ISO 9001:2008 Certified Institute

Java Institute for Advanced Technology

Department of Examinations

COURSE(S) – (LEADING TO)	BIRMINGHAM CITY BSC (HONS) SE - TOP-UP
SUBMISSION DATE	
UNIT NAME	DATA STRUCTURES AND ALGORITHMS
UNIT ID	JIAT/DSA
EXAMINATION ID	JIAT/DSA/EX 01
BRANCH	JAVA INSTITUTE, COLOMBO

NAME	: M.R.P.N.THARUKSHA RAJAPAKSHA (BLOCK CAPITALS)
BCU STUDENT ID	: 22178965
NIC NO	: 200019401866
SCN NO	: 207977608

Acknowledgments

First and foremost, I would like to express my deep and sincere gratitude to my lecturer and advisor for the Data Structures and Algorithms subject, Mr. Vishwa Nuwantha for the continuous support, motivation, and knowledge he has given to me to complete this research.

I am incredibly grateful to my parent for their love, care, and sacrifices they made to build a better future for me. And the encouragement they have given me to complete this research.

I want to thank all of my friends for their support and the sleepless nights we were working together before deadlines.

I thank the management of Java Institute for their support to do this project.

Finally, my thanks go to all the people who have helped me to complete this project directly or indirectly.

Abstract

This Research Document contains about the algorithms such as what is an algorithm, what features are required to be considered an algorithm, properties of the algorithm, factors that determine the complexity of an algorithm, types of categories divided by the complexity of the algorithm, and types of algorithms.

Furthermore, this research document discusses about the given exam scenario, Huffman Coding, Huffman Tree, Huffman Tree for the given scenario, Huffman Encoding, Huffman Decoding, and Java code solution for the given scenario.

Table of Contents

1. What is an Algorithm?

“An algorithm is a well-defined set of instructions for solving a problem.” It accepts a collection of inputs and outputs the desired result.

For example, an algorithm to multiply two numbers and print the result:

Step 1. “Start”

Step 2. Declare the a,b & c variables. (a & b for assigning numbers and c for holding the result.)

Step 3. Input two numbers.

Step 4. Assign the two numbers to the a & b variables.

Step 5. Multiply the a & b variables, using the * operator, and assign the result to the c variable. (c = a*b)

Step 6. Print the c variable.

Step 7. “End”

Most published computer instructions aren’t algorithms, though. To be considered an algorithm, instructions must include the following characteristics:

- Language Independence: The intended algorithm should be language-independent, in other words, it should be only simple instructions that can be executed in any language and still provide the desired results.
- Well-Defined Inputs: If an algorithm needs inputs, the inputs must be well-defined. It may or may not require inputs.
- Well-Defined Outputs: What output the algorithm will be produced should be clearly stated and well-defined. It should generate at least one output.
- Straightforward and Unambiguous: Algorithms should be unambiguous and clear. Each of its stages should be clear in every way and lead to only one meaning.
- Feasible: The algorithm should be simple, general, and practical enough to be implemented with the given resources. It should not include future technologies or anything else.
- Finiteness: The algorithm must terminate after a certain amount of time.

Properties of the Algorithm:

- Each step of the algorithm must be effective, that is, each step must perform some task.
- It should accept 0 or more input.
- It should generate at least one output.
- It should be deterministic, meaning it should provide the same result for every input instance.
- Algorithm Properties It should finish after a limited time.

The space and time an algorithm uses determine its complexity. Consequently, the complexity of an algorithm is related to the time it takes to execute it and produce the desired results, as well as the amount of space required to store all the data (input, temporary data, and output).

- Space Factor: Calculates the maximum memory space required by the algorithm to run/execute.
- Time Factor: Time is determined by measuring the number of essential operations in the sorting process such as comparisons.

As a result, the complexity of the algorithm can be classified into two categories:

1. Space Complexity

   The amount of memory required by an algorithm to store the variables and get the result is called its space complexity. This applies to inputs, temporary processes, and outputs.

   The space complexity of an algorithm is calculated by identifying the following two components:

   - Fixed Part: This refers to the space that the algorithm absolutely needs. E.g. program size, input variables, output variables, etc.
   - Variable Part: This refers to the space that may change depending on how the algorithm is executed. E.g. temporary variables, recursive stack space, dynamic memory allocation, etc.

The mathematical equation for the space complexity (S(P)) of any algorithm P is:

S(P) = C + SP(I)

C: Fixed Part

SP(I): Variable part of the algorithm that relies on instance characteristic I.

2. Time Complexity

   The time required to execute an algorithm and generate a result is called its time complexity. This can be used for standard operations, conditional if-else statements, loop statements, and so on.

   The time complexity of an algorithm is also measured by finding the following two components:

   - Constant Time Part: This section contains any instructions that are executed only once. E.g. input, output, switch, if-else, arithmetic operations, etc.
   - Variable Time Part: This section contains any instruction that is executed more than once, say n times. E.g. loops, iterations, and so on.

   The mathematical equation for the time complexity (T(P)) of any algorithm P is:

   T(P) = C + TP(I)

   C: Constant Time Part

   TP(I): Variable time part of the algorithm that depends on the instance characteristics I.

1.1 Types of Algorithms

1. Backtracking Algorithm
2. Brute Force Algorithm
3. Divide and Conquer Algorithm
4. Dynamic Programming Algorithm
5. Greedy Algorithm
6. Hashing Algorithm

7. Randomized Algorithm
8. Recursive Algorithm
9. Searching Algorithm
10. Sorting Algorithm

2. Scenario

2. Intelligent Encoding

The National Emergency Security Foundation (NESF) is an existing secret intelligence service. They use various methods to send the messages they want. Their main task is to identify humanitarian problems through a secret intelligence service and solve them with the help of the country's tri-military forces.

Drugs can be pointed out as one of the main problems facing our society at present. To eliminate these drugs, an operation has been implemented by NESF. A special method has to be used to send some secret message required for this operation.

Message → WE WANT INFO

To use Huffman coding for secret messaging in the context of the NESF anti-narcotics operation, the secret message is first encoded using Huffman coding for data compression and then encrypted for transmission.

3. Huffman Coding

"Huffman code is an optimized prefix code frequently used for lossless data compression". This algorithm was developed by David A. Huffman and published in the 1952 publication "A Method for the Construction of Minimum-Redundancy Codes".

The output of Huffman's algorithm can be thought of as a variable-length code table for encoding a source symbol (such as a character in a file). This table is generated by the algorithm based on the estimated probability or frequency of occurrence (weight) for each potential value of the source symbol. As in other entropy coding schemes, frequent

symbols are often encoded with fewer bits than less common symbols. If the input weights are classified, Huffman's approach can be effectively applied, and a code can be found in linear time for the number of input weights. However, Huffman coding is not optimal among all compression methods, although it is the best among symbol-independent encoding methods; if a better compression ratio is required, it is substituted by arithmetic coding or asymmetric number systems.

Huffman coding applies a prefix rule to remove ambiguity during decoding. Huffman coding consists of two steps:

1. Create a Huffman tree using the given string, text, or characters.
2. By traversing the tree, each character is assigned a Huffman code.

Uses of the Huffman coding mechanism include:

- It is used when using a group of regularly occurring characters. For example, used to send data in the form of text or fax.
- Huffman coding is used in multimedia formats such as JPEG, PNG, and MP3.
- GZIP, PKZIP, and other traditional compression formats use it.

Character	Frequency
W	2
E	1
A	1
N	2
T	1
I	1
F	1
O	1
Space	2
Total	12

If we use ASCII code, it takes 96 (12*8 = 96) bits to message “WE WANT INFO”.

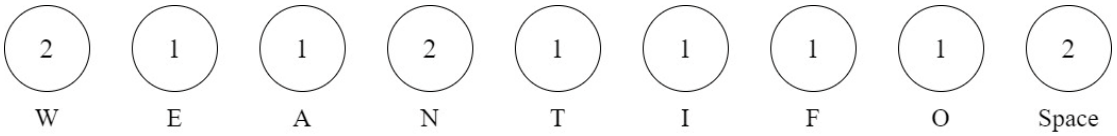
4. Huffman Tree

The following steps are used to generate a Huffman Tree:

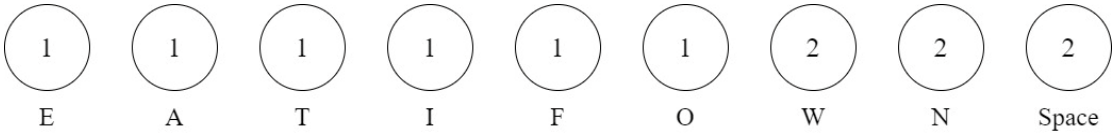
1. Create a leaf node for each text character.
2. Arrange all nodes in order of increasing frequency.
3. Considering that the first two nodes have the lowest frequency.
    - Create an internal node.
    - The frequency of this internal node is the sum of the frequencies of the previous two nodes.
    - Accordingly, make the first and second nodes the left and right children of the newly generated node.
4. Steps 2 and 3 should be repeated until all nodes have formed a tree. The resulting tree is a Huffman Tree.
5. Give weight to each end. The left edge has a weight of 0 and the right edge has a weight of 1.

4.1 Huffman Tree for the Given Scenario

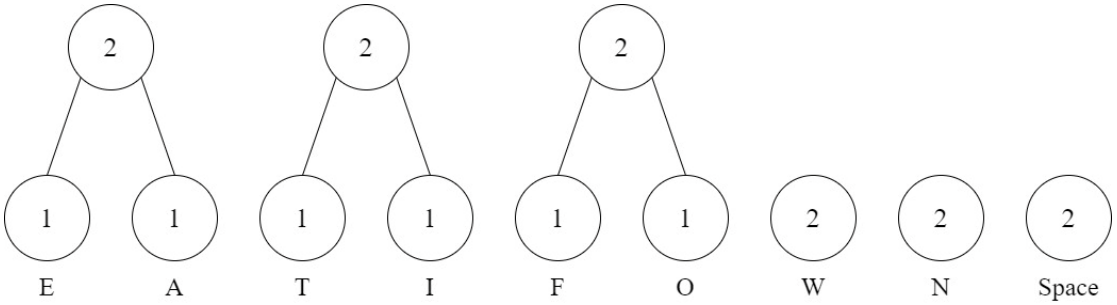
Step 1



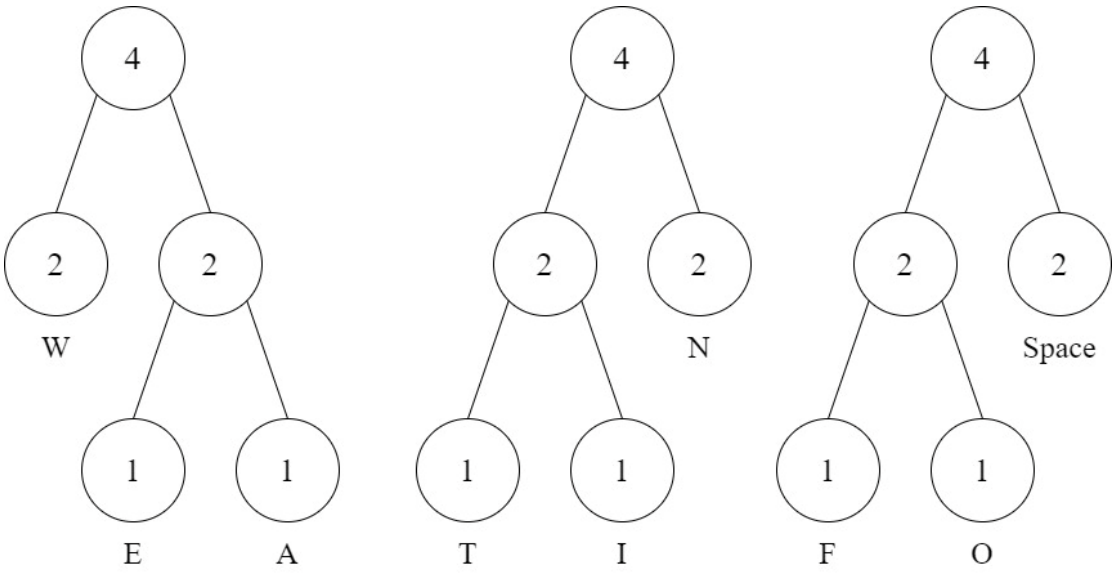
Step 2



Step 3

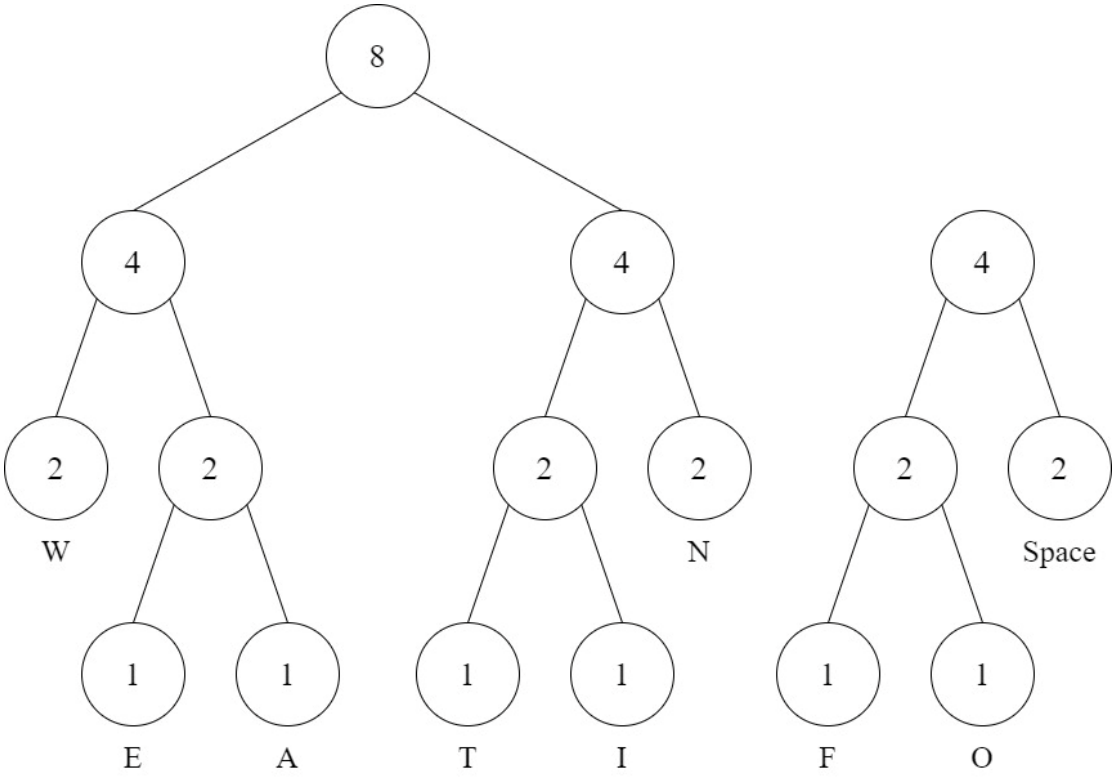


Step 4

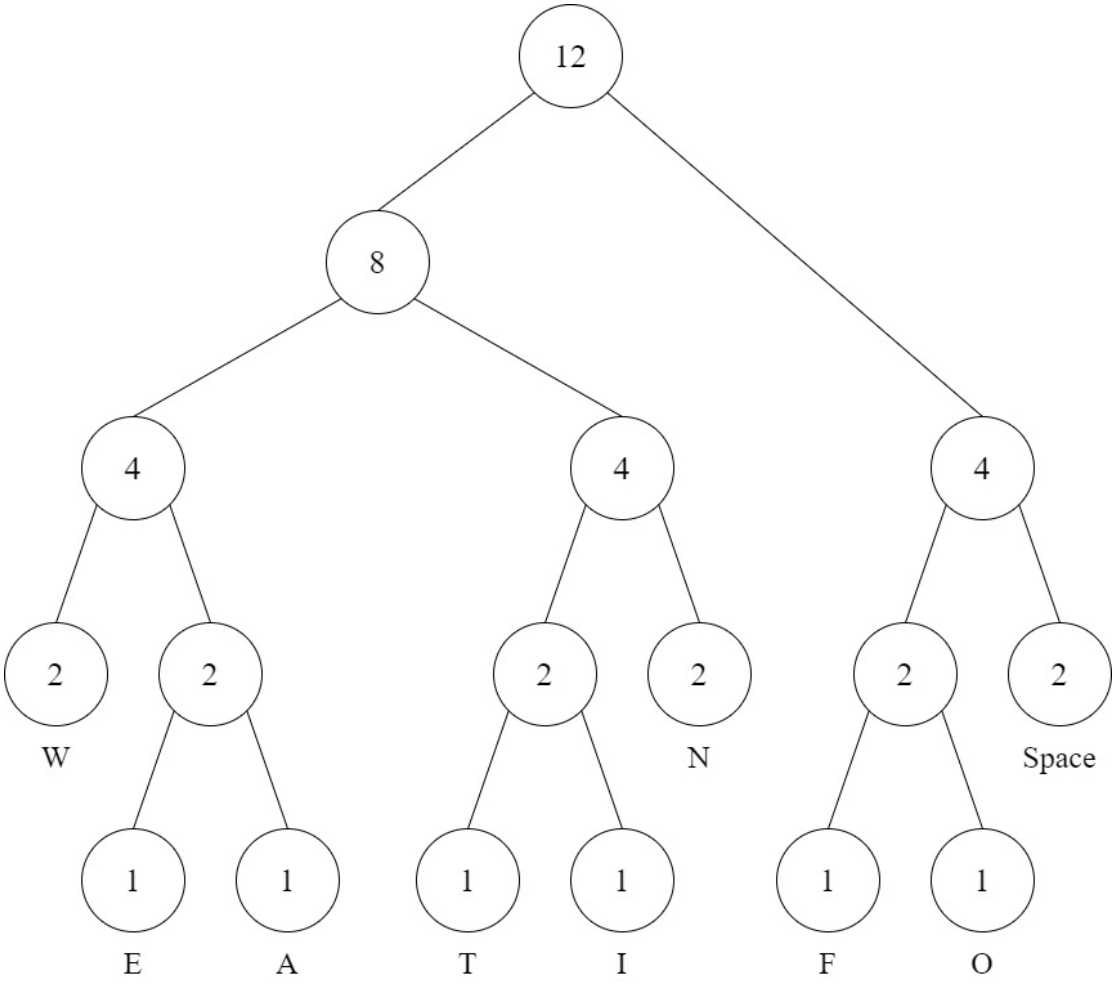


x

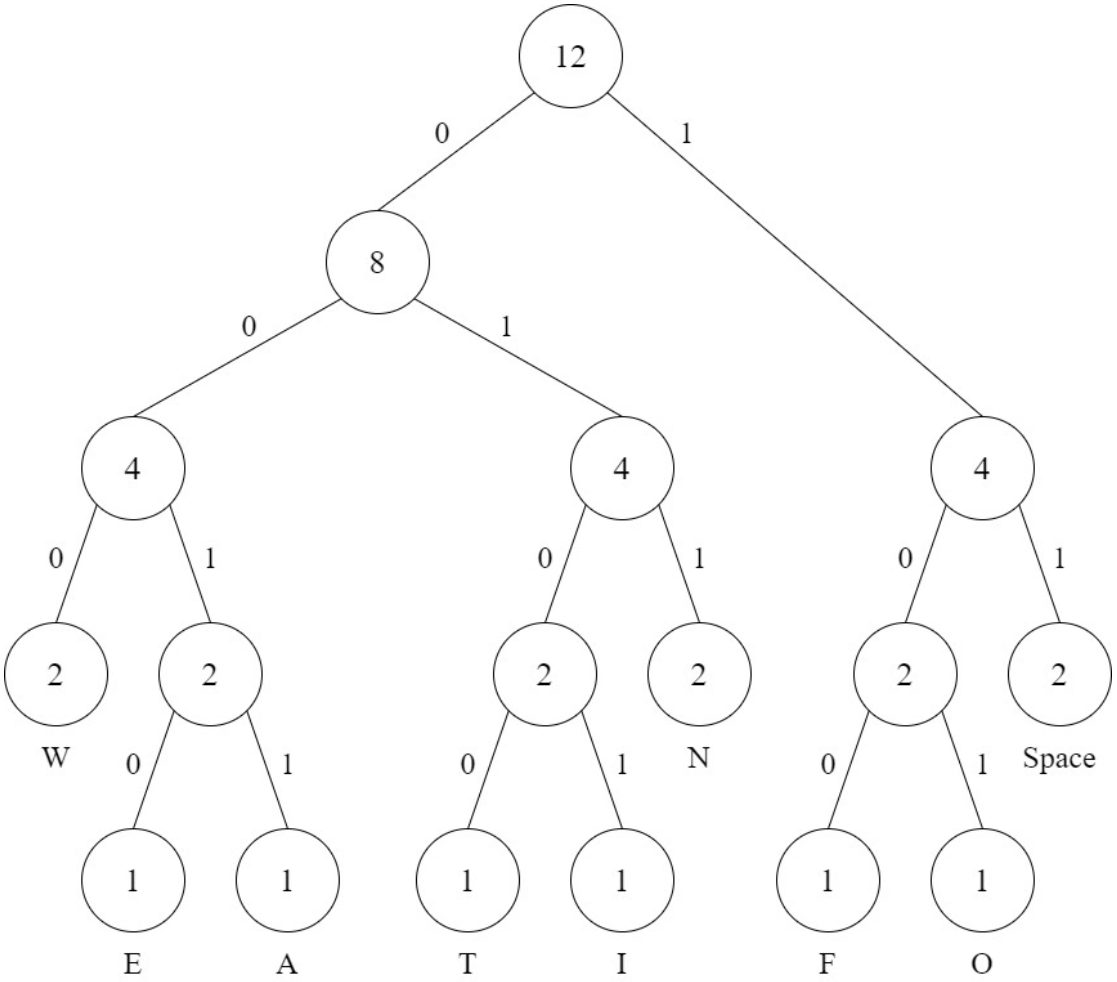
Step 5



Step 6



Step 7



5. Huffman Encoding

To find the Huffman code for each character, we traverse the Huffman Tree from the root node to the leaf node of the character we are looking for the code.

Character	Frequency	Code
W	2	000
E	1	0010
A	1	0011
N	2	011
T	1	0100
I	1	0101
F	1	100
O	1	101
Space	2	11
Total	12	30

The total bits needed for Huffman coding is:

$\sum$ (frequency) x (bits) = $2\times3 + 1\times4 + 1\times4 + 2\times3 + 1\times4 + 1\times4 + 1\times3 + 1\times3 + 2\times2$

= 38bits

The Huffman algorithm is a greedy algorithm. Because the algorithm searches for the best possible options at each step.

The time complexity of Huffman coding is O(n log n). The number of characters in the provided text is n.

6. Huffman Decoding

Huffman decoding is a technique for converting encoded data back to the original data. A Huffman Tree is constructed for an input string, and the characters are decoded according to their position in the tree. Below is the decryption procedure:

- Traverse the tree from the root node and start searching for the character.
- Add 0 to the code as we travel left in the binary tree.
- If we go right in the binary tree, add 1 to the code.

The input character is stored in the child node. It is then given the code made up of 0s and 1s. The time complexity of decoding a string is O(n) where n is the length of the string.

7. Java Code Solution for the Given Scenario

In the program below, I created compression and decompression logic using data structures such as Priority Queues, HashMaps, and Maps.

7.1 Code Solution for Encoding

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;

/**
 *
 * @author NUWAA
 */
public class HuffmanEncoding {

    static class Node implements Comparable<Node> {
```

```
        char character;
        int frequency;
        Node leftNode, rightNode;

        Node(char character, int frequency) {
            this.character = character;
            this.frequency = frequency;
        }

        Node(Node leftNode, Node rightNode) {
            this.leftNode = leftNode;
            this.rightNode = rightNode;
            this.frequency = leftNode.frequency + rightNode.frequency;
        }

        public int compareTo(Node node) {
            return this.frequency - node.frequency;
        }
    }

    static Map<Character, String> huffmanCodes = new HashMap<>();
    static String encodedMessage = "";

    public static void buildHuffmanTree(String message) {
        Map<Character, Integer> frequency = new HashMap<>();

        for (int i = 0; i < message.length(); i++) {
            if (!frequency.containsKey(message.charAt(i))) {
                frequency.put(message.charAt(i), 0);
            }
            frequency.put(message.charAt(i),
frequency.get(message.charAt(i)) + 1);
        }

        PriorityQueue<Node> priorityQueue = new PriorityQueue<>();

        for (Map.Entry<Character, Integer> entry :
frequency.entrySet()) {
            priorityQueue.add(new Node(entry.getKey(),
entry.getValue()));
        }

        while (priorityQueue.size() > 1) {
            Node leftNode = priorityQueue.poll();
            Node rightNode = priorityQueue.poll();
```

```
            priorityQueue.add(new Node(leftNode, rightNode));
        }

        Node rootNode = priorityQueue.poll();

        buildHuffmanCodes(rootNode, "");
    }

    public static void buildHuffmanCodes(Node rootNode, String message)
{
        if (rootNode.leftNode == null && rootNode.rightNode == null) {
            huffmanCodes.put(rootNode.character, message);
            return;
        }
        buildHuffmanCodes(rootNode.leftNode, message + "0");
        buildHuffmanCodes(rootNode.rightNode, message + "1");
    }

    public static String encodeMessage(String message) {
        for (int i = 0; i < message.length(); i++) {
            encodedMessage += huffmanCodes.get(message.charAt(i));
        }
        return encodedMessage;
    }

    public static void main(String[] args) {
        String message = "WE WANT INFO";
        buildHuffmanTree(message);
        System.out.println("Original Message: " + message);
        System.out.println("Huffman Encoded Message: " +
encodeMessage(message));
        HuffmanDecoding.decodeMessage(encodedMessage);
        System.out.println("Huffman Decoded Message: " +
HuffmanDecoding.decodedMessage);
    }
}
```

This code uses a Huffman Tree to encode the input string "WE WANT INFO" into a binary string. The buildHuffmanTree method builds a Huffman Tree using a frequency map of the characters in the input string and a priority queue to combine the least frequent characters. The buildHuffmanCodes method checks whether the character is left root or right root. The encodeMessage method then returns the entire encoded message.

7.2 Code Solution for Decoding

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package model;

import java.util.Map;

/**
 *
 * @author NUWAA
 */
public class HuffmanDecoding {

    static Map<Character, String> huffmanCodes =
HuffmanEncoding.huffmanCodes;
    static String decodedMessage = "";

    public static void decodeMessage(String encodedMessage) {
        StringBuilder stringBuilder = new StringBuilder();
        for (int i = 0; i < encodedMessage.length(); i++) {
            stringBuilder.append(encodedMessage.charAt(i));
            for (Map.Entry<Character, String> entry :
huffmanCodes.entrySet()) {
                if (entry.getValue().equals(stringBuilder.toString()))
{
                    decodedMessage += entry.getKey();
                    stringBuilder = new StringBuilder();
                }
            }
        }
    }
}
```

This code uses the Huffman codes generated in the previous example to decode the encoded string. It iterates through the encoded string, adding each character to a temporary string and checking if the temporary string matches any Huffman code. If a match is found, the corresponding character is appended to the decoded string and the

temporary string is reset. Note that it uses the huffmanCodes and encodedMessage variables from the previous example.

Output:

Original Message: WE WANT INFO

Huffman Encoded Message: 00101111000101010001111001010011111110

Huffman Decoded Message: WE WANT INFO

8. References:

Wikipedia Contributors (2019). *Algorithm*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Algorithm.

Programiz (n.d.). *What is an Algorithm?* [online] www.programiz.com. Available at: https://www.programiz.com/dsa/algorithm.

Upadhyay, S. (2022). *What Is An Algorithm? Characteristics, Types and How to write it | Simplilearn*. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/what-is-an-algorithm.

Prabhu, R. (2019). *Introduction to Algorithms*. [online] GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-algorithms/.

Downey, L. (2019). *Algorithm Definition*. [online] Investopedia. Available at: https://www.investopedia.com/terms/a/algorithm.asp.

Kumar, P. (2020). *What is the algorithm? Write its criteria and characteristics*. [online] Quescol. Available at: https://quescol.com/data-structure/algorithm-criteria-and-characteristics.

Tutorialspoint (2019). *Data Structures - Algorithms Basics*. [online] Tutorialspoint.com. Available at: https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm.

Shlok Bhatt (2019). *Characteristics of an Algorithm*. [online] Medium. Available at: https://medium.com/@bhattshlok12/characteristics-of-an-algorithm-49cf4d7bcd9.

Anon, (n.d.). *Need of an Algorithm – StudiousGuy*. [online] Available at: https://studiousguy.com/need-of-an-algorithm/ [Accessed 26 Jan. 2023].

Anon, (n.d.). *Explain Characteristics And Factors Of Algorithm*. [online] Available at: https://www.engineerscreator.com/2021/05/features-of-algorithm.html.

tutorialink.com. (n.d.). *Key features of an algorithm | Data Structure | Tutorialink.com*. [online] Available at: https://tutorialink.com/ds/key-features-of-an-algorithm.ds.

Study.com. (2019). *Properties of Algorithms - Video & Lesson Transcript | Study.com*. [online] Available at: https://study.com/academy/lesson/properties-of-algorithms.html.

Aristides S. Bouras. (n.d.). *Properties of an Algorithm*. [online] Available at: https://www.bouraspage.com/repository/algorithmic-thinking/properties-of-an-algorithm.

www.includehelp.com. (n.d.). *Algorithm and its properties - IncludeHelp*. [online] Available at: https://www.includehelp.com/data-structure-tutorial/algorithm-and-its-properties.aspx.

Webeduclick.com. (n.d.). *Properties of an Algorithm - Webeduclick*. [online] Available at: https://webeduclick.com/properties-of-an-algorithm/.

Doubtnut (n.d.). *Define Big O notation. State the two factors which determine the c*. [online] doubtnut. Available at: https://www.doubtnut.com/pcmb-questions/define-

big-o-notation-state-the-two-factors-which-determine-the-complexity-of-an-algorithm-96018 [Accessed 26 Jan. 2023].

www.tutorialspoint.com. (n.d.). *Time and Space Complexity in Data Structure*. [online] Available at: https://www.tutorialspoint.com/time-and-space-complexity-in-data-structure.

an (2018). *Define Big 'O' notation. State the two factors which determine the complexity of an algorithm.* [online] Sarthaks eConnect | Largest Online Education Community. Available at: https://www.sarthaks.com/219896/define-big-notation-state-the-two-factors-which-determine-the-complexity-of-an-algorithm [Accessed 26 Jan. 2023].

Team, G.L. (2020). *Time Complexity Algorithm | What is Time Complexity?* [online] GreatLearning. Available at: https://www.mygreatlearning.com/blog/why-is-time-complexity-essential/.

www.tutorialspoint.com. (n.d.). *Algorithms and Complexities*. [online] Available at: https://www.tutorialspoint.com/Algorithms-and-Complexities.

www.javatpoint.com. (n.d.). *DAA Complexity of Algorithm - javatpoint*. [online] Available at: https://www.javatpoint.com/daa-complexity-of-algorithm.

Simplilearn.com. (n.d.). *Data Structure and Algorithm Complexity (A Complete Guide) | Simplilearn*. [online] Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/algorithm-complexity-in-data-structure.

GeeksforGeeks. (2020). *Most important type of Algorithms*. [online] Available at: https://www.geeksforgeeks.org/most-important-type-of-algorithms/.

EDUCBA. (2019). *Types of Algorithms | Learn The Top 6 Important Types of Algorithms*. [online] Available at: https://www.educba.com/types-of-algorithms/.

www.codingninjas.com. (n.d.). *Code Studio*. [online] Available at: https://www.codingninjas.com/codestudio/library/types-of-algorithms-and-their-uses.

Team, C. (2022). *Best 7 Types Of Algorithms You Should Know - CallTutors*. [online] Available at: https://www.calltutors.com/blog/types-of-algorithms/.

Anon, (2022). *Types of Algorithms - 8 Types | Types of Algorithm*. [online] Available at: https://codinghero.ai/discover-8-types-of-algorithms-for-efficient-problem-solving/ [Accessed 26 Jan. 2023].

Wikipedia Contributors (2019). *Huffman coding*. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Huffman_coding.

GeeksforGeeks. (2012). *Huffman Coding | Greedy Algo-3*. [online] Available at: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/.

www.programiz.com. (n.d.). *Huffman Coding Algorithm*. [online] Available at: https://www.programiz.com/dsa/huffman-coding.

www.javatpoint.com. (n.d.). *Huffman Coding Java - Javatpoint*. [online] Available at: https://www.javatpoint.com/huffman-coding-java [Accessed 26 Jan. 2023].

www.tutorialspoint.com. (n.d.). *Huffman Coding Algorithm*. [online] Available at: https://www.tutorialspoint.com/Huffman-Coding-Algorithm.

brilliant.org. (n.d.). *Huffman Code | Brilliant Math & Science Wiki*. [online] Available at: https://brilliant.org/wiki/huffman-encoding/.

engineering.purdue.edu. (n.d.). *Huffman Coding*. [online] Available at: https://engineering.purdue.edu/ece264/17au/hw/HW13?alt=huffman.

www.topcoder.com. (n.d.). *Huffman Coding and Decoding Algorithm*. [online] Available at: https://www.topcoder.com/thrive/articles/huffman-coding-and-decoding-algorithm [Accessed 26 Jan. 2023].

www.gatevidyalay.com. (n.d.). *Huffman Coding | Huffman Coding Example | Time Complexity | Gate Vidyalay*. [online] Available at: https://www.gatevidyalay.com/huffman-coding-huffman-encoding/.

www.studytonight.com. (n.d.). *Huffman Coding Algorithm | Studytonight*. [online] Available at: https://www.studytonight.com/data-structures/huffman-coding.

www.sciencedirect.com. (n.d.). *Huffman Code - an overview | ScienceDirect Topics*. [online] Available at: https://www.sciencedirect.com/topics/mathematics/huffman-code [Accessed 26 Jan. 2023].

GeeksforGeeks. (2018). *Canonical Huffman Coding*. [online] Available at: https://www.geeksforgeeks.org/canonical-huffman-coding/ [Accessed 26 Jan. 2023].

www.tutorialspoint.com. (n.d.). *Huffman Trees in Data Structure*. [online] Available at: https://www.tutorialspoint.com/huffman-trees-in-data-structure.

cgi.luddy.indiana.edu. (n.d.). *Data Structures*. [online] Available at: https://cgi.luddy.indiana.edu/~yye/c343-2019/huffman.php [Accessed 26 Jan. 2023].

OpenGenus IQ: Computing Expertise & Legacy. (n.d.). *Huffman Encoding [explained with example and code]*. [online] Available at: https://iq.opengenus.org/huffman-encoding/.

GeeksforGeeks. (2017). *Huffman Decoding*. [online] Available at: https://www.geeksforgeeks.org/huffman-decoding/.

Aktaş, Y.Ç. (2021). *Huffman Decoding*. [online] Medium. Available at: https://towardsdatascience.com/huffman-decoding-cca770065bab [Accessed 26 Jan. 2023].

OpenGenus IQ: Computing Expertise & Legacy. (n.d.). *Huffman Decoding [explained with example]*. [online] Available at: https://iq.opengenus.org/huffman-decoding/.

HackerRank. (n.d.). *Tree: Huffman Decoding*. [online] Available at: https://www.hackerrank.com/challenges/tree-huffman-decoding/problem [Accessed 26 Jan. 2023].