```
/*
Student Name : Abhang Rushikesh
Roll NO : BEA-30
 */
import java.util.Comparator;
import java.util.Scanner;
import java.util.HashMap;
import java.util.Map;
import java.util.PriorityQueue;
//defining a class that creates nodes of the tree
class Node
{
    //storing character in ch variable of type character
    Character ch;
    //storing frequency in freq variable of type int
    Integer freq;
    //initially both child (left and right) are null
    Node left = null;
    Node right = null;
    //creating a constructor of the Node class
    Node(Character ch, Integer freq)
    {
        this.ch = ch;
        this.freq = freq;
    }
    //creating a constructor of the Node class
    public Node(Character ch, Integer freq, Node left, Node right)
    {
        this.ch = ch;
        this.freq = freq;
        this.left = left;
        this.right = right;
    }
}
//main class
public class Main //HuffmanCode
{
    //function to build Huffman tree
    public static void createHuffmanTree(String text)
    {
        //base case: if user does not provides string  text
        if (text == null || text.length() == 0)
        {
            return;
        }
        //count the frequency of appearance of each character and store
it in a map
        //creating an instance of the Map
        Map<Character, Integer> freq = new HashMap<>();
        //loop iterates over the string and converts the text into
character array
        for (char c: text.toCharArray())
        {
            //storing character and their frequency into Map by invoking
the put() method
            freq.put(c, freq.getOrDefault(c, 0) + 1);
        }
```

```
        //create a priority queue that stores current nodes of the
Huffman tree.
        //here a point to note that the highest priority means the lowest
frequency
        PriorityQueue<Node> pq = new
PriorityQueue<>(Comparator.comparingInt(a -> a.freq));
        //loop iterate over the Map and returns a Set view of the
mappings contained in this Map
        for (var entry: freq.entrySet())
        {
            //creates a leaf node and add it to the queue
            pq.add(new Node(entry.getKey(), entry.getValue()));
        }
        //while loop runs until there is more than one node in the queue
        while (pq.size() != 1)
        {
            //removing the nodes having the highest priority (the lowest
frequency) from the queue
            Node left = pq.poll();
            Node right = pq.poll();
            //create a new internal node with these two nodes as children
and with a frequency equal to the sum of both nodes' frequencies. Add the
new node to the priority queue.
            //sum up the frequency of the nodes (left and right) that we
have deleted
            int sum = left.freq + right.freq;
            //adding a new internal node (deleted nodes i.e. right and
left) to the queue with a frequency that is equal to the sum of both
nodes
            pq.add(new Node(null, sum, left, right));
        }
        //root stores pointer to the root of Huffman Tree
        Node root = pq.peek();
        //trace over the Huffman tree and store the Huffman codes in a
map
        Map<Character, String> huffmanCode = new HashMap<>();
        encodeData(root, "", huffmanCode);
        //print the Huffman codes for the characters
        System.out.println("Huffman Codes of the characters are: " +
huffmanCode);
        //prints the initial data
        System.out.println("The initial string is: " + text);
        //creating an instance of the StringBuilder class
        StringBuilder sb = new StringBuilder();
        //loop iterate over the character array
        for (char c: text.toCharArray())
        {
            //prints encoded string by getting characters
            sb.append(huffmanCode.get(c));
        }
        System.out.println("The encoded string is: " + sb);
 /*       System.out.print("The decoded string is: ");
        if (isLeaf(root))
        {
            //special case: For input like a, aa, aaa, etc.
            while (root.freq-- > 0)
            {
                System.out.print(root.ch);
```

```
            }
        }
        else
        {
            //traverse over the Huffman tree again and this time, decode
the encoded string
            int index = -1;
            while (index < sb.length() - 1)
            {
                index = decodeData(root, index, sb);
            }
        }  */
    }
//traverse the Huffman Tree and store Huffman Codes in a Map
    //function that encodes the data
    public static void encodeData(Node root, String str, Map<Character,
String> huffmanCode)
    {
        if (root == null)
        {
            return;
        }
        //checks if the node is a leaf node or not
        if (isLeaf(root))
        {
            huffmanCode.put(root.ch, str.length() > 0 ? str : "1");
        }
        encodeData(root.left, str + '0', huffmanCode);
        encodeData(root.right, str + '1', huffmanCode);
    }
    //traverse the Huffman Tree and decode the encoded string function
that decodes the encoded data
   /* public static int decodeData(Node root, int index, StringBuilder
sb)
    {
        //checks if the root node is null or not
        if (root == null)
        {
            return index;
        }
        //checks if the node is a leaf node or not
        if (isLeaf(root))
        {
            System.out.print(root.ch);
            return index;
        }
        index++;
        root = (sb.charAt(index) == '0') ? root.left : root.right;
        index = decodeData(root, index, sb);
        return index;
    }  */
    //function to check if the Huffman Tree contains a single node
    public static boolean isLeaf(Node root)
    {
        //returns true if both conditions return ture
        return root.left == null && root.right == null;
    }
    //driver code
```

```
    public static void main(String args[])
    {

        Scanner myObj = new Scanner(System.in);  // Create a Scanner
object
        System.out.println("Enter string:");

        String text = myObj.nextLine();  // Read user input
        //function calling
        createHuffmanTree(text);
    }
}
```