Mini Project Report

Mariam ELghandoor
202200886

Test cases

Unit Test:

```
Test 1 - Input: "abcccaaaa"
Apr 05, 2025 8:10:55 PM com.sun.javafx.application.PlatformImpl startup
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @670a5213'
the tree is:
null 9 100
null 4 98 0
null 1 96 00
null 0 94 000
b 1 95 001
c 3 97 01
a 5 99 1
the sequence is: 011000010011000100001100011101000101110
null 1 100
null 0 98 0
a 1 99 1
the tree is:
null 9 100
null 4 98 0
null 1 96 00
null 0 94 000
b 1 95 001
c 3 97 01
a 5 99 1
the sequence is: abcccaaaa
Original: abcccaaaa
Encoded:  011000010011000100001100011101000101110
Expected Encoded: 011000010011000100001100011101000101110
Decoded:  abcccaaaa
Expected Encoded: 011000010011000100001100011101000101110
***** Compression Test *****
Pass
***** Decompression Test *****
Pass
```

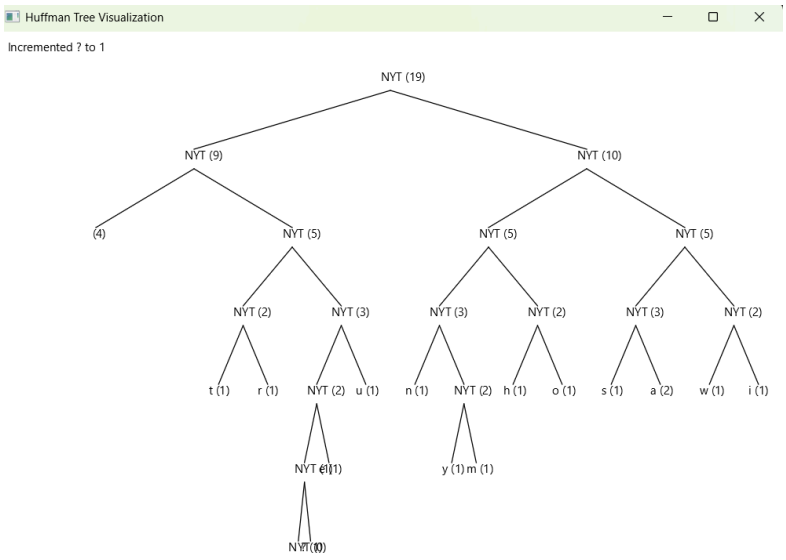
```
------------------------
Test 2 - Input: "aabb"
the tree is:
null 4 100
null 2 98 0
null 0 96 00
b 2 97 01
a 2 99 1
the sequence is: 01100001100110001001
null 1 100
null 0 98 0
a 1 99 1
the tree is:
null 4 100
null 2 98 0
null 0 96 00
b 2 97 01
a 2 99 1
the sequence is: aabb
Original: aabb
Encoded:  01100001100110001001
Expected Encoded: 01100001100110001001
Decoded:  aabb
Expected Encoded: 01100001100110001001
***** Compression Test *****
Pass
***** Decompression Test *****
Pass
```

```
------------------------
Test 3 - Input: "xyz"
the tree is:
null 3 100
x 1 98 0
null 2 99 1
null 1 96 10
null 0 94 100
z 1 95 101
y 1 97 11
the sequence is: 01111000001111001000111101O
null 1 100
null 0 98 0
x 1 99 1
the tree is:
null 3 100
x 1 98 0
null 2 99 1
null 1 96 10
null 0 94 100
z 1 95 101
y 1 97 11
the sequence is: xyz
Original: xyz
Encoded:  01111000001111001000111101O
Expected Encoded: 01111000001111001000111101O
Decoded:  xyz
Expected Encoded: 01111000001111001000111101O
***** Compression Test *****
Pass
***** Decompression Test *****
Pass
```

```
------------------------
Test 5 - Input: "aaaa"
the tree is:
null 4 100
null 0 98 0
a 4 99 1
the sequence is: 01100001111
null 1 100
null 0 98 0
a 1 99 1
the tree is:
null 4 100
null 0 98 0
a 4 99 1
the sequence is: aaaa
Original: aaaa
Encoded:  01100001111
Expected Encoded: 01100001111
Decoded:  aaaa
Expected Encoded: 01100001111
***** Compression Test *****
Pass
***** Decompression Test *****
Pass
------------------------
All Tests Passed
```

Encode and Decode dynamically:

TestCase 1: "What is your name ?"



Visualisation:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 1
Enter the string to encode: what is your name ?
Apr 04, 2025 8:04:43 PM com.sun.javafx.application.PlatformImpl startup
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @6f1140ab'
the tree is:
null 19 100
null 9 98 0
  4 94 00
null 5 95 01
null 2 88 010
t 1 76 0100
r 1 77 0101
null 3 89 011
null 2 78 0110
null 1 72 01100
null 0 70 011000
? 1 71 011001
e 1 73 01101
u 1 79 0111
null 10 99 1
null 5 96 10
null 3 90 100
n 1 80 1000
null 2 81 1001
y 1 74 10010
m 1 75 10011
null 2 91 101
h 1 82 1010
o 1 83 1011
null 5 97 11
null 3 92 110
s 1 84 1100
a 2 85 1101
null 2 93 111
w 1 86 1110
```

Encode:

```
the sequence is: 011101110011010000001100001100011101000000010000011000110100100000111001111011100011110011100000110111100000111010110100011100100010000011011001001000001101101011000110010100100100000111111
Encoded result: 011101110011010000001100001100011101000000010000011000110100100000111001111011100011110011100000110111100000111010110100011100100010000011011100100100000011011010110001100101001001000011111
```

Decode:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 2
Enter the encoded string to decode: 011101110011010000001100001100011101000000010000011000110100100000111001111011100011110011100000110111100000111010110100011100:
0010000011011100100100000011011010110001100101001000011111
Apr 04, 2025 8:08:23 PM com.sun.javafx.application.PlatformImpl startup
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @356ddf07'
null 1 100
null 0 98 0
w 1 99 1
the tree is:
null 19 100
null 9 98 0
  4 94 00
null 5 95 01
null 2 88 010
t 1 76 0100
r 1 77 0101
null 3 89 011
null 2 78 0110
null 1 72 01100
null 0 70 011000
? 1 71 011001
e 1 73 01101
u 1 79 0111
null 10 99 1
null 5 96 10
null 3 90 100
n 1 80 1000
null 2 81 1001
y 1 74 10010
m 1 75 10011
null 2 91 101
h 1 82 1010
o 1 83 1011
null 5 97 11
null 3 92 110
```

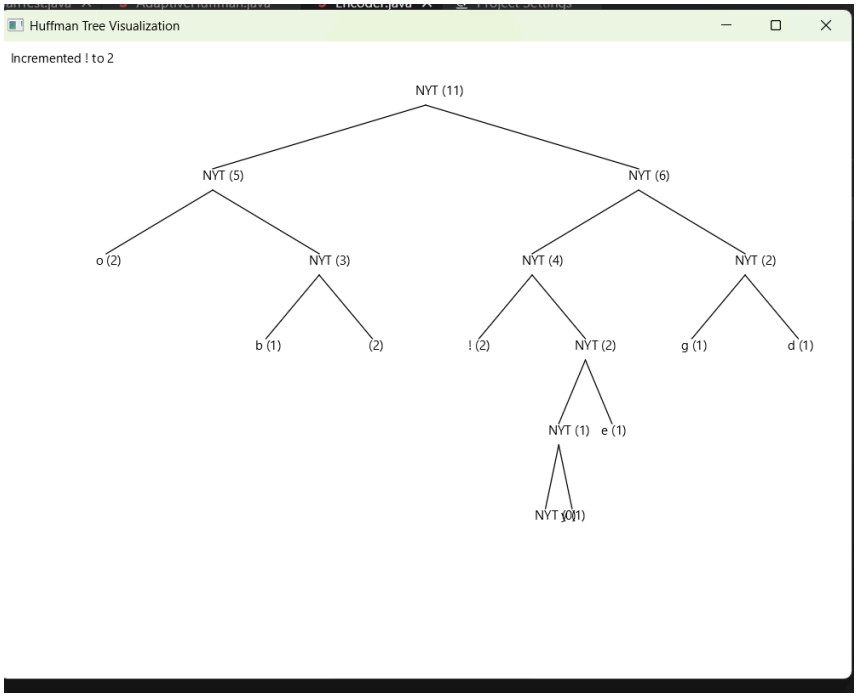
```
s 1 84 1100
a 2 85 1101
null 2 93 111
w 1 86 1110
i 1 87 1111
the sequence is: what is your name ?
Decoded result: what is your name ?

Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3):
```

Exit:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 3
Exiting program. Goodbye!
```

TestCase 2: "good bye !!"

Visualization:



compress:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 1
Enter the string to encode: good bye !!
Apr 04, 2025 8:12:29 PM com.sun.javafx.application.PlatformImpl startup
WARNING: Unsupported JavaFX configuration: classes were loaded from 'unnamed module @6f1140ab'
the tree is:
null 11 100
null 5 98 0
o 2 94 00
null 3 95 01
b 1 88 010
  2 89 011
null 6 99 1
null 4 96 10
! 2 90 100
null 2 91 101
null 1 86 1010
null 0 84 10100
y 1 85 10101
e 1 87 1011
null 2 97 11
g 1 92 110
d 1 93 111
the sequence is: 0110011100110111101000110010000000100000110001100010100001111001010001100101010000000100000101101
Encoded result: 0110011100110111101000110010000000100000110001100010100001111001010001100101010000000100000101101
```

decompress:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 2
Enter the encoded string to decode: 0110011100110111101000110010000000100000110001100010100001111001010001100101010000000100000101101
null 1 100
null 0 98 0
g 1 99 1
the tree is:
null 11 100
null 5 98 0
o 2 94 00
null 3 95 01
b 1 88 010
  2 89 011
null 6 99 1
null 4 96 10
! 2 90 100
null 2 91 101
null 1 86 1010
null 0 84 10100
y 1 85 10101
e 1 87 1011
null 2 97 11
g 1 92 110
d 1 93 111
the sequence is: good bye !!
Decoded result: good bye !!
```

Exit:

```
Adaptive Huffman Coding Menu:
1. Encode a sequence
2. Decode a sequence
3. Exit
Enter your choice (1-3): 3
Exiting program. Goodbye!
```

Code:

File : AdaptiveHuffman.java

```
import java.util.Scanner;

public class AdaptiveHuffman {
    public static void main(String[] args) throws Exception
{
        Scanner scanner = new Scanner(System.in);
        int root_value = 100;

        while (true) {
            System.out.println("\nAdaptive Huffman Coding
Menu:");
            System.out.println("1. Encode a sequence");
            System.out.println("2. Decode a sequence");
            System.out.println("3. Exit");
            System.out.print("Enter your choice (1-3): ");

            String choice = scanner.nextLine();

            switch (choice) {
                case "1": // Encode
                    System.out.print("Enter the string to
encode: ");
                    String inputToEncode =
scanner.nextLine();
                    String encoded =
Encoder.encodeSequence(inputToEncode, root_value);
                    System.out.println("Encoded result: " +
encoded);
                    break;

                case "2": // Decode
```

```
                    System.out.print("Enter the encoded
string to decode: ");
                    String inputToDecode =
scanner.nextLine();
                    String decoded =
Decoder.decode(inputToDecode, root_value);
                    System.out.println("Decoded result: " +
decoded);
                    break;

                case "3": // Exit
                    System.out.println("Exiting program.
Goodbye!");
                    scanner.close();
                    return;

                default:
                    System.out.println("Invalid choice!
Please enter 1, 2, or 3.");
                    break;
            }
        }

    }
}
```

File: Decoder.java

```
import java.util.Map;
import java.util.HashMap;

public class Decoder {
    public static String decode(String sequence, int
root_value) {
```

```
        if(sequence.length() == 0) return "";
        ///used as when decompress i have binary not
letters i want to find letters by their binary
        Map<String, Character> swappedMap = swapMap();
        /// initilaze root
        Node root = new Node(0, null);
        HuffmanTree.number_the_root(root, root_value);

        int index_of_binary = 8; // ASCII uses 8 bits
        int binary_length_index = 1;
        StringBuilder symbol_list = new StringBuilder();
        boolean found = false;
        /// first letter doesnt apply the nyt first rule
        Character letter =
swappedMap.get(sequence.substring(0, 8));
        Node new_node_start = new Node(0, letter);
        HuffmanTree.assign_right_and_NYT(root,
new_node_start);
        HuffmanTree.assign_binary(root);
        HuffmanTree.assign_number(root);

HuffmanTree.trace_and_swap_then_increment(new_node_start,
root);
        symbol_list.append(letter);

        HuffmanTree.print_tree(root);
        try {
            Thread.sleep(1000); // Pause after first step
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        while (index_of_binary < sequence.length()) {
```

```
            //// search up to 8 values infront of me
            while(binary_length_index < 8 &&
binary_length_index <= sequence.length() -
index_of_binary){
                Node node =
HuffmanTree.find_symbol_using_binary(root,
sequence.substring(index_of_binary, index_of_binary +
binary_length_index));

                if(node != null &&
HuffmanTree.is_leaf(node)){
                    if(HuffmanTree.isNYT(node)){
                        String temp =
sequence.substring(index_of_binary + binary_length_index,
index_of_binary + binary_length_index + 8);
                        letter = swappedMap.get(temp);
                        // System.out.println();
                        Node new_node = new Node(0,
letter);

HuffmanTree.assign_right_and_NYT(node, new_node);
                        HuffmanTree.assign_binary(root);
                        HuffmanTree.assign_number(root);

HuffmanTree.trace_and_swap_then_increment(new_node, root);
                        symbol_list.append(letter);
                        index_of_binary +=
(binary_length_index + 8);
                    }else{
                        // System.out.println("found: " +
node.symbol);
                        symbol_list.append(node.symbol);

HuffmanTree.trace_and_swap_then_increment(node, root);
```

```
                        index_of_binary +=
binary_length_index;
                    }
                    found = true;
                    break;
                }else binary_length_index++;
            }
            if(!found){
                index_of_binary++;
                System.out.println("error: no symbol
found");
            }
            binary_length_index = 1;
            found = false;
            try {
                Thread.sleep(1000); // Pause after each
step
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("the tree is: ");
        HuffmanTree.print_tree(root);
        System.out.println("the sequence is: " +
symbol_list);
        HuffmanTreeVisualizer.close();
        return symbol_list.toString();
    }

    /// Create swapped ASCII map
    private static Map<String, Character> swapMap() {
        Map<String, Character> swappedMap = new
HashMap<>();
```

```
        for (int i = 0; i < 128; i++) {
            String binary = String.format("%8s",
Integer.toBinaryString(i)).replace(' ', '0');
            swappedMap.put(binary, (char)i);
        }
        return swappedMap;
    }
}
```

File: Encoder.java

```
import java.util.Map;
import java.util.HashMap;

public class Encoder {
    /// search for letter
    public static void encode(Character letter, Node root,
Map<Character, String> map, StringBuilder sequence) {
        Node node = HuffmanTree.find_node(root, letter);
        if (node == null) { // not found create it
            // System.out.println("not found");
            /// find nyt to expand from it
            Node nyt_node = HuffmanTree.find_NYT(root);
            if (nyt_node == null)///avoid
nullpointerexception if no tree yet
                nyt_node = root;
            if (sequence.length() != 0)// first letter no
nyt before it
                sequence.append(nyt_node.binary_code);
            ///append letter then construct tree
            sequence.append(map.get(letter));
            Node new_node = new Node(0, letter);
            HuffmanTree.assign_right_and_NYT(nyt_node,
new_node);
            HuffmanTree.assign_binary(root);
```

```
            HuffmanTree.assign_number(root);

HuffmanTree.trace_and_swap_then_increment(new_node, root);
        } else { /// i already have the letter just
increment and check swap conditions
            sequence.append(node.binary_code);
            HuffmanTree.trace_and_swap_then_increment(node,
root);
        }
    }

    /// Convert string to character array
    private static char[] splitStringToSymbols(String
input) {
        return input.toCharArray();
    }

    /// take the string and compress using ASCII codes
    public static String encodeSequence(String input, int
root_value) {
        StringBuilder sequence = new StringBuilder();
        /// initialize root then start
        Node root = new Node(0, null);
        HuffmanTree.number_the_root(root, root_value);
        HuffmanTreeVisualizer.visualize(root, "Initial NYT
node", root);
        // Create ASCII mapping
        Map<Character, String> asciiMap = new HashMap<>();
        for (int i = 0; i < 128; i++) {
            String binary = String.format("%8s",
Integer.toBinaryString(i)).replace(' ', '0');
            asciiMap.put((char)i, binary);
        }
```

```
        char[] symbols = splitStringToSymbols(input);
        for (char c : symbols) {
            // System.out.println("Inserting: " + c);
            encode(c, root, asciiMap, sequence);
            try {
                Thread.sleep(1000); // Pause 1 second to
observe each step
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        System.out.println("the tree is: ");
        HuffmanTree.print_tree(root);
        System.out.println("the sequence is: " + sequence);

        return sequence.toString();
    }
}
```

File: Huffman_tree.java

```
import java.util.*;

////// notes for me to remember what i was doing
public class HuffmanTree {
    ///// da el main function that does most of the alg
    public static void trace_and_swap_then_increment(Node
node, Node root) {
        if (node == null) {
            return;
        }
        // print_tree(root);
        // System.out.println("----------------------");
```

```
        if (node.value == 0) {///// if i am a new node only
increment and go to my parent
            node.value += 1;
            // if (node != root)
            //     swap_operation(node.parent, root);
        } else { ////// else swap first then increment
            if (node != root)
                swap_operation(node, root);
            node.value += 1;
        }
        // System.out.println("Incremented value of " +
node.symbol + " to " + node.value);
        // System.out.println("----------------------");
        if (node.parent != null) { // wa recursevly do that
again to my parent
            trace_and_swap_then_increment(node.parent,
root);
        }
        String message = "Incremented " + (node.symbol !=
null ? node.symbol : "NYT") + " to " + node.value;
        HuffmanTreeVisualizer.visualize(root, message,
node);// Visualize after increment
    }


    ///// dy swaps node if after seaching the whole tree i
found the conditions satisfied
    public static void swap_operation(Node node, Node root)
{
        //// search
        Node swapCandidate =
find_nearest_swap_candidate(root, node, null, 0);
        if (swapCandidate != null && swapCandidate.parent
!= null) {
            swap_nodes(node, swapCandidate);
```

```
            ////re assign the values
            assign_number(root);
            assign_binary(root);
            // System.out.println("Swapped " + node.symbol
+ " with " + swapCandidate.symbol);
            // Visualize after swap
            String message = "Swapped " + (node.symbol !=
null ? node.symbol : "NYT") + " with " +
                                (swapCandidate.symbol != null ?
swapCandidate.symbol : "NYT");
            HuffmanTreeVisualizer.visualize(root, message,
node);
        }
    }

    ////traversing the tree dfs pioratizing the nearst to
the root == smallest depth
    private static Node find_nearest_swap_candidate(Node
current, Node target, Node bestCandidate, int depth) {
        if (target.parent == null || current == null) {
            return bestCandidate;
        }
        if (current.value == target.value && current.number
> target.number && target.parent != current) {
            bestCandidate = current;
        }
        bestCandidate =
find_nearest_swap_candidate(current.right, target,
bestCandidate, depth + 1);
        bestCandidate =
find_nearest_swap_candidate(current.left, target,
bestCandidate, depth + 1);
        return bestCandidate;
    }
```

```
    ///// swap the the node parests and if the other is
left child i become left  child
    /// i dont swap children its a whole complete tree swap
    private static void swap_nodes(Node a, Node b) {
        if (a == null || b == null || a == b)
            return;
        Node aParent = a.parent;
        Node bParent = b.parent;
        boolean aIsLeft = aParent != null && aParent.left
== a;
        boolean bIsLeft = bParent != null && bParent.left
== b;
        if (aParent != null) {
            if (aIsLeft)
                aParent.left = b;
            else
                aParent.right = b;
        }
        if (bParent != null) {
            if (bIsLeft)
                bParent.left = a;
            else
                bParent.right = a;
        }
        a.parent = bParent;
        b.parent = aParent;
    }

    //// this initializes the root vars
    public static void number_the_root(Node root, int
value) {
        root.number = value;
```

```
        root.binary_code = "";
    }

    //// traverse the whole tree dfs and assign the binary
code
    public static void assign_binary(Node root) {
        if (root.left == null && root.right == null) {
            return;
        }
        root.left.binary_code = root.binary_code + "0";
        root.right.binary_code = root.binary_code + "1";
        assign_binary(root.left);
        assign_binary(root.right);
    }

    //// traverse the whole tree bfs and assign the numbers
    public static void assign_number(Node root) {
        if (root.left == null) {
            return;
        }
        Queue<Node> queue = new LinkedList<Node>();
        if (root.right != null)
            queue.add(root.right);
        if (root.left != null)
            queue.add(root.left);
        int count = root.number - 1;
        while (!queue.isEmpty()) {
            Node node = queue.poll();
            node.number = count--;
            if (node.right != null)
                queue.add(node.right);
            if (node.left != null)
                queue.add(node.left);
```

```
        }
    }

    //// dy bnadyha when i get a new letter so that i can
expand the tree from nyt
    /// and then assign the right values and assign parents
and childs
    public static void assign_right_and_NYT(Node root, Node
right) {
        root.right = right;
        right.parent = root;
        Node nyt_node = new Node(0, null);
        root.left = nyt_node;
        nyt_node.parent = root;
        String message = "Inserted symbol: " +
(right.symbol != null ? right.symbol : "NYT");
        HuffmanTreeVisualizer.visualize(root, message,
right); // Visualize after adding new node
    }

    ///// check if nyt
    public static boolean isNYT(Node root) {
        if (root == null)
            return false;
        return root.symbol == null && root.parent != null
&& root.value == 0;
    }

    ///b7awl ala2y dfs if the letter was seen before or not
if yes i return the node to increament on it
    public static Node find_node(Node root, Character
symbol) {
        if (root == null) {
            return null;
```

```
        }
        if (root.symbol == symbol) {
            return root;
        }
        Node left = find_node(root.left, symbol);
        Node right = find_node(root.right, symbol);
        if (left != null) {
            return left;
        }
        if (right != null) {
            return right;
        }
        return null;
    }

    /// this is used to be able to expand from nyt so i
search for it dfs
    public static Node find_NYT(Node root) {
        if (root == null) {
            return null;
        }
        if (isNYT(root)) {
            return root;
        }
        Node left = find_NYT(root.left);
        Node right = find_NYT(root.right);
        if (left != null) {
            return left;
        }
        if (right != null) {
            return right;
        }
        return null;
```

```
    }

    ///// when decreapting i search the tree with the
binaary code i have from string
    public static Node find_symbol_using_binary(Node node,
String binary) {
        if (node == null) {
            return null;
        }
        if (binary.equals(node.binary_code)) {
            return node;
        }
        Node leftResult =
find_symbol_using_binary(node.left, binary);
        return (leftResult != null) ? leftResult :
find_symbol_using_binary(node.right, binary);
    }

    /// to check if the node i found actualy an nyt or
letter not a null node
    /// i use it when searching with binary to not take a
parent node and mistake it for letter
    public static boolean is_leaf(Node root) {
        return root.left == null && root.right == null;
    }

    ///// just to trace
    public static void print_tree(Node root) {
        if (root == null) {
            return;
        }
        System.out.println(root.symbol + " " + root.value +
" " + root.number + " " + root.binary_code);
        print_tree(root.left);
```

```
        print_tree(root.right);
    }
}
```

File: node.java

```
public class Node {
    Integer value;
    Node left;
    Node right;
    Character symbol;
    String binary_code;
    Integer number;
    Node parent;

    Node(int value, Character symbol) {
        this.symbol = symbol;
        this.value = value;
        left = right = null;
        binary_code = "";
        number = null;
        parent = null;
    }

    Node(Node left, Node right) {
        this.left = left;
        this.right = right;
        this.value = left.value + right.value;
        binary_code = null;
        number = null;
        parent = null;
    }
}
```

File: AdavptiveHuffmanTest.java

```
public class AdaptiveHuffmanTest {
    static class TestCase {
        String input;
        String expectedEncoded;
        int rootValue;

        TestCase(String input, String expectedEncoded, int
rootValue) {
            this.input = input;
            this.expectedEncoded = expectedEncoded;
            this.rootValue = rootValue;
        }
    }

    public static void main(String[] args) {
        TestCase[] testCases = {
            new TestCase("abcccaaaa",
"011000010011000100001100011101000101110", 100),
            new TestCase("aabb", "01100001100110001001",
100),
            new TestCase("xyz",
"011110000011110010001111010", 100),
            new TestCase("", "", 100),
            new TestCase("aaaa", "01100001111", 100)
        };

        boolean allTestsPassed = true;

        for (int i = 0; i < testCases.length; i++) {
            TestCase test = testCases[i];
            System.out.println("Test " + (i + 1) + " -
Input: \"" + test.input + "\"");
```

```
            String encoded =
Encoder.encodeSequence(test.input, test.rootValue);
            String decoded = Decoder.decode(encoded,
test.rootValue);

            System.out.println("Original: " + test.input);
            System.out.println("Encoded:  " + encoded);
            System.out.println("Expected Encoded: " +
test.expectedEncoded);
            System.out.println("Decoded:  " + decoded);

            System.out.println("Expected Encoded: " +
encoded);
            boolean encodePass =
test.expectedEncoded.equals(encoded);
            System.out.println("***** Compression Test
*****");
            System.out.println(encodePass ? "Pass" :
"Fail");

            boolean decodePass =
test.input.equals(decoded);
            System.out.println("***** Decompression Test
*****");
            System.out.println(decodePass ? "Pass" :
"Fail");

            if (!encodePass || !decodePass) {
                allTestsPassed = false;
            }

            System.out.println("------------------------");
        }
```

```
        System.out.println("All Tests " + (allTestsPassed ?
"Passed" : "Failed"));

        HuffmanTreeVisualizer.close();
    }
}
```

File:HuffmanTreeVisulizer.java

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.shape.Line;
import javafx.scene.text.Text;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import java.util.HashMap;
import java.util.Map;

public class HuffmanTreeVisualizer extends Application {
    private static Node currentRoot;
    private static Pane pane = new Pane();
    private static Stage primaryStage;
    private static Text statusText = new Text(10, 20, "");
    // private static final double NODE_RADIUS = 20;
    private static final double VERTICAL_GAP = 80;
    // private static final double HORIZONTAL_GAP = 60;
    private static Node lastModifiedNode; // Track the last
modified node
```

```
    // update the visualization with a new tree root and
message
    public static void visualize(Node root, String message,
Node modifiedNode) {
        currentRoot = root;
        lastModifiedNode = modifiedNode; // Highlight this
node
        if (!Platform.isFxApplicationThread()) {
            if (primaryStage == null) {
                new Thread(() ->
launch(HuffmanTreeVisualizer.class)).start();
                try {
                    Thread.sleep(500);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
            Platform.runLater(() -> updateTree(message));
        } else {
            updateTree(message);
        }
    }

    //  close the visualization
    public static void close() {
        Platform.runLater(() -> {
            if (primaryStage != null) {
                primaryStage.close();
                Platform.exit();
            }
        });
    }
```

```
    @Override
    public void start(Stage stage) {
        primaryStage = stage;
        pane.getChildren().add(statusText);
        Scene scene = new Scene(pane, 800, 600);
        primaryStage.setTitle("Huffman Tree
Visualization");
        primaryStage.setScene(scene);
        primaryStage.show();

        updateTree("Initializing tree");
    }

    // Update the tree visualization with message
    private static void updateTree(String message) {
        pane.getChildren().clear();
        pane.getChildren().add(statusText); // Re-add
status text
        statusText.setText(message); // Update status
message
        if (currentRoot != null) {
            drawTree(currentRoot, 400, 50, 200, new
HashMap<>());
        }
    }

    // Recursively draw the tree with highlighting
    private static void drawTree(Node node, double x,
double y, double xOffset, Map<Node, Double> xPositions) {
        if (node == null) return;

        String label = node.symbol != null ? node.symbol +
" (" + node.value + ")" : "NYT (" + node.value + ")";
        Text text = new Text(x - 10, y, label);
```

```
        // Highlight if this is the last modified node
        if (node == lastModifiedNode) {
            text.setFill(Color.RED); // Highlight in red
            // Revert to black after a delay
            new Thread(() -> {
                try {
                    Thread.sleep(800); // Highlight for 0.8
seconds
                    Platform.runLater(() ->
text.setFill(Color.BLACK));
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }).start();
        } else {
            text.setFill(Color.BLACK);
        }

        pane.getChildren().add(text);
        xPositions.put(node, x);

        if (node.left != null) {
            double leftX = x - xOffset;
            double childY = y + VERTICAL_GAP;
            Line line = new Line(x, y + 10, leftX, childY -
10);
            pane.getChildren().add(line);
            drawTree(node.left, leftX, childY, xOffset / 2,
xPositions);
        }
        if (node.right != null) {
            double rightX = x + xOffset;
```

```
            double childY = y + VERTICAL_GAP;
            Line line = new Line(x, y + 10, rightX, childY
- 10);
            pane.getChildren().add(line);
            drawTree(node.right, rightX, childY, xOffset /
2, xPositions);
        }
    }
}
```