

ASSIGNMENT COVER SHEET

Student Name: Daniel Aigbe

ID Number: B00150058

Course: TU860/757

Year: Year 3

Module: Data Structures and Algorithms

Lecturer: Dr. Simon McLoughlin

Title of Assignment: Assignment 1: Huffman Coding

Due Date: 25th November (11.59 pm deadline) on Brightspace

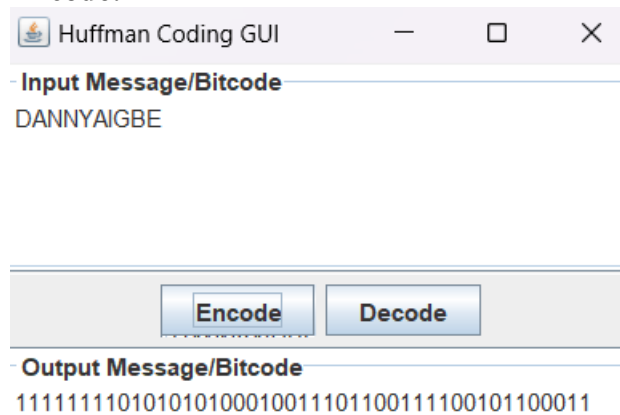
The material contained in this assignment is the author's original work, except where work quoted is duly acknowledged in the text. No aspect of this assignment has been previously submitted for assessment in any other unit or course.

Signed: Daniel Aigbe

Date: 23 / 11 / 2024

Screenshots of code running:

Encode:



The screenshot shows a window titled "Huffman Coding GUI" with standard window controls. It features two input fields: "Input Message/Bitcode" containing the text "DANNYAIGBE" and "Output Message/Bitcode" containing the binary string "111111110101010100010011101100111100101100011". Between these fields are two buttons, "Encode" and "Decode", with "Encode" being the active button.

Huffman Coding GUI

Input Message/Bitcode

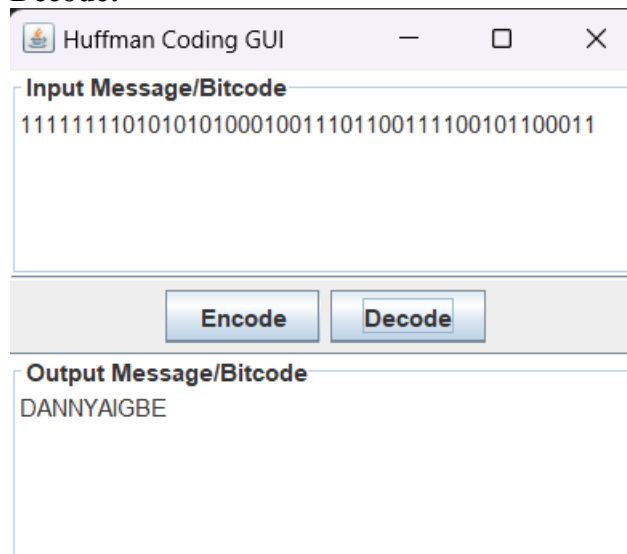
DANNYAIGBE

Encode Decode

Output Message/Bitcode

111111110101010100010011101100111100101100011

Decode:



The screenshot shows the same "Huffman Coding GUI" window, but now the "Decode" button is active. The "Input Message/Bitcode" field contains the binary string "111111110101010100010011101100111100101100011", and the "Output Message/Bitcode" field contains the decoded text "DANNYAIGBE".

Huffman Coding GUI

Input Message/Bitcode

111111110101010100010011101100111100101100011

Encode Decode

Output Message/Bitcode

DANNYAIGBE

Methods Used and Explanation:

readFrequencyTable(String filename): This method was used to read symbol frequencies from a file and it stores them in a list, and then it sorts them by frequency. It uses the frequency table that was provided by the lecturer.

generateHuffmanTree(): builds the huffman tree by combining nodes with the lowest frequencies.

Encode(String input): Encodes a string using the generated Huffman codes as shown in the images above.

buildCodes(TreeNode node, String code, List<CodePair> huffmanCodes): Recursively builds Huffman codes for each tree node.

decode(String encoded): Decodes a binary string using the Huffman Tree.

CodePair class: Stores a symbol and its corresponding Huffman code.

Algorithm used: Huffman coding – Greedy Algorithm

I used Huffman Coding because it is a highly efficient, well-established algorithm for compressing data without loss, making it suitable for applications where minimizing file size while retaining full data integrity is essential. The code implements the steps for building the Huffman Tree, generating codes, and encoding/decoding data, all of which are crucial for effective data compression.

The algorithm is **greedy**, meaning it makes locally optimal choices (combining the two least frequent symbols) at each step. T

I chose this algorithm because it ensures that the overall structure of the Huffman Tree is optimal, which is crucial for achieving good compression.