ASSIGNMENT COVER SHEET

Student Name:	**David Thornton**
ID Number:	**B00152842**
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: David Thornton

Date: 24 / 11 / 2024

Introduction

This is a report detailing a program which will create a Huffman tree and will then encode and decode messages using this tree. A Huffman tree works by using character symbols frequency to encode messages, giving more commonly used characters a shorter encoded string then those used less often. The codes are found by navigating the Huffman tree to find the character. On the path to the tree, if a left child node is taken, a 0 is added to the code for that letter, and a 1 for the right node. To ensure that the more commonly used symbols have shorter code, they should be higher in the tree.

Solution Design

Generating tree

The method used to generate the tree, as described in the assignment, involves generating Tree nodes with each of the symbol frequency pairs form the frequency table. Then, the two nodes with the lowest frequency are taken from the tree node list, their frequencies are added to get the frequency for a new node, which will have no (null) symbol and each of the two lowest nodes as its child nodes, with the lowest frequency of the two being the left node and the other being the right. This new null symbol node will then be added to the node list, positioned relative to its frequency. This process is then continued till a full tree is generated and there is only one node left in the node list, which will be the root node. Encoding and decoding algorithms will use this root node and its child nodes to carry out their processes.

With this design, letters with a higher frequency (that appear and are used more often) will have shorted encoded strings the those with lower frequency, meaning the resulting encoded string will be as short as possible. This is because they are the last to be added to the tree, and as a result will be closer to the root.

Encoding a String

Before encoding a string, an arrays is first built with each of the codes associated with each symbol. To construct this array, the tree is traversed in an in order depth first search. This involves starting at the root node for the Huffman tree, going down each node till a leaf node is reached, then moving back to the previous node and searching its child nodes till nun remain, and continuing this cycle till all nodes have been searched. To construct the codes during this process, when going down a left node a 0 will be added to the code and a 1 will be added if a right node is taken. Once a leaf node is reached, the full code for that symbol will be generated and it can be added to the list of codes to be used to encode the string.

Once this list has been generated, when a message is entered to be encoded a search is done to find the code for each symbol in the message to make the encoded string.

Decoding a String

The method used to decode a string involves traversing the tree from the root, going left if the next number of the encode string is a 0 and right if it is a 1. This is repeated until a leaf node is reached, meaning the symbol associated with the current set of numbers has been found. This symbol will then be added to the decoded string, and the process of finding the symbols will restart, going back to the root node, and will resume searching the tree, starting at the next character in the encoded string.

Design Problems

Problem 1

Problem: If a user entered an empty string “”, no warning to tell the user that the input was incorrect was given.

Solution: Check if input length is 0, if so, generate a message to tell the user to enter text to textbox

Problem 2

Problem: If a user entered an invalid character in the encode message (not from A - Z), an error would occur as it would search for a position in the array that was out of bounds ($X < 0$ and $X > 26$)

Solution: Check if each character is in range before searching for code in array

Problem 3

Problem: If a user entered an invalid character (not 0 or 1), no message is given to tell the user this and the decode method would skip this character.

Solution: Check if each character is a 0 or 1 value before processing next tree navigation for each character

Problem 4

Problem: If a user enters a code string that is incomplete or a random string (meaning the last portion is only a partial code) the system won’t warn the user of the incorrect input and will return the decoded string, ignoring the end portion

Solution: Ensure that when the end of the encoded string has been reached in the decode algorithm that the symbol of the last node searched is a non-null value.

Testing

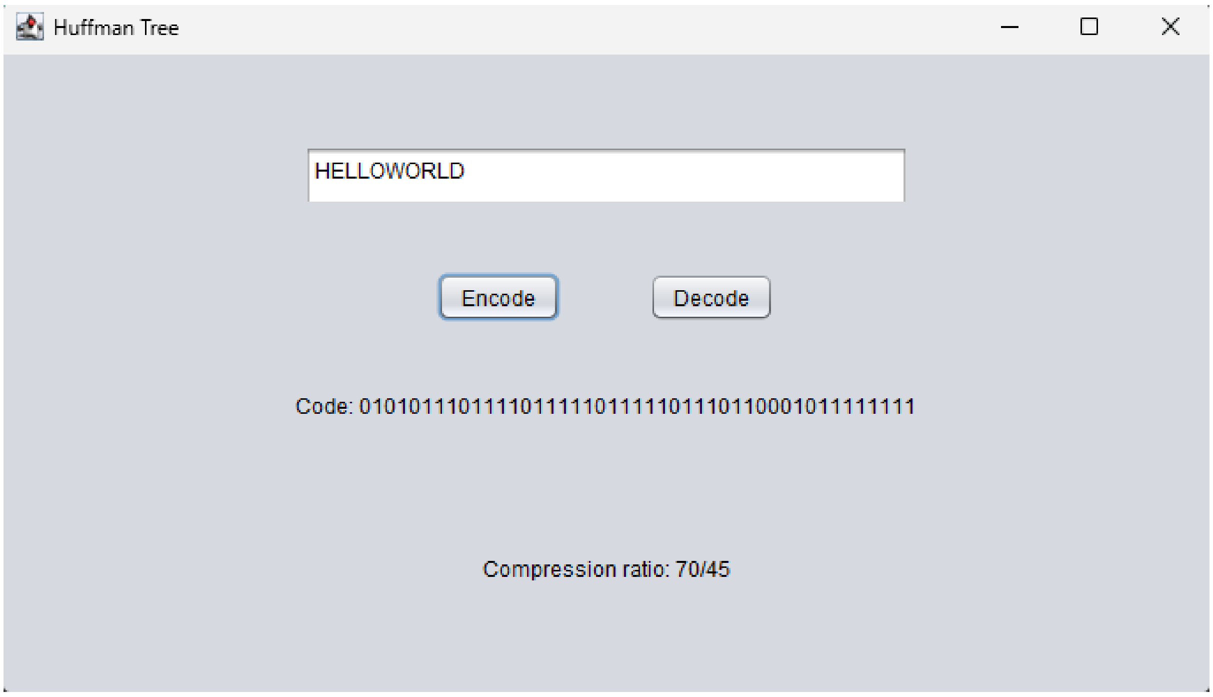
White Box Testing

Testing values for encoding:

Expected Inputs:
“HELLO”, “WORLD”, “HELLOWORLD”, “HUFFMAN”, “HUFFMANTREE”

Expected result: encoded string

Result: Expected



HELLOWORLD

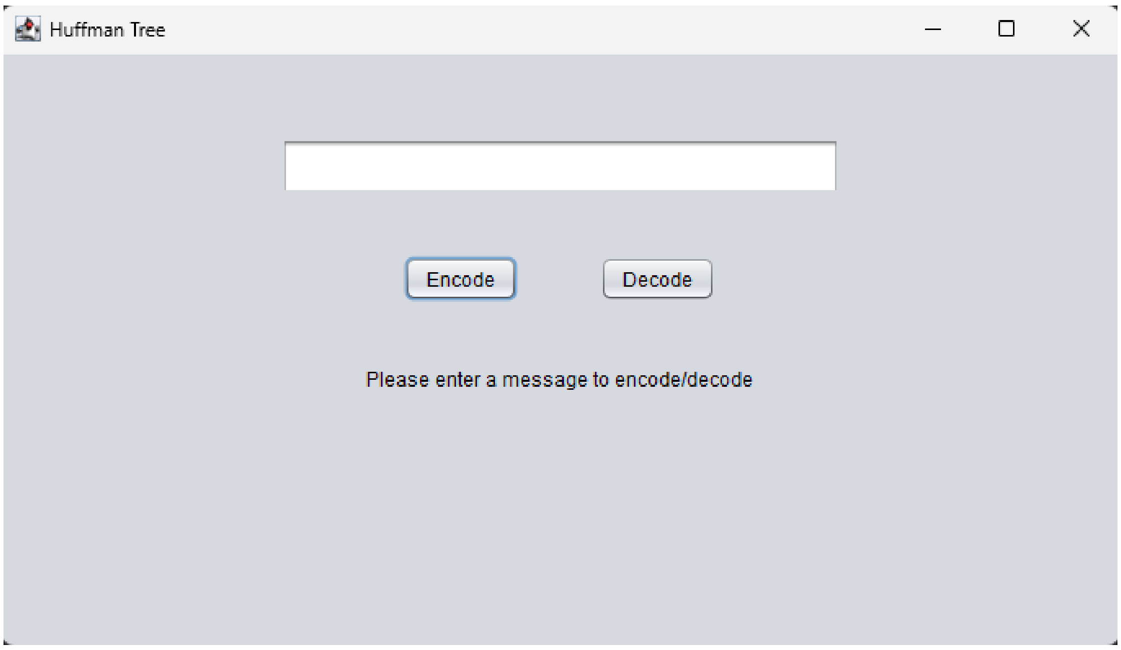
Code: 010101110111101111101111101110110001011111111

Compression ratio: 70/45

Incorrect Inputs:

“”

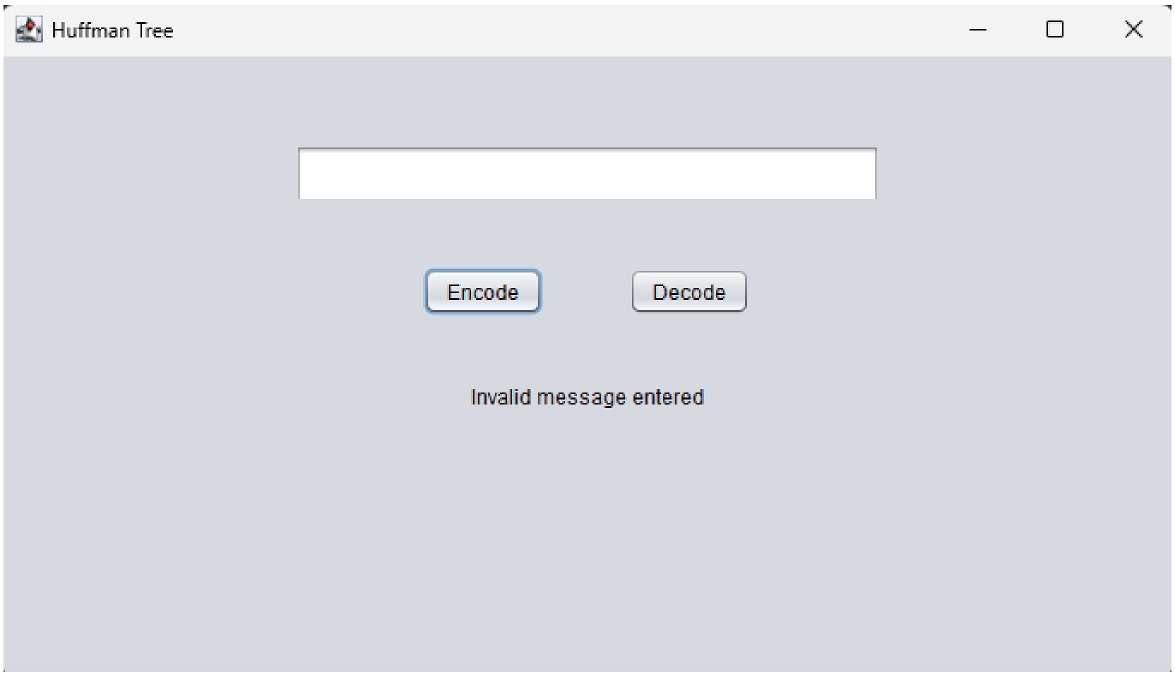
Expected result: Message to tell the user the input is incorrect
Initial result: “”
Result after fix: “Please enter a message to be encode/decoded”
See design problem 1



Please enter a message to encode/decode

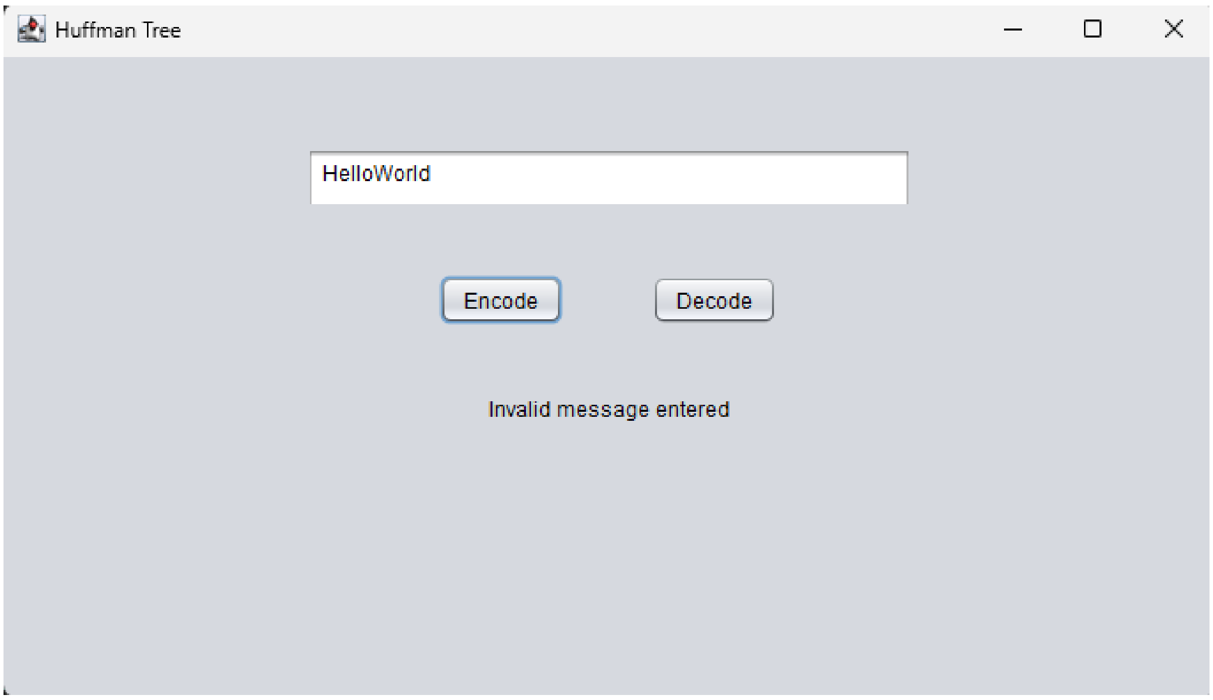
“ ”

Expected result: Message to tell the user the input is incorrect
Initial result: ERROR Array index out of bounds
Result after fix: “Invalid message entered, character not from A-Z”
See design problem 2



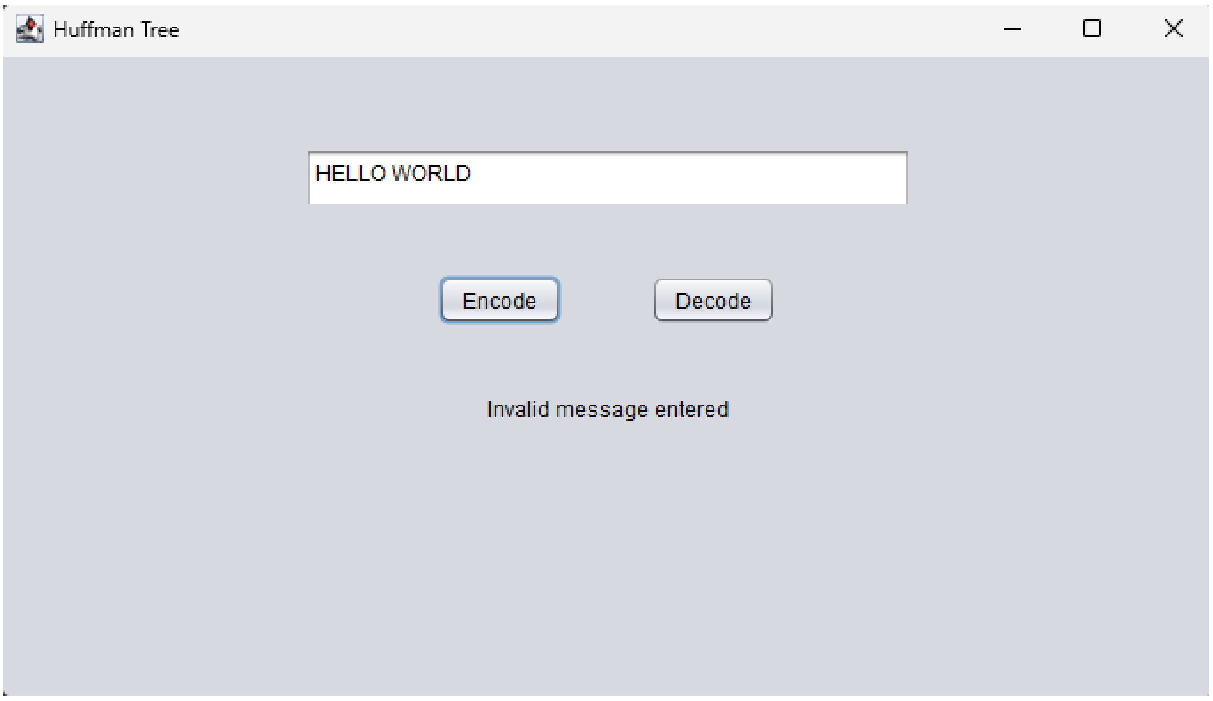
Invalid message entered

"HelloWorld"
Expected result: Message to tell the user the input is incorrect
Result: "Invalid message entered, character not from A-Z"



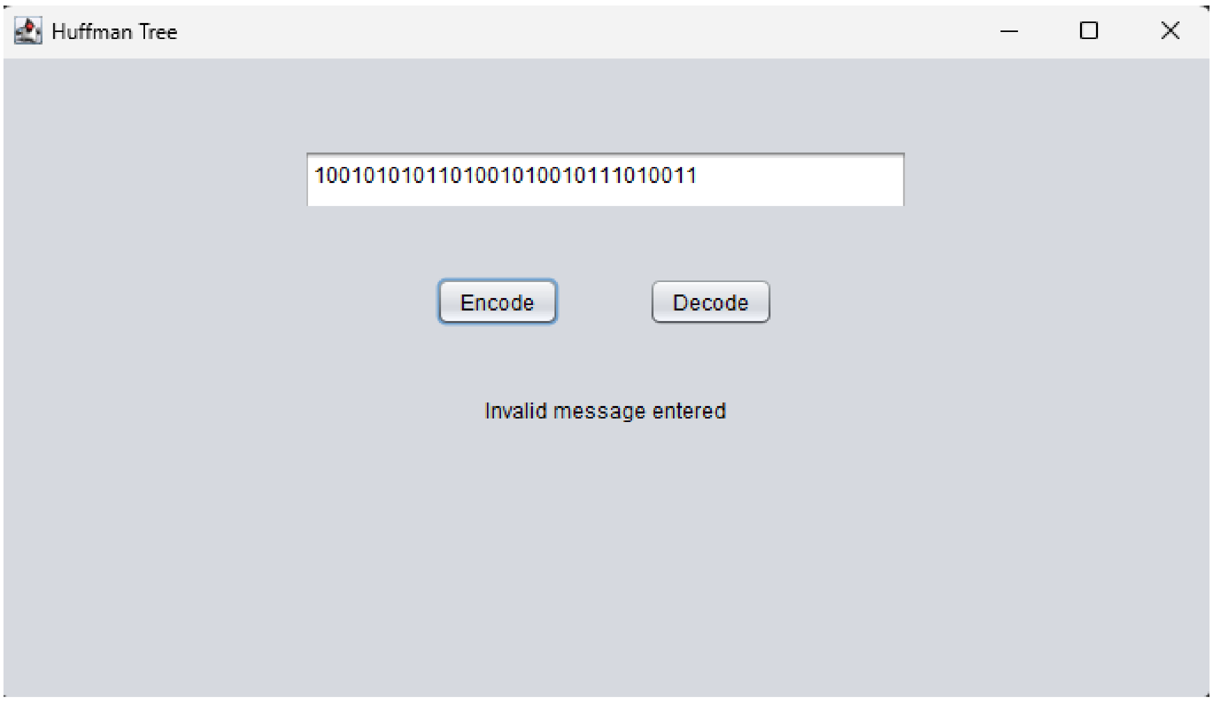
Invalid message entered

"HELLO WORLD"
Expected result: Message to tell the user the input is incorrect
Result: "Invalid message entered, character not from A-Z"



Invalid message entered

“10010101011010010100101110100 11”
Expected result: Message to tell the user the input is incorrect
Result: “Invalid message entered, character not from A-Z”

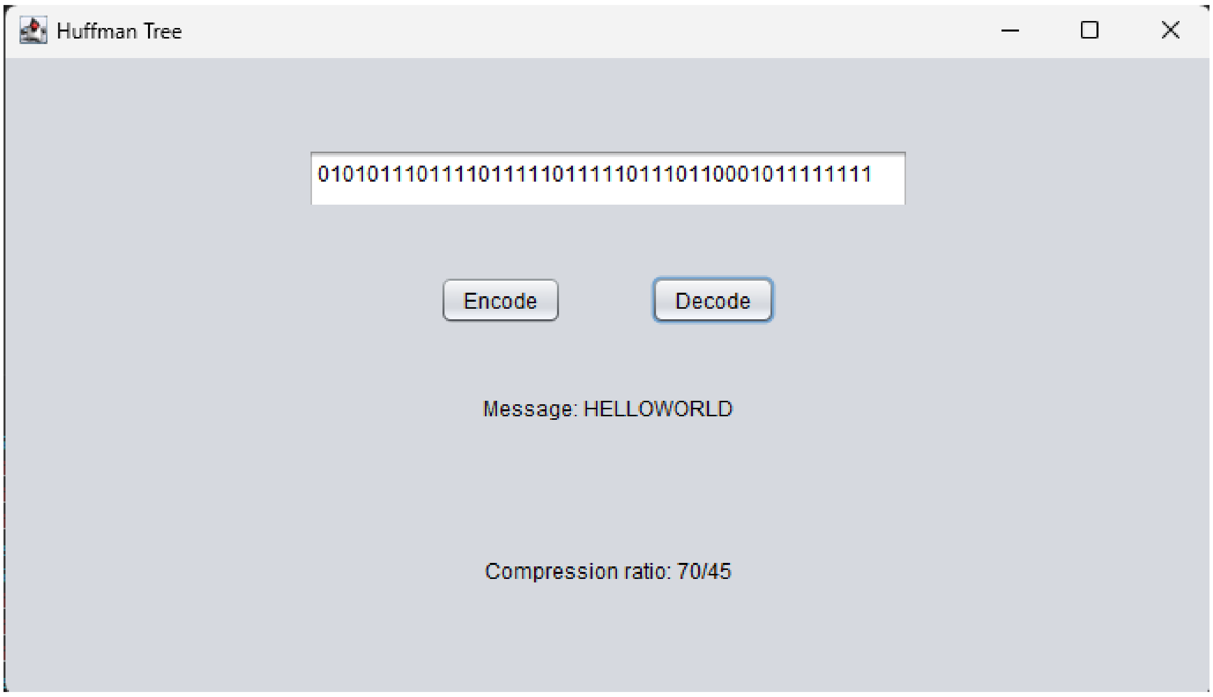


```
1001010101101001010010111010011
```

Invalid message entered

Testing values for decoding:

Expected Inputs:
“01010111011111011111101” (HELLO), “1111011101100010111111111” (WORLD),
“0101011101111101111110111110111011000101111111111” (HELLOWORLD),
“010101001001100011000111111101010” (HUFFMAN),
“01010100100110001100011111101010000010000011011” (HUFFMANTREE)
Expected result: decoded string
Result: Expected



```
0101011101111101111110111110111011000101111111111
```

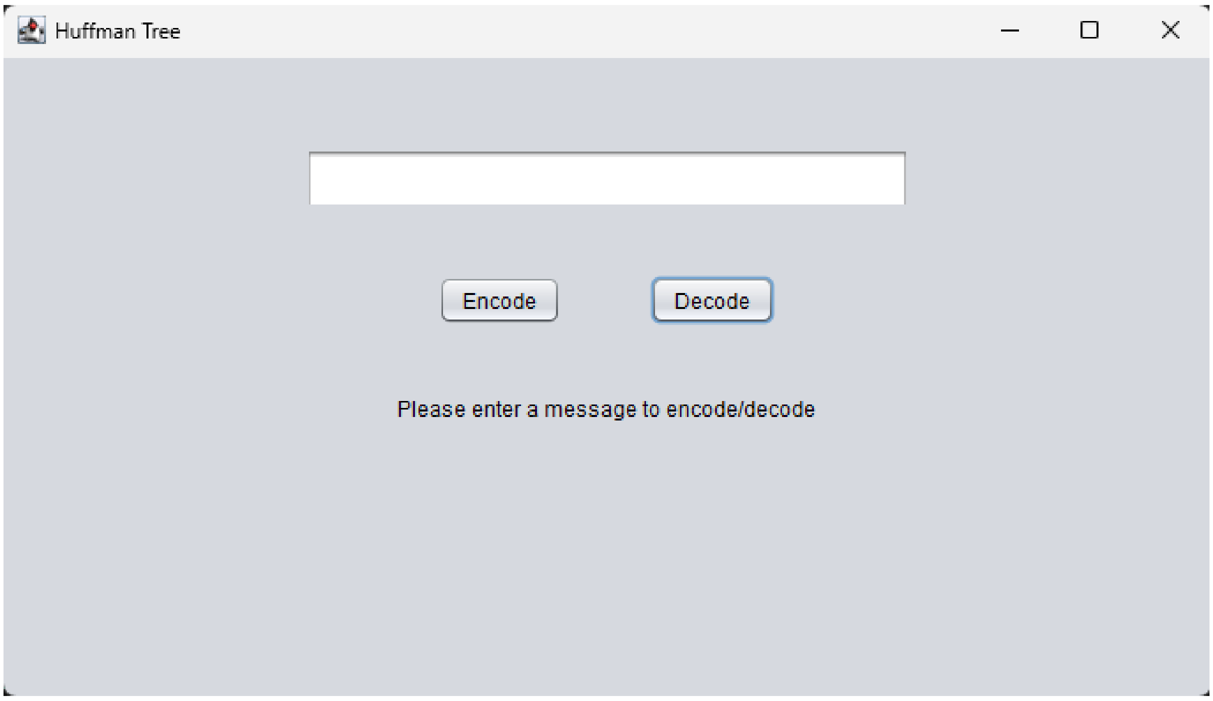
Message: HELLOWORLD

Compression ratio: 70/45

Incorrect Inputs:

“”

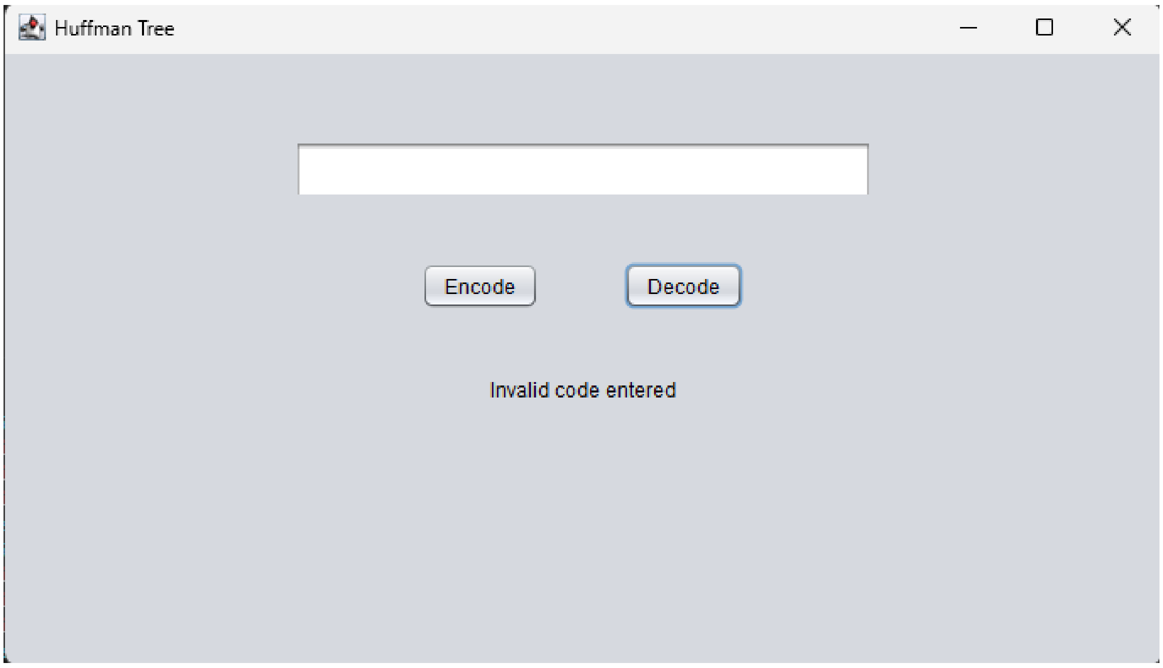
Expected result: Message to tell the user the input is incorrect
Result: “Please enter a message to be encode/decoded”



Please enter a message to encode/decode

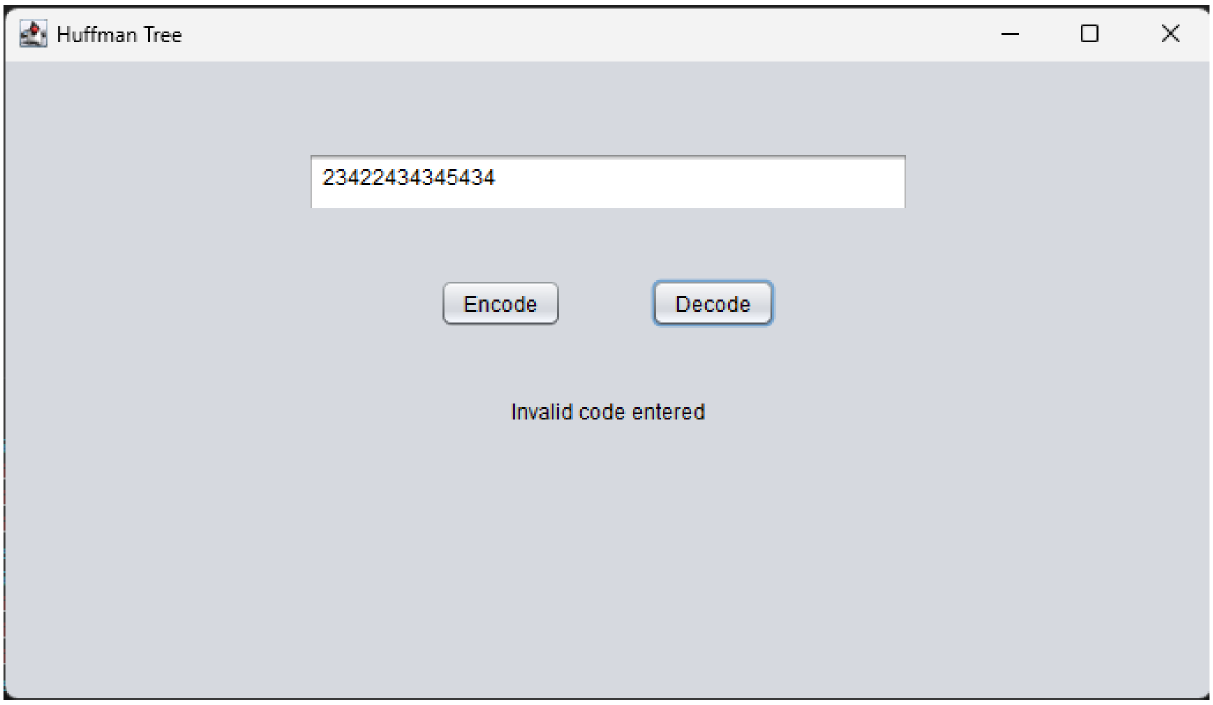
“ ”

Expected result: Message to tell the user the input is incorrect
Initial result: “”
Result after fix: “Invalid code entered”
See design problem 3



Invalid code entered

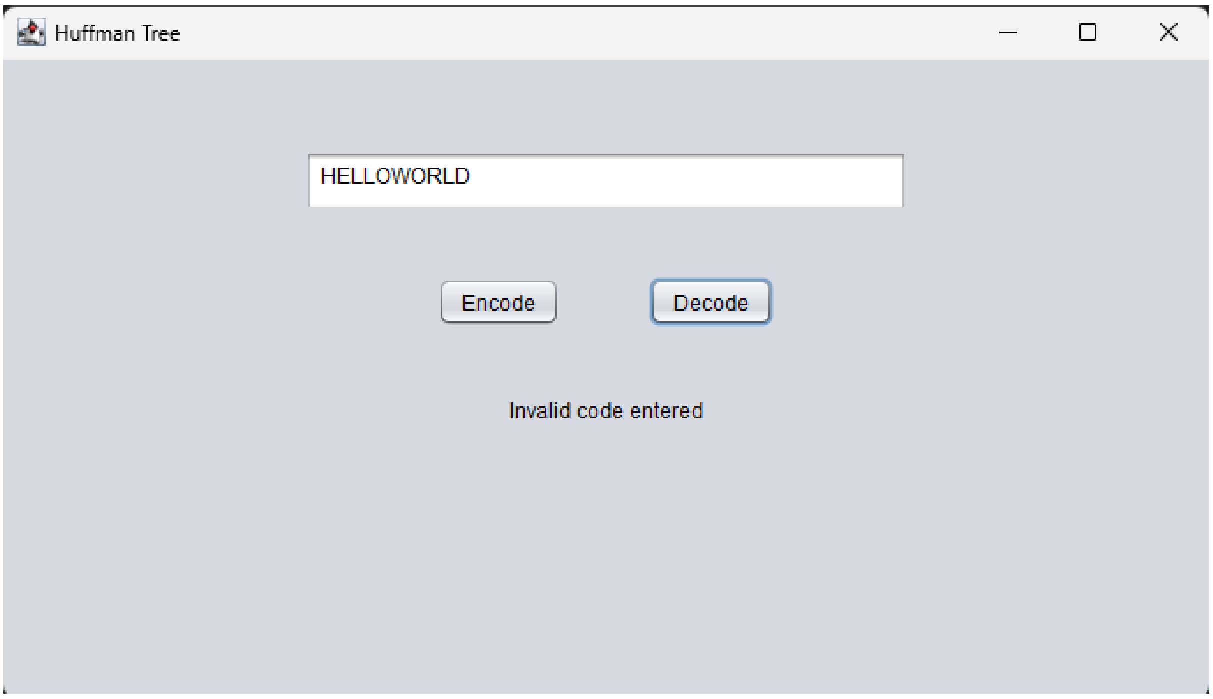
“23422434345434”
Expected result: Message to tell the user the input is incorrect
Result: “Invalid code entered”



23422434345434

Invalid code entered

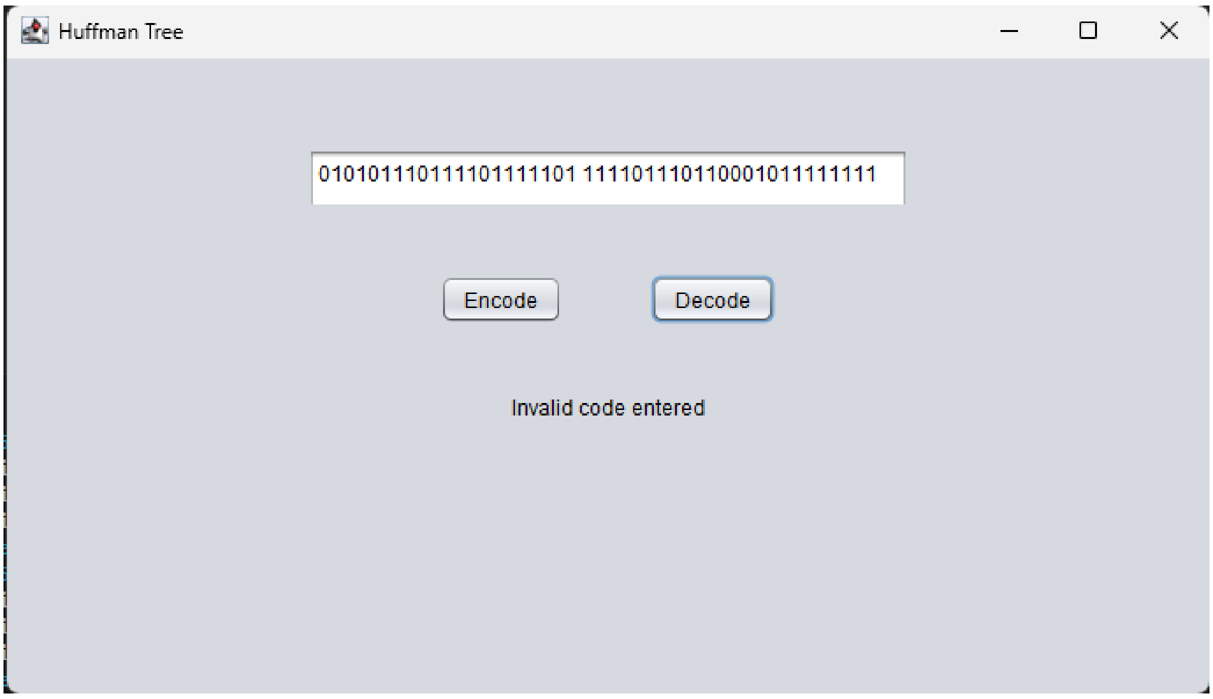
“HELLOWORLD”
Expected result: Message to tell the user the input is incorrect
Result: “Invalid code entered”



HELLOWORLD

Invalid code entered

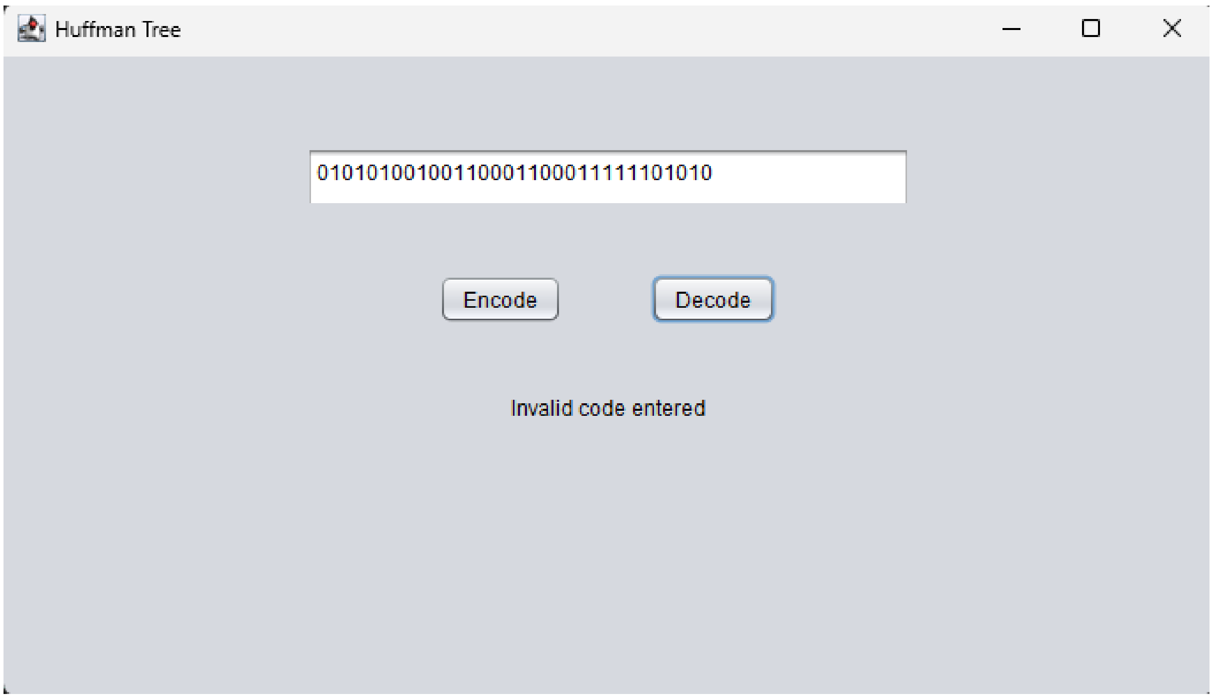
“01010111011111011111101 11110111011000101111111”(HELLO + “ ” + “WORLD”)
Expected result: Message to tell the user the input is incorrect
Result: “Invalid code entered”



```
01010111011111011111101 11110111011000101111111
```

Invalid code entered

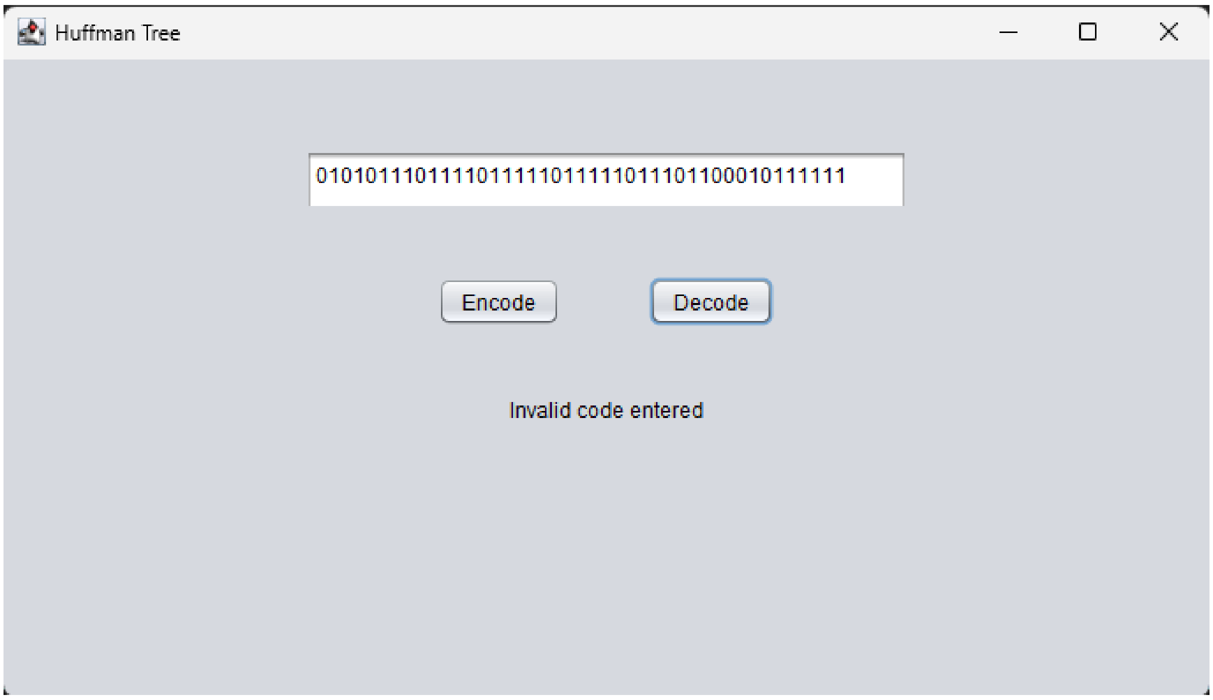
“01010100100110001100011111101010 ”(HUFFMAN + “ ”)
Expected result: Message to tell the user the input is incorrect
Result: “Invalid code entered”



```
01010100100110001100011111101010
```

Invalid code entered

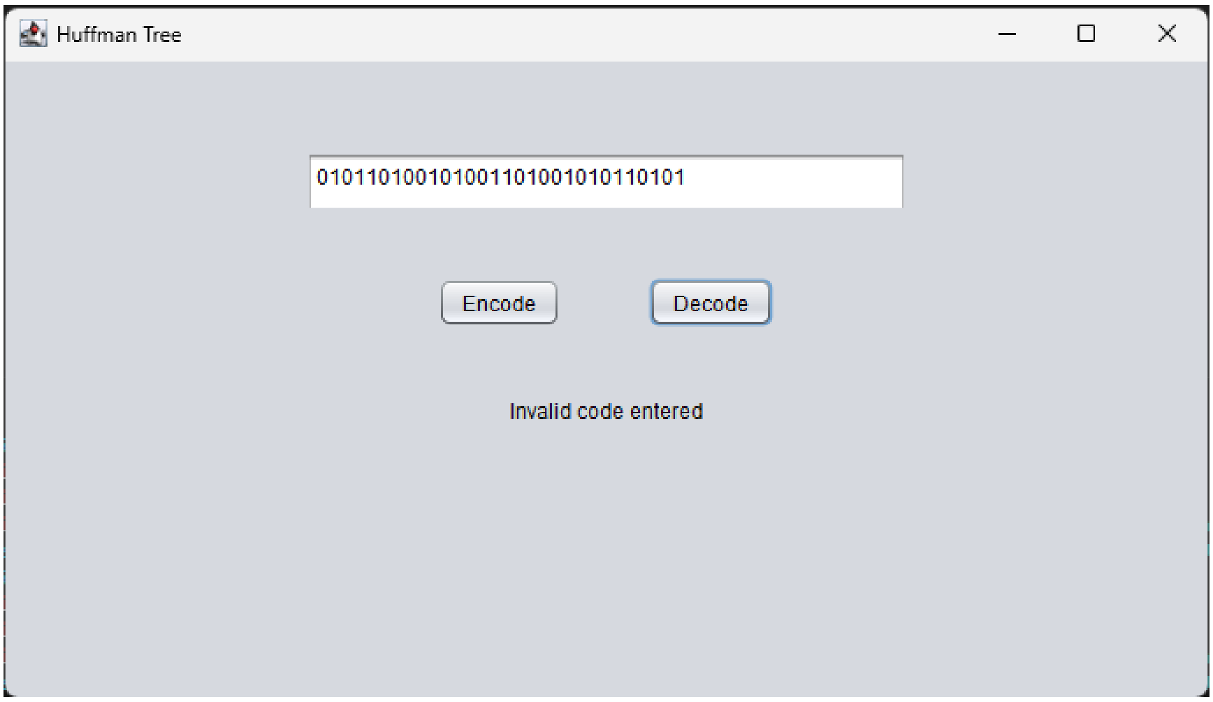
“010101110111101111011111011101100010111111”(HELLOWORLD without the last 2 digits of the encoded string)
Expected result: Message to tell the user the input is incorrect
Initial result: “HELLOWORL”
Result after fix: “Invalid code entered”
See design problem 4



010101110111101111011111011101100010111111

Invalid code entered

“0101101001010011010010101101 01” (Random string of numbers)
Expected result: Message to tell the user the input is incorrect
Result: “Invalid code entered”



010110100101001101001010110101

Invalid code entered