

云南大学数学与统计学院

上机实践报告

课程名称：信息论基础实验
指导教师：陆正福
上机实践名称：Huffman 编码

年级：2013
姓名：金洋
学号：20131910023

上机实践成绩：
上机实践日期：2016/4/29
上机实践时间：14:16

上机实践编号：No. 4

组号：

一、实验目的

1. 进一步熟悉 Huffman 编码过程
2. 加深对最优码的理解

二、实验内容

1. 编程实现 Huffman 编码算法
2. 分析其中的树形数据结构

三、实验环境

1. 个人计算机，任意可以完成实验的平台，如 Java 平台、Python 语言、R 语言、Matlab 平台、Magma 平台等。
2. 对于信息与计算科学专业的学生，建议选择 Java、Python、R 等平台。
3. 对于非信息与计算科学专业的学生，建议选择 Matlab、Magma 等平台。

四、实验记录与实验结果分析

（注意记录实验中遇到的问题。实验报告的评分依据之一是实验记录的细致程度、实验过程的真实性、实验结果的解释和分析。如果涉及实验结果截屏，应选择白底黑字。）

1. Huffman 编码

Huffman 编码需要构造 Huffman 树。

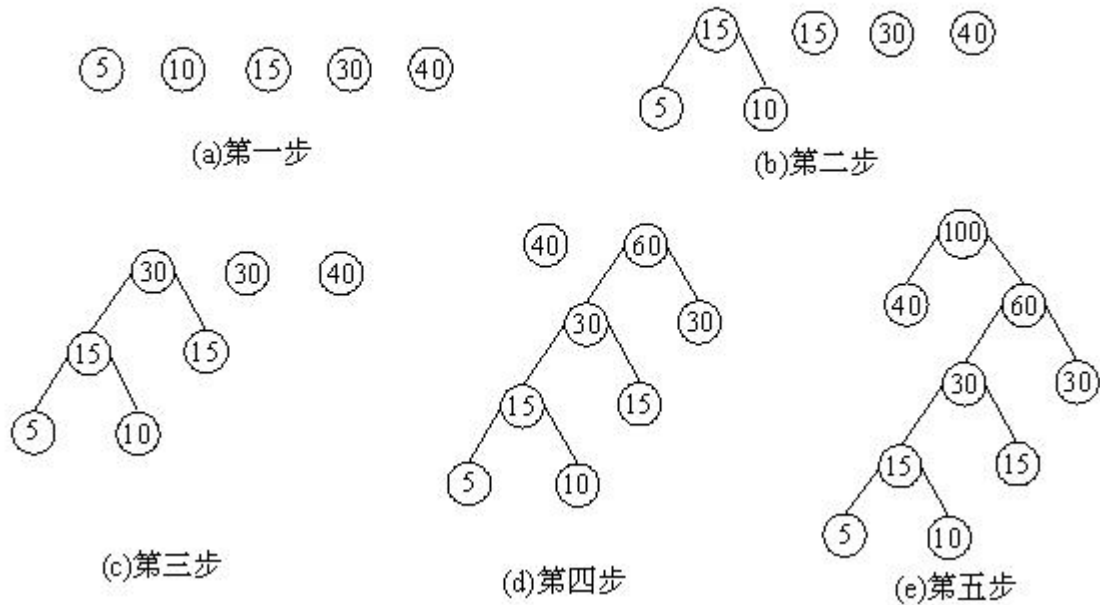
霍夫曼树是最优二叉树。树的带权路径长度规定为所有叶子结点的带权路径长度之和，带权路径长度最短的树，即为最优二叉树。在最优二叉树中，权值较大的结点离根较近。

首先就需要构建一个霍夫曼树，一般利用优先级队列来构造霍夫曼树。构造过程为：

首先，将字符按照频率插入一个优先级队列，频率越低越靠近队头，然后循环执行下面的操作：

- ①取出队头的两个树
- ②以它们为左右子节点构建一棵新树，新树的权值是两者之和
- ③将这棵新树插入队列

直到队列中只有一棵树时，这棵树就是我们需要的霍夫曼树，示意图如下



Node.java

package IT4;

//节点类

```
public class Node{
    private String key;           //树节点存储的关键字，如果是非叶子节点为空
    private int frequency;        //关键字词频
    private Node left;           //左子节点
    private Node right;          //右子节点
    private Node next;           //优先级队列中指向下一个节点的引用

    public Node(int fre,String str){ //构造方法 1
        frequency = fre;
        key = str;
    }
    public Node(int fre){ //构造方法 2
        frequency = fre;
    }

    public String getKey() {
        return key;
    }
    public void setKey(String key) {
        this.key = key;
    }
}
```

```
}
public Node getLeft() {
    return left;
}
public void setLeft(Node left) {
    this.left = left;
}
public Node getRight() {
    return right;
}
public void setRight(Node right) {
    this.right = right;
}
public Node getNext() {
    return next;
}
public void setNext(Node next) {
    this.next = next;
}
public int getFrequency() {
    return frequency;
}
public void setFrequency(int frequency) {
    this.frequency = frequency;
}
}
```

PriorityQueue.java

```
package IT4;
```

```
//用于辅助创建霍夫曼树的优先级队列
```

```
public class PriorityQueue{
    private Node first;
    private int length;

    public PriorityQueue(){
        length = 0;
        first = null;
    }

    //插入节点
    public void insert(Node node){
        if(first == null){ //队列为空
            first = node;
        }else{
```

```

        Node cur = first;
        Node previous = null;
        while(cur.getFrequency() < node.getFrequency()){ //定位要
插入位置的前一个节点和后一个节点
            previous = cur;
            if(cur.getNext() == null){ //已到达队尾
                cur = null;
                break;
            }else{
                cur = cur.getNext();
            }
        }
        if(previous == null){ //要插入第一个节点之前
            node.setNext(first);
            first = node;
        }else if(cur == null){ //要插入最后一个节点之后
            previous.setNext(node);
        }else{ //插入到两个节点之间
            previous.setNext(node);
            node.setNext(cur);
        }
    }
    length++;
}

//删除队头元素
public Node delete(){
    Node temp = first;
    first = first.getNext();
    length--;
    return temp;
}

//获取队列长度
public int getLength(){
    return length;
}

//按顺序打印队列
public void display(){
    Node cur = first;
    System.out.print("优先级队列: \t");
    while(cur != null){
        System.out.print(cur.getKey()+":"+cur.getFrequency()+"\t");
        cur = cur.getNext();
    }
}

```

```

        System.out.println();
    }

    //构造霍夫曼树
    public HuffmanTree buildHuffmanTree(){
        while(length > 1){
            Node hLeft = delete(); //取出队列的第一个节点作为新节点的左
子节点
            Node hRight = delete(); //取出队列的第二个节点作为新节点的右
子节点

            //新节点的权值等于左右子节点的权值之和
            Node hRoot = new
Node(hLeft.getFrequency()+hRight.getFrequency());
            hRoot.setLeft(hLeft);
            hRoot.setRight(hRight);
            insert(hRoot);
        }
        //最后队列中只剩一个节点，即为霍夫曼树的根节点
        return new HuffmanTree(first);
    }
}

```

HuffmanTree.java

```

package IT4;

import java.util.HashMap;
import java.util.Map;
//霍夫曼树类
public class HuffmanTree {
    private Node root;
    private Map codeSet = new HashMap(); //该霍夫曼树对应的字符编码集

    public HuffmanTree(Node root){
        this.root = root;
        buildCodeSet(root, ""); //初始化编码集
    }

    //生成编码集的私有方法，运用了迭代的思想
    //参数 currentNode 表示当前节点，参数 currentCode 代表当前节点对应的代码
    private void buildCodeSet(Node currentNode, String currentCode){
        if(currentNode.getKey() != null){
            //霍夫曼树中，如果当前节点包含关键字，则该节点肯定是叶子节

```

点，将该关键字和代码放入代码集

```

        codeSet.put(currentNode.getKey(), currentCode);
    }else{//如果不是叶子节点，必定同时包含左右子节点，这种节点没有对应
关键字
        //转向左子节点需要将当前代码追加 0
        buildCodeSet(currentNode.getLeft(), currentCode+"0");
        //转向右子节点需要将当前代码追加 1
        buildCodeSet(currentNode.getRight(), currentCode+"1");
    }
}

//获取编码集
public Map getCodeSet(){
    return codeSet;
}

}

```

TestHuffmanTree.java

```

package IT4;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
public class TestHuffmanTree {
    public static void main(String[] args) throws Exception{
        PriorityQueue queue = new PriorityQueue();
        Node n1 = new Node(1, "K");
        Node n2 = new Node(1, "U");
        Node n3 = new Node(1, "T");
        Node n4 = new Node(2, "Y");
        Node n5 = new Node(2, "E");
        Node n6 = new Node(2, "A");
        Node n7 = new Node(3, "I");
        Node n8 = new Node(4, "sp");
        Node n9 = new Node(5, "S");
        queue.insert(n1);
        queue.insert(n2);
        queue.insert(n3);
        queue.insert(n4);
        queue.insert(n5);
        queue.insert(n6);
        queue.insert(n7);
        queue.insert(n8);
        queue.insert(n9);
    }
}

```

```

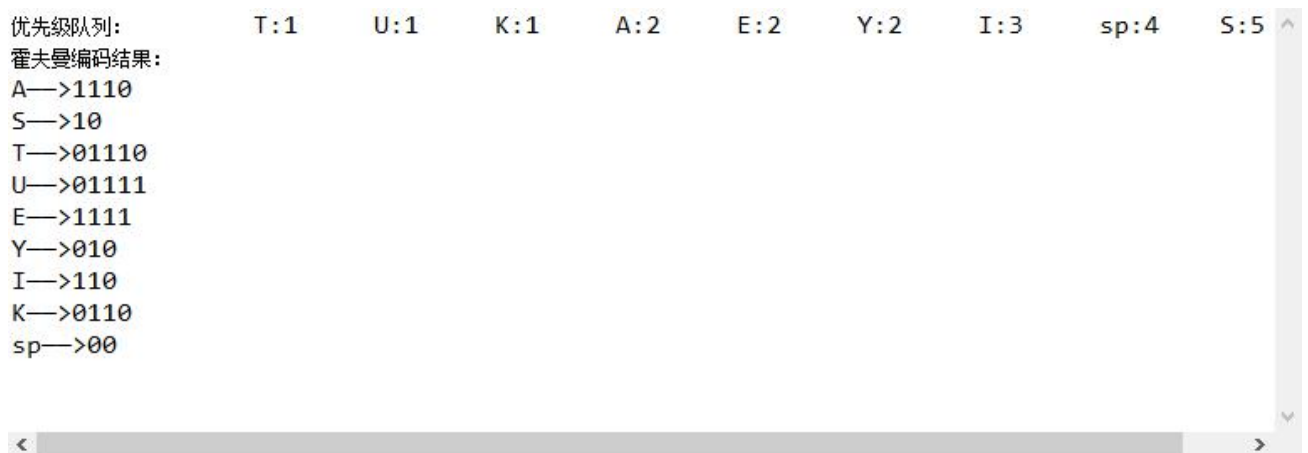
        queue.display();

        HuffmanTree tree =queue.buildHuffmanTree();
        Map map = tree.getCodeSet();
        Iterator it =map.entrySet().iterator();
        System.out.println("霍夫曼编码结果: ");
        while(it.hasNext()){
            Entry<String,String>entry = (Entry)it.next();
            System.out.println(entry.getKey()+"—>" +entry.getValue());
        }
    }
}

```

运行结果:

测试类中 Node 节点输入了字符和字符出现的次数



```

优先级队列:      T:1      U:1      K:1      A:2      E:2      Y:2      I:3      sp:4      S:5
霍夫曼编码结果:
A—>1110
S—>10
T—>01110
U—>01111
E—>1111
Y—>010
I—>110
K—>0110
sp—>00

```

2.一般利用优先级队列来构造霍夫曼树。构造过程为:

首先,将字符按照频率插入一个优先级队列,频率越低越靠近队头,然后循环执行下面的操作:

- ①取出队头的两个树
- ②以它们为左右子节点构建一棵新树,新树的权值是两者之和
- ③将这棵新树插入队列

直到队列中只有一棵树时,这棵树就是我们需要的霍夫曼树。

3.直接输入一段英文字符文本,给出对应的 Huffman 码

利用上面已经写出的代码来封装一个编码类，这个类的一个方法接受一个字符串消息，返回霍夫曼编码，此外，还有一个解码类，接受一段完整的霍夫曼编码，返回解码后的消息内容。这实际上就是压缩与解压的模拟过程。

```

请输入一段由英文小写字母组成的文本: jinyang
优先级队列:      j:1      y:1      i:1      g:1      a:1      n:2
代码集:
a—>00
g—>011
i—>010
y—>101
j—>100
n—>11

编码结果: 100010111010011011
解码结果: jinyang

```

```

请输入一段文本: Yunnan University
优先级队列:      y:1      Y:1      v:1      U:1      u:1      t:1      s:1      r:1      e:1      a:1      :1      i:2      n:4
代码集:
—>1110
a—>11111
e—>11110
i—>110
n—>10
r—>0101
s—>0100
t—>0111
U—>0001
u—>0110
v—>0000
y—>0010
Y—>0011

编码结果: 00110110101011111011100001101100000111100101010011001110010
解码结果: Yunnan University

```

输入长文本时，测试结果如下：

请输入一段文本: In its latest twice-yearly global assessment, the OECD warned that the world economy is “stuck in a low-growth trap”. The organisation said monetary policy alone could no longer be relied on to deliver growth and governments should be using the fiscal tools at their disposal, such as increases in investment spending, to stimulate demand. It also pointed to several downside risks to global growth, the most immediate of which would be if Britain votes to leave the European Union in a referendum on June 23rd.

```

优先级队列:      3:1      2:1      ”:1      “:1      U:1      T:1      O:1      J:1      D:1      C:1      B:1
               -:2      k:2      I:2      E:2      .:3      ,:4      f:4      y:5      b:5      v:6      p:6      c:9
               w:10     u:10     g:10     m:11     h:15     d:19     r:22     l:23     a:29     i:30     s:31     n:33
               t:38     o:38     e:47     :83

```

```

代码集:
B—>01000000
C—>010000011

```


D—>010000010
 E—>01111101
 I—>01111100
 J—>011110101
 O—>011110100
 T—>011110111
 U—>011110110
 “—>011110001
 ”—>011110000
 —>111
 a—>0101
 b—>1010011
 c—>101000
 d—>10101
 e—>001
 f—>0100111
 g—>110110
 h—>01110
 i—>0110
 k—>01111111
 l—>0001
 ,—>0100110
 -—>01111110
 m—>110111
 .—>0100001
 n—>1001
 o—>1100
 p—>010010
 r—>0000
 2—>011110011
 3—>011110010
 s—>1000
 t—>1011
 u—>110101
 v—>010001
 w—>110100
 y—>1010010

编码结果:

0111110010011110110101110001110001010110110011000101111110111101000110101
 0000010111111010100100010101000000011010010111110110000111001010011010100
 0111101011000100000110001000110111001100110110100110111101101110001111011
 1101000111110101000001101000001011111010001010000100100110101111101101110
 0101101111110110111000111111010011000000000110101111001101000110010011100
 1101111010010111011010001110111100011000101111010110100001111111011010011
 1101011110001110011010001111110110110000011001101001011011101111011000001
 0101001001111000001000011110111101110111000111111000000110110010110010110
 1000010110110110110010011111000010101101010111111011111001001001101101010
 0001010010111010010110000010110101000101001011101010001110010010011111010

```

0011001101010001101011111001110011100011100100111011000100001111010011001
1110000001000101100011010111111001001111101111001111010100100010110010001
001000011111011000001100110100101101110111010110011010111110110110001000
1001000010011101110011001101110001111000011101100110101000110101111101001
1001111110101100001101001110110111101101110001111010011101101000101000010
100011111011110011000001100011101011011111011011100010110000011110101011
0100001001011001000010100010100110111100011010110100001110111010110001110
1101001101000000000101011000001100011101101001111011010010100010011000101
1110111001100110111111000010010001100110101011010011101100100110111101111
00111100010110110111111010100010101101100111110101001110111010110011010
1010000111101111100101111101010001100011001110100101100011010011011001101
0111110111100111100000101000100100000101000111110101110011010010011000011
010101001111000001101000011111110001111011110011111011000011100101001101
0100011111101100000110011010010110111001001101111011011100011111101111100
10001011111011011011111011100110101100101101100111111000100111111110100
0111001101010000111011111010011001101010001101011111010011001111011001001
1111101000000000000110101101011010011110100011100101100111000111101111001
1100010010101010001001111101101110001111011111011101010000110001001000101
0110011110111101101001011011001001111011010011110101111000000101001110010
000001100110101110101110111111100100111101111010111010110010011110111100
110111100100000101010100001

```

解码结果: In its latest twice-yearly global assessment, the OECD warned that the world economy is “stuck in a low-growth trap”. The organisation said monetary policy alone could no longer be relied on to deliver growth and governments should be using the fiscal tools at their disposal, such as increases in investment spending, to stimulate demand. It also pointed to several downside risks to global growth, the most immediate of which would be if Britain votes to leave the European Union in a referendum on June 23rd.

五、实验体会

(请认真填写自己的真实体会)

1. 霍夫曼树是最优二叉树。树的带权路径长度规定为所有叶子结点的带权路径长度之和，带权路径长度最短的树，即为最优二叉树。在最优二叉树中，权值较大的结点离根较近。
2. 对于相同心愿字母表的任意其他编码，不可能比 Huffman 算法构造出的编码具有更小的期望长度。
3. 在电文传输中，需要将电文中出现的每个字符进行二进制编码。在设计编码时需要遵守两个原则：

(1) 发送方传输的二进制编码，到接收方解码后必须具有唯一性，即解码结果与发送方发送的电文完全一样；

(2) 发送的二进制编码尽可能地短。有两种编码的方式：

(1) 等长编码

这种编码方式的特点是每个字符的编码长度相同（编码长度就是每个编码所含的二进制位数）。假设字符集只含有 4 个字符 A, B, C, D, 用二进制两位表示的编码分别为 00, 01, 10, 11。若现在有一段电文为：ABACCDA，则应发送二进制序列：00010010101100，总长度为 14 位。当接收方接收到这段电文后，将按两位一段进行译码。这种编码的特点是译码简单且具有唯一性，但编码长度并不是最短的。

(2) 不等长编码

在传送电文时，为了使其二进制位数尽可能地少，可以将每个字符的编码设计为不等长的，使用频度较高的字符分配一个相对较短的编码，使用频度较低的字符分配一个比较长的编码。例如，可以为 A, B, C, D 四个字符分别分配 0, 00, 1, 01，并可将上述电文用二进制序列：000011010 发送，其长度只有 9 个二进制位，但随之带来了一个问题，接收方接到这段电文后无法进行译码，因为无法断定前面 4 个 0 是 4 个 A, 1 个 B、2 个 A，还是 2 个 B，即译码不唯一，因此这种编码方法不可使用。

因此，为了设计长短不等的编码，以便减少电文的总长，还必须考虑编码的唯一性，即在建立不等长编码时必须使任何一个字符的编码都不是另一个字符的前缀，这宗编码称为前缀编码（prefix code）

六、参考文献

1. （主讲课英文教材）
- 2 （数据结构与算法课程的英文教材）
3. 冰河. 数据结构之霍夫曼树 [EB/OL]. <http://www.2cto.com/kf/201412/363776.html>, 2014-12-23.