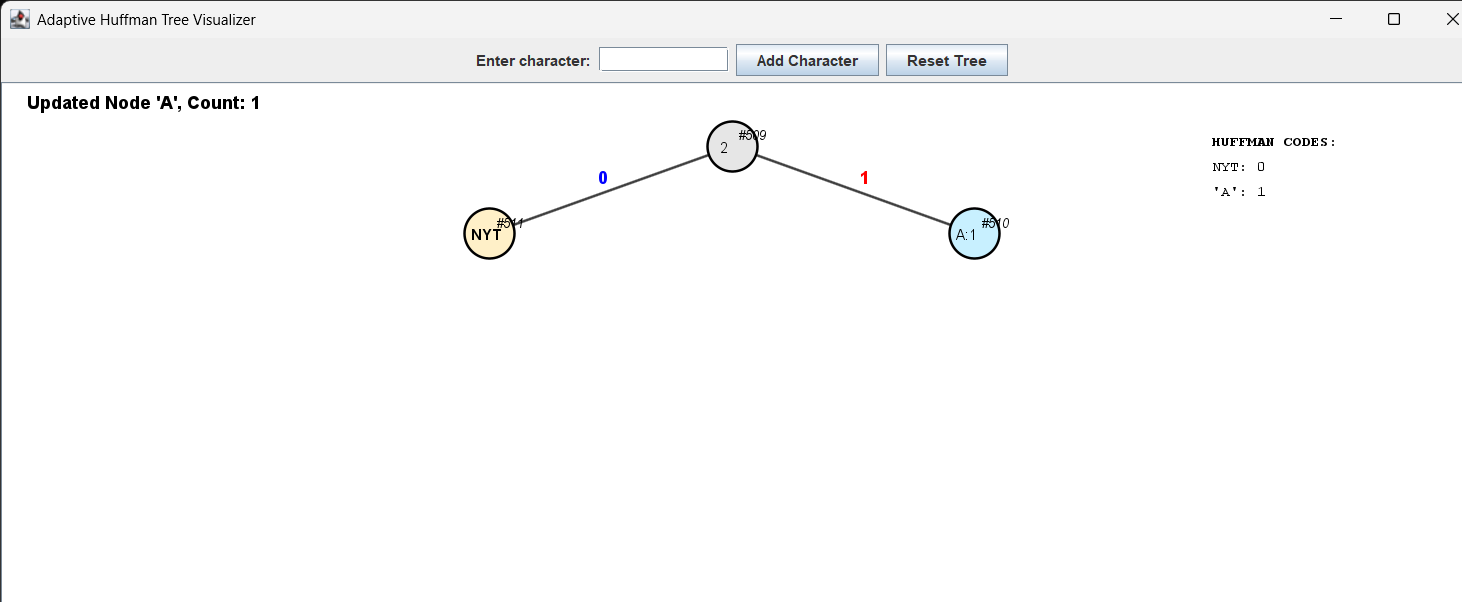
Test Cases:

A screenshot of a computer

AI-generated content may be incorrect.

Visualization with Swing to the first test case:

After Adding A:



After Adding B:

A screenshot of a computer

AI-generated content may be incorrect.

Second A :

A screenshot of a computer

AI-generated content may be incorrect.

Second B:

A screenshot of a computer screen

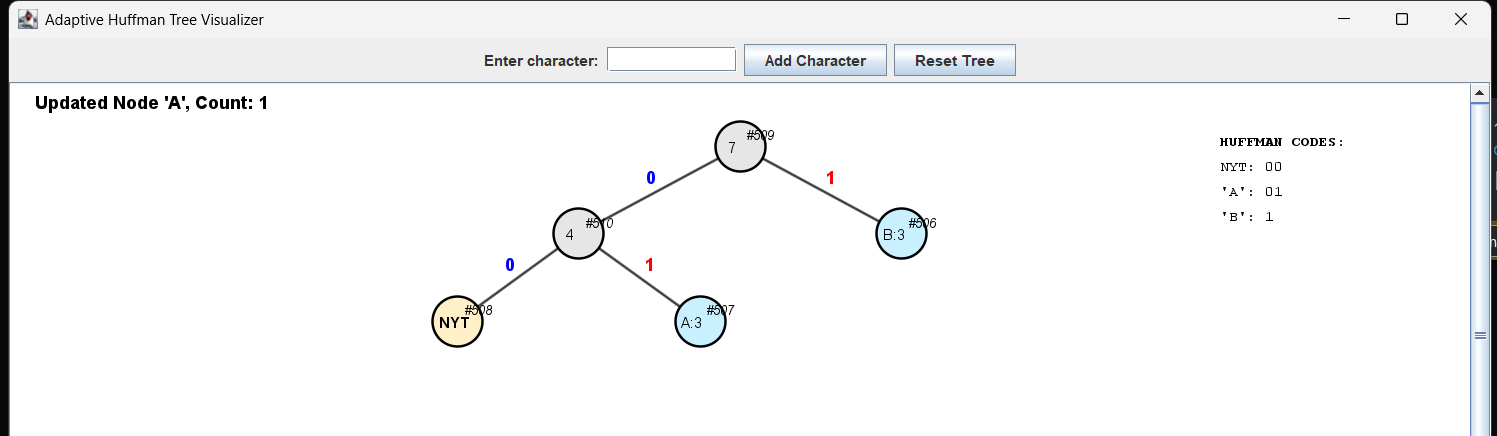
AI-generated content may be incorrect.

Third A:

A screenshot of a computer

AI-generated content may be incorrect.

Third B:



References

<https://www.youtube.com/watch?v=TFW4OWpw4f8>

<https://ben-tanen.com/adaptive-huffman/>

https://experiencestack.co/adaptive-huffman-coding-2f6379bc23fe