import java.util.\*;  
  
class Node {  
 char character;//存储节点所对应的字符串  
 int frequency;//节点对应字符出现的频次  
 Node left = null, right = null;  
 Node(char character, int frequency) {  
 this.character = character;  
 this.frequency = frequency;  
 }  
 Node(char character, int frequency, Node left, Node right) {  
 this.character = character;  
 this.frequency = frequency;  
 this.left = left;  
 this.right = right;  
 }  
}  
  
public class Huffmancoding {  
 // 为哈夫曼树中的每个字符生成编码  
 public static void encode(Node root, String str, java.util.Map<Character, String> huffmanCode) {  
 //root为此根节点，str是改节点对应的编码字符串，Map用于存储字符和对应的哈夫曼编码的映射关系  
 if (root == null)  
 return;  
 //判断若为叶子结点  
 if (root.left == null && root.right == null) {  
 huffmanCode.put(root.character, str.length() > 0 ? str : "1");//将叶子节点对应的字符和路径上的编码字符串存入  
 }  
 //哈弗曼编码左子树为0，右子树为1  
 encode(root.left, str + "0", huffmanCode);  
 encode(root.right, str + "1", huffmanCode);  
 }  
  
 // 建立哈夫曼树  
 public static Node buildHuffmanTree(java.util.Map<Character, Integer> freq) {//根据出现频率来建树  
 // 创建一个优先队列来存储按频率排序的树节点  
 PriorityQueue<Node> pq;  
 pq = new PriorityQueue<>(Comparator.comparingInt(n -> n.frequency));//指定节点的频率frequencd作为比较依据  
 // 为每个字符创建一个节点并添加到优先队列中  
 for (var entry : freq.entrySet()) {//遍历freq映射中的每一个键值对  
 //entry是在每次循环迭代时，用来表示当前键值对的临时变量，表示一个字符getKey及其对应的频率getValue  
 pq.add(new Node(entry.getKey(), entry.getValue()));  
 }  
 // 循环直到队列中只剩一个节点  
 while (pq.size() != 1) {  
 // 移除两个最高优先级（最小频率）的节点  
 Node left = pq.poll();  
 Node right = pq.poll();  
 // 创建一个新的内部节点，频率是两个节点之和  
 int sum = left.frequency + right.frequency;  
 pq.add(new Node('\0', sum, left, right));  
 }  
 // 返回队列中的剩余节点（哈夫曼树的根节点）  
 return pq.peek();  
 }  
  
 public static void decode(Node root, String encodedString) {  
 Node current = root;  
 System.out.print("解码结果：");  
 for (char bit : encodedString.toCharArray()) {  
 if (bit == '0') {  
 current = current.left;  
 } else if (bit == '1') {  
 current = current.right;  
 }  
  
 if (current.left == null && current.right == null) {  
 System.out.print(current.character);  
 current = root;  
 }  
 }  
 System.out.println(); // 打印换行符  
 }  
  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.in);  
 Map<Character, Integer> frequency = new HashMap<>();  
 System.out.print("输入字符和它们对应的频次(例如：a 5)，");  
 while (true) {  
 System.out.print("输入字符和对应的频次（输入 done 结束）：");  
 String input = scanner.nextLine();  
 if (input.equals("done")) {  
 break;  
 }  
 String[] parts = input.split(" ");  
 if (parts.length != 2) {  
 System.out.println("输入格式错误，请重新输入！");  
 continue;  
 }  
 char c = parts[0].charAt(0);  
 int freq = Integer.parseInt(parts[1]);  
 frequency.put(c, freq);  
 }  
 Node root = buildHuffmanTree(frequency);  
 Map<Character, String> huffmanCode = new HashMap<>();  
 encode(root, "", huffmanCode);  
 // 哈夫曼编码  
 System.out.println("哈夫曼编码：" + huffmanCode);  
  
 // 输入要解码的哈夫曼编码  
 System.out.print("输入要解码的哈夫曼编码：");  
 String encodedString = scanner.nextLine();  
 decode(root, encodedString); // 解码  
 System.out.print("请输入想要编码的字符串：");  
 String testString = scanner.next();  
 System.out.print("字符串 \"" + testString + "\" 的哈夫曼编码为：");  
 for (char c : testString.toCharArray()) {  
 System.out.print(huffmanCode.get(c));  
 }  
 }  
}