**成都信息工程大学计算机学院**

**课**

**程**

**实**

**验**

**报**

**告**

|  |  |
| --- | --- |
| **实验课程：** | **计算机体系结构** |
| **实验项目：** | **对指令操作码进行霍夫曼编码** |
| **指导教师：** | **文武** |
| **学生姓名:** | **XXX** |
| **学生学号：** | **XXXXXXXXX** |
| **班 级：** | **计科XXX** |
| **实验地点：** | **B213** |
| **实验时间：** | **2024 年 4 月20日 8:30 ~ 12:00** |
| **实验成绩：** |  |

**！！！报告抄袭和被抄袭者一律0分**

**一【上机实验内容】**

了解和掌握指令编码的基本要求和基本原理

使用编程工具编写一个程序，对一组指令进行霍夫曼编码，并输出最后的编码结果以及对指令码的长度进行评价。与扩展操作码和等长编码进行比较。

**二【上机实验中的其他它问题及心得】**

**遇见问题：**

**理解难度：Huffman编码算法涉及到优先队列和树的构建**

**编码实现：在实现Huffman树的过程中，正确地创建节点并管理它们的频率可能是一个挑战，尤其是要确保合并后的新节点频率是两个子节点频率之和。**

**错误处理：输入数据的异常情况处理，例如空数据或非法字符，可能会引起程序运行异常。**

**测试用例：编写覆盖各种情况的测试用例可能是一项挑战，需要仔细考虑边界条件和特殊情况。**

**心得体会：**

**理论与实践相结合：通过实验可以加深对于Huffman编码算法原理的理解，将理论知识转化为实际代码。**

**数据结构应用：学习如何在具体场景下运用数据结构，比如使用优先队列来辅助构建Huffman树。**

**调试技巧：在编码过程中不断测试和调试，提升了解决编程问题的能力。**

**编码规范：编写代码时更加注重编码风格和规范，提高代码的可读性和可维护性。**

**性能意识：意识到算法效率的重要性，了解如何分析和改进算法的性能。**

**三【源程序、测试数据及结果】**

|  |
| --- |
| HuffmanCode.java  **package** HuffmanByzxt;  **import** java.util.\*;  */\*\*  \* Created by zhangxiaotian on 2024/4/20.  \*/* **public class** HuffmanCode {   *//存储每个指令和其对应的概率（权重）<指令，权重>* **static** Map<Character, Integer> *map* = **new** HashMap<>();  *//存储每个指令和其Huffman编码 <指令，编码>* **static** Map<Character, String> *HuffmanCode* = **new** HashMap<>();   */\*\*  \* 对字符串进行编码  \*/* **public float** encode(String originalStr) {   *//若原始指令集为空，返回0* **if** (originalStr == **null** || originalStr.isEmpty()) {  **return** 0;  }   *//将原始数据转换为Node表* List<Node> leafNodes = **new** ArrayList<>();   *//建立Huffman树* buildTree(originalStr, leafNodes);   *//获得每个指令的Huffman编码 <指令，编码>* Map<Character, String> codeMap = HuffmanEncode(leafNodes);   *//打印* System.***out***.println(**"编码号：--概率 --编码"**);  codeMap.forEach((key,value)-> System.***out***.println(key+**" "** + *map*.get(key) +**" "**+ value));   *//计算Huffman编码的平均长度* **float** re = 0;  **for** (Map.Entry<Character, String> entry : codeMap.entrySet()) {  Character key = entry.getKey();*//获取指令名* String value = entry.getValue();*//获取指令对应的Huffman编码  //获取指令对应的概率* re += value.length() \* *map*.get(key);  }   **return** re/100;  }   */\*\*  \* 对每个字符进行编码，返回一个map集合  \*/* **private** Map<Character, String> HuffmanEncode(List<Node> leafNodes) {   **for** (Node leaf : leafNodes) {  Character character = leaf.getChars().charAt(0);  StringBuilder code = **new** StringBuilder();  Node currentNode = leaf;   **do** {  **if** (currentNode.isLeftChild()) {  code.append(**"0"**);  } **else** {  code.append(**"1"**);  }   currentNode = currentNode.getParent();   } **while** (currentNode.getParent() != **null**);   *HuffmanCode*.put(character, code.toString());  }  System.***out***.println();  **return** *HuffmanCode*;  }   */\*\*  \* 创建树的过程  \*/* **private** Tree buildTree(String originStr, List<Node> leafs) {   Map<Character, Integer> statistics = statistics(originStr.toCharArray());   Comparator<Node> comparator = **new** Comparator<Node>() {  @Override  **public int** compare(Node o1, Node o2) {  **return** o1.getWeight() - o2.getWeight();  }  };   *//通过有序队列，对头是最小的元素* PriorityQueue<Node> queue = **new** PriorityQueue<>(comparator);    **for** (Character c : statistics.keySet()) {   Node node = **new** Node();  node.setChars(c.toString());  node.setWeight(statistics.get(c));   leafs.add(node); *//每个字符都是叶子结点* queue.add(node);   }   **int** size = queue.size();  **for** (**int** i = 1; i < size; i++) { *//n - 1次* Node node1 = queue.poll();  Node node2 = queue.poll();   Node sumNode = **new** Node();  **if** (node1 != **null**) {  **if** (node2 != **null**) {  sumNode.setChars(node1.getChars() + node2.getChars());  }  }  **if** (node1 != **null**) {  **if** (node2 != **null**) {  sumNode.setWeight(node1.getWeight() + node2.getWeight());  }  }   sumNode.setLeftChild(node1);  sumNode.setRightChild(node2);   **if** (node1 != **null**) {  node1.setParent(sumNode);  }  **if** (node2 != **null**) {  node2.setParent(sumNode);  }   queue.add(sumNode);  }   Tree tree = **new** Tree();  tree.setRoot(queue.poll()); *//最后一次只剩下一个结点* **return** tree;  }   */\*\*  \* 通过频率设置权重  \*/* **private** Map<Character, Integer> statistics(**char**[] charArray) {   *// 输入每个指令的概率（即权重）* Scanner sc = **new** Scanner(System.***in***);   **for** (Character c : charArray) {  System.***out***.println(**"请输入指令"**+ c +**"的使用概率"**);  **float** n=sc.nextFloat();  *map*.put(c, (**int**) (n\*100));  }  **return** *map*;  }   */\*\*  \* 1-2-3-5扩展编码最短平均编码长度的计算  \*/* **public double** kuozhanEncode() {  *//1-2-3-5* **float** re = 0;  **for** (Map.Entry<Character, String> entry : *HuffmanCode*.entrySet()) {  Character key = entry.getKey();*//获取指令名* String value = entry.getValue();*//获取指令对应的Huffman编码  //获取指令对应的概率* **double** weigth = *map*.get(key);  **if**(value.length()==1 || value.length()==2 || value.length()==3 || value.length()==5){  re += (**float**) (value.length() \* weigth);  } **else** {  re += (**float**) ((value.length()+1) \* weigth);  }  }  **return** re/100;  } } |
| Test.java  **package** HuffmanByzxt.test;  **import** HuffmanByzxt.HuffmanCode;  **import** java.util.Scanner;  */\*\*  \* Created by zhangxiaotian on 2024/4/20.  \*/* **public class** Test {  **static** Scanner sc = **new** Scanner(System.in);  **public static void** main(String[] args) {   System.out.println(**"请输入指令集"**);  String str = sc.next();   *//实例化Huffman编码* HuffmanCode huffmanCode = **new** HuffmanCode();   *//Huffman* **double** huffman\_length = huffmanCode.encode(str);  System.out.printf(**"该指令集操作码Huffman编码的平均长度是：%.5f\n"**,huffman\_length);   *//等长编码* **double** q\_length = Math.log(str.length()) / Math.log(2);  System.out.printf(**"等长编码的平均长度是%.5f\n"**,q\_length);   *//1-2-3-5扩展编码* **double** kuo\_length = huffmanCode.kuozhanEncode();  System.out.printf(**"按1-2-3-5扩展编码的最短平均编码长度是%.2f\n"**,kuo\_length);   System.out.print( (huffman\_length<q\_length?**"Huffman"**:**"等长编码"**)  + **"的平均长度要比"** +  (huffman\_length>q\_length?**"Huffman"**:**"等长编码"**+ **"的平均长度短"**) + **"\n"**);   System.out.print( (huffman\_length<kuo\_length?**"Huffman"**:**"1-2-3-5扩展编码"**)  + **"的平均长度要比"** +  (huffman\_length>q\_length?**"Huffman"**:**"1-2-3-5扩展编码"**+ **"的最短平均长度短"**));  }  } |
| Node.java  **package** HuffmanByzxt;  */\*\*  \* Created by zhangxiaotian on 2024/4/20.  \*/* **public class** Node **implements** Comparable<Node> {   **private** String chars = **""**;  **private int** weight = 0;   **private** Node parent;  **private** Node leftChild;  **private** Node rightChild;   @Override  **public int** compareTo(Node o) {  **return** weight - o.getWeight();  }   **public boolean** isRoot() {  **return** parent == **null**;  }   **public boolean** isLeftChild() {  **return** parent != **null** && **this** == parent.leftChild;  }   **public boolean** isRightChile() {  **return** parent != **null** && **this** == parent.rightChild;  }   **public boolean** isLeaf() {  **return** leftChild == **null** && rightChild == **null**;  }   **public** String getChars() {  **return** chars;  }   **public void** setChars(String chars) {  **this**.chars = chars;  }   **public int** getWeight() {  **return** weight;  }   **public void** setWeight(**int** weight) {  **this**.weight = weight;  }   **public** Node getParent() {  **return** parent;  }   **public void** setParent(Node parent) {  **this**.parent = parent;  }   **public** Node getLeftChild() {  **return** leftChild;  }   **public void** setLeftChild(Node leftChild) {  **this**.leftChild = leftChild;  }   **public** Node getRightChild() {  **return** rightChild;  }   **public void** setRightChild(Node rightChild) {  **this**.rightChild = rightChild;  }  } |
| Tree.java  **package** HuffmanByzxt;  */\*\*  \* Created by zhangxiaotian on 2024/4/20.  \*/* **public class** Tree {   **private** Node root;   **public** Node getRoot() {  **return** root;  }   **public void** setRoot(Node root) {  **this**.root = root;  } } |

