Huffman coding using Greedy Method

```
import java.util.PriorityQueue;

class Node implements Comparable<Node> {
   char ch;
   int freq;
   Node left, right;
   Node(char ch, int freq) {
      this.ch = ch;
      this.freq = freq;
      this.left = null;
      this.right = null;
   }
 Node(int freq, Node left, Node right) {
      this.ch = '-';
      this.freq = freq;
      this.left = left;
      this.right = right;
   }
   public int compareTo(Node n) {
      return this.freq - n.freq;
   }
}


public class HuffmanCodingGreedy {
   public static void printCodes(Node root, String code) {
      if (root == null) return;
```

```
    if (root.left == null && root.right == null && root.ch != '-') {
        System.out.println(root.ch + " : " + code);
        return;
    }
    printCodes(root.left, code + "0");
    printCodes(root.right, code + "1");
}

public static void main(String[] args) {
    char[] chars = {'A', 'B', 'C', 'D', 'E'};
    int[] freq = {5, 9, 12, 13, 16};

    PriorityQueue<Node> pq = new PriorityQueue<>();
   // Step 1: Create leaf nodes and add to priority queue
    for (int i = 0; i < chars.length; i++) {
        pq.add(new Node(chars[i], freq[i]));
    }

    // Step 2: Apply greedy approach (similar to Optimal Merge Pattern)
    while (pq.size() > 1) {
        Node left = pq.poll();
        Node right = pq.poll();
     Node merged = new Node(left.freq + right.freq, left, right);
        pq.add(merged);
    }
   // Step 3: Print Huffman codes
    Node root = pq.peek();
```

```
        System.out.println("Huffman Codes for given characters:");
        printCodes(root, "");
    }
}
```

Output