



**Green University of Bangladesh**  
**Department of Computer Science and Engineering (CSE)**  
**Faculty of Sciences and Engineering**  
**Semester: (Fall, Year: 2023), B.Sc. in CSE (Day)**

**Lab Report NO: 10**

**Course Title : Algorithm Lab**  
**Course Code : CSE 204**  
**Section : D - 9**

**Lab Experiment Name : Huffman Coding Algorithm.**

**Student Details**

Name		ID
1.	Jahidul Islam	221002504

**Lab Date : 04/12/2023**  
**Submission Date : 31/12/2023**  
**Course Teacher's Name : Md. Abu Rahman Refat**

**Lab Report Status**

**Marks: .....**  
**Comments:.....**

**Signature:.....**  
**Date:.....**

## 1. TITLE:

Huffman Coding Algorithm.

## 2. OBJECTIVES/AIM:

The primary objectives or aim of this lab experiment are:

- To define Huffman Coding.
- To understand how Huffman Coding works.
- To implement Huffman Coding algorithm.

## 3. PROBLEM ANALYSIS & PROCEDURE:

Step 1: For each character of the node, create a leaf node. The leaf node of a character contains the frequency of that character.

Step 2: Set all the nodes in sorted order according to their frequency.

Step 3: There may exist a condition in which two nodes may have the same frequency.

In such a case, do the following:

1. Create a new internal node.
2. The frequency of the node will be the sum of the frequency of those two nodes that have the same frequency.
3. Mark the first node as the left child and another node as the right child of the newly created internal node.

Step 4: Repeat step 2 and 3 until all the node forms a single tree. Thus, we get a Huffman tree.

## 4. ALGORITHM:

---

### Algorithm 1: Huffman Coding

---

```
1 Algorithm Huffman(c):  
2   n = |c|  
3   Q = c  
4   for i < -1 to n - 1 do  
5       temp = get_node()  
6       left[temp] = get_min(Q)  
7       right[temp] = get_min(Q)  
8       a = left[temp]  
9       b = right[temp]  
10      F[temp] = f[a] + f[b]  
11      insert(Q, temp)  
12   end  
13 return get_min(0)
```

---

## 5. IMPLEMENTATION:

### Code in Java:

```
import java.util.*;

class HuffmanNode implements Comparable<HuffmanNode> {
    char data;
    int frequency;
    HuffmanNode left, right;

    public HuffmanNode(char data, int frequency) {
        this.data = data;
        this.frequency = frequency;
        left = right = null;
    }

    @Override
    public int compareTo(HuffmanNode node) {
        return this.frequency - node.frequency;
    }
}

public class HuffmanCoding {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter a string for Huffman coding: ");
        String inputString = scanner.nextLine();

        scanner.close();

        HashMap<Character, String> huffmanCodes =
buildHuffmanTree(inputString);

        System.out.println("Original String: " + inputString);
        System.out.println("Huffman Codes: " + huffmanCodes);

        String encodedString = encode(inputString, huffmanCodes);
        System.out.println("Encoded String: " + encodedString);

        String decodedString = decode(encodedString, huffmanCodes);
        System.out.println("Decoded String: " + decodedString);
    }

    private static HashMap<Character, String> buildHuffmanTree(String input)
    {
        HashMap<Character, Integer> frequencyMap = new HashMap<>();
        for (char c : input.toCharArray()) {
            frequencyMap.put(c, frequencyMap.getOrDefault(c, 0) + 1);
        }
    }
}
```

```

    }

    PriorityQueue<HuffmanNode> priorityQueue = new PriorityQueue<>();
    for (char key : frequencyMap.keySet()) {
        priorityQueue.add(new HuffmanNode(key, frequencyMap.get(key)));
    }

    while (priorityQueue.size() > 1) {
        HuffmanNode left = priorityQueue.poll();
        HuffmanNode right = priorityQueue.poll();
        HuffmanNode combinedNode = new HuffmanNode('\0', left.frequency +
right.frequency);
        combinedNode.left = left;
        combinedNode.right = right;
        priorityQueue.add(combinedNode);
    }

    HashMap<Character, String> huffmanCodes = new HashMap<>();
    generateHuffmanCodes(priorityQueue.peek(), "", huffmanCodes);

    return huffmanCodes;
}

private static void generateHuffmanCodes(HuffmanNode root, String code,
HashMap<Character, String> huffmanCodes) {
    if (root == null) return;

    if (root.data != '\0') {
        huffmanCodes.put(root.data, code);
    }

    generateHuffmanCodes(root.left, code + "0", huffmanCodes);
    generateHuffmanCodes(root.right, code + "1", huffmanCodes);
}

private static String encode(String input, HashMap<Character, String>
huffmanCodes) {
    StringBuilder encodedString = new StringBuilder();
    for (char c : input.toCharArray()) {
        encodedString.append(huffmanCodes.get(c));
    }
    return encodedString.toString();
}

private static String decode(String encodedString, HashMap<Character,
String> huffmanCodes) {
    StringBuilder decodedString = new StringBuilder();
    int index = 0;

    while (index < encodedString.length()) {
        for (char key : huffmanCodes.keySet()) {
            String code = huffmanCodes.get(key);
            if (index + code.length() <= encodedString.length() &&
encodedString.substring(index, index +
code.length()).equals(code)) {
                decodedString.append(key);
                index += code.length();
            }
        }
    }
}

```

```

        break;
    }
}
}

return decodedString.toString();
}
}

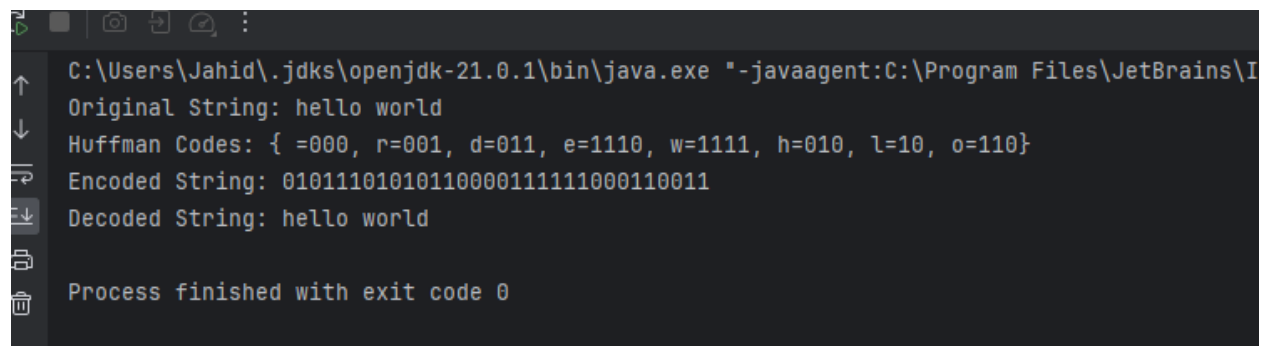
```

## 6. INPUT STRING:

Input taken from the user during the execution.

## 7. TEST RESULT / OUTPUT:

**This code is static**



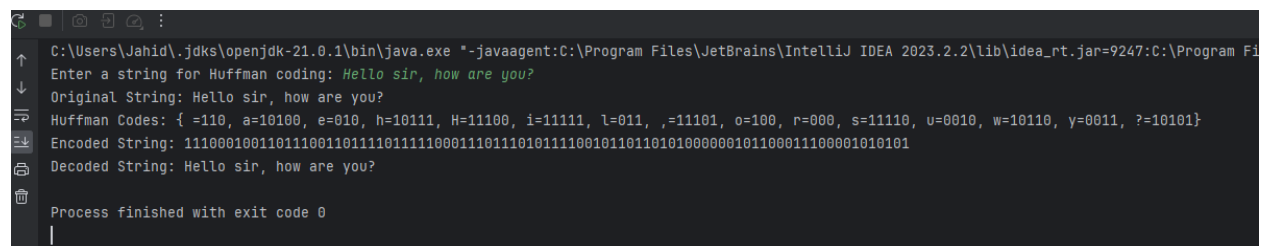
```

C:\Users\Jahid\.jdk\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\I
Original String: hello world
Huffman Codes: { =000, r=001, d=011, e=1110, w=1111, h=010, l=10, o=110}
Encoded String: 01011101010110000111111000110011
Decoded String: hello world

Process finished with exit code 0

```

This is dynamic. Taken input form the user.



```

C:\Users\Jahid\.jdk\openjdk-21.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=9247:C:\Program Fi
Enter a string for Huffman coding: Hello sir, how are you?
Original String: Hello sir, how are you?
Huffman Codes: { =110, a=10100, e=010, h=10111, H=11100, i=11111, l=011, ,=11101, o=100, r=000, s=11110, u=0010, w=10110, y=0011, ?=10101}
Encoded String: 1110001001101110011011111000111011101011110010110110101000000101100011100001010101
Decoded String: Hello sir, how are you?

Process finished with exit code 0

```

## 8. DISCUSSION AND ANALYSIS:

The construction of the Huffman tree is a critical aspect of the program. The frequency analysis accurately determines the occurrence of each character in the input

string, and the priority queue facilitates the creation of an optimal Huffman tree. The use of a priority queue ensures that characters with higher frequencies receive shorter codes.

- **Huffman Code Generation:**

The 'generateHuffmanCodes' method effectively traverses the Huffman tree to assign unique binary codes to each character. The codes are stored in a HashMap, enabling quick access during the encoding and decoding processes. The program efficiently utilizes recursion to navigate through the tree, demonstrating a clear understanding of the underlying algorithm.

- **Encoding and Decoding:**

The encoding and decoding processes exhibit the correctness of the Huffman coding implementation. The program successfully encodes the input string by replacing each character with its corresponding Huffman code. Subsequently, the decoding process accurately reconstructs the original string from the encoded data, emphasizing the lossless nature of Huffman coding.

- **Limitations:**

- a. **Quadratic Time Complexity in Worst Case:**

- While the dynamic programming approach provides an efficient solution, the worst-case time complexity of  $O(N*M)$  could be a limitation for extremely long sequences.

- b. **Memory Consumption:**

- The algorithm's space complexity is proportional to the product of the lengths of the input sequences, potentially leading to high memory consumption for large sequences.

## **9. CONCLUSION:**

In conclusion, the Huffman coding program demonstrates a successful implementation of the Huffman coding algorithm in Java. The program effectively engages users through interactive input, constructs an optimal Huffman tree, generates Huffman codes, and demonstrates the compression and decompression processes. The discussion and analysis provided valuable insights into the program's functionality, identified areas for improvement, and highlighted its educational significance in

understanding data compression techniques.