***Problem statement :***

Compress and decompress any one of the messages sent between any two nodes.

***Solution:***  
  
***Code :***

// Decompression

String decodedMessage = decodeMessage(encodedMessage, root);

System.out.println("Decoded Message: " + decodedMessage);

}

public static HuffmanNode buildHuffmanTree(PriorityQueue<HuffmanNode> priorityQueue) {

while (priorityQueue.size() > 1) {

HuffmanNode left = priorityQueue.poll();

HuffmanNode right = priorityQueue.poll();

HuffmanNode parent = new HuffmanNode('$', left.frequency + right.frequency);

parent.left = left;

parent.right = right;

priorityQueue.offer(parent);

}

return priorityQueue.peek();

}

public static void generateCodes(HuffmanNode root, String code, Map<Character, String> huffmanCodes) {

if (root == null) {

return;

}

if (root.left == null && root.right == null) {

huffmanCodes.put(root.data, code);

return;

}

generateCodes(root.left, code + "0", huffmanCodes);

generateCodes(root.right, code + "1", huffmanCodes);

}

public static String encodeMessage(String text, Map<Character, String> huffmanCodes) {

StringBuilder encodedMessage = new StringBuilder();

for (char c : text.toCharArray()) {

encodedMessage.append(huffmanCodes.get(c));

}

return encodedMessage.toString();

}

public static String decodeMessage(String encodedMessage, HuffmanNode root) {

StringBuilder decodedMessage = new StringBuilder();

HuffmanNode current = root;

for (char bit : encodedMessage.toCharArray()) {

if (bit == '0') {

current = current.left;

} else if (bit == '1') {

current = current.right;

}

if (current.left == null && current.right == null) {

decodedMessage.append(current.data);

current = root;

}

}

return decodedMessage.toString();

}

}

***Explanation and Answers:***

*Huffman Coding for Decompression*

An algorithm for lossless data compression is huffman coding. In order to recover the original characters from a binary tree (Huffman tree), a Huffman-encoded message must be decompressed. This explains why Huffman coding works well for decompression.

1.Tree-Based Decompression: The Huffman code for a character is determined by the path from the root to a leaf node in a binary tree created by Huffman coding. Leaf nodes in the tree represent individual characters. During the decompression process, the original characters are reconstructed by following the encoded message, which is represented by a sequence of bits, as it traverses this Huffman tree.

2.Prefix-Free Property- There are no codes that are prefixes to other codes according to the Huffman coding technique. This characteristic removes uncertainty in decoding, enabling a clear exploration of the Huffman tree according to the bits of the encoded message.

3,Efficient Decompression- Decompression is effective because it follows the encoded bits to traverse the Huffman tree, where a '0' indicates the left child and a '1' indicates the right child. The matching character is added to the decoded message upon reaching a leaf node, facilitating a speedy reconstruction of the original message.

4.Reset After Decoding- The traversal resets to the root to proceed decoding the remaining characters in the message after decoding a character (upon reaching a leaf node). This reset guarantees error-free handling of continuous encoded character sequences by the decompression process.

**Worst-Case Time Complexity for Decompression**

The worst-case time complexity for Huffman decompression depends on the number of bits in the encoded message and the traversal through the Huffman tree:

* **Tree Traversal**: The time complexity for decompressing the message is 𝑂(𝑛)*O*(*n*), where 𝑛*n* is the length of the encoded message. Each bit requires one step through the Huffman tree.
* **Reset and Continue**: After decoding a character (reaching a leaf node), the traversal resets to the root to continue decoding the next sequence of bits.

Given these factors, the worst-case time complexity for Huffman decompression is 𝑂(𝑛)*O*(*n*), where 𝑛*n* represents the length of the encoded message. This complexity ensures that Huffman decompression is efficient and suitable for real-time operations, even with large encoded messages.