# 云南大学数学与与统计学院

# 上机实践报告

| | | |
|---|---|---|
| **课程名称**：信息论基础实验 | **年级**：2013 | **考试成绩**： |
| **指导教师**：陆正福 | **姓名**：金洋 | |
| **上机实践名称**：**IT 期中考试实验** | **学号**：20131910023 | |
| **分组成员（学号-姓名）**：<br><br>**组号**： | | |

## 一、实验目的

应用所学信息论的基本知识和技能，完成数据的压缩处理、加密处理、差错控制保护。

## 二、实验内容

任取一段数据（记为 MyData）， 编写程序，实现下列功能:
（1）实现对数据 MyData 的无损压缩，得到压缩数据（记为 MyCompData）；
（2）实现对 MyCompData 的加密 ，得到加密的压缩数据（记为 MyEncCompData）；
（3）实现对 MyEncCompData 的差错控制，得到受差错控制保护的加密的压缩数据（记为 MyEccEncCompData）。

## 三、实验平台

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。**）

任取一段数据（记为 MyData）， 为了简单并能说明问题，我们将 MyData 设为 String 型，字符串内容为 ”jin yang”；

编写程序，实现下列功能:

（1）实现对数据 MyData 的无损压缩，得到压缩数据（记为 MyCompData）；

数据压缩的原理是对数据源最频繁出现的结果分配较短的描述，而对不经常出现的结果分配较长的描述时，从而达到数据压缩的目的。

要做到无损压缩，码长的最优期望长度为数据的熵，即压缩下限是熵。关于给定分布构造最优（最短期望长度）前缀码，Huffman 给出了一个简单的算法，可以证明，对于相同心愿字母表的任意其他编码，不可能比 Huffman 算法构造的编码具有更短的期望长度。

Huffman 编码需要构造 Huffman 树。

Huffman 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 Huffman 树，一般利用优先级队列来构造 Huffman 树。构造过程为：

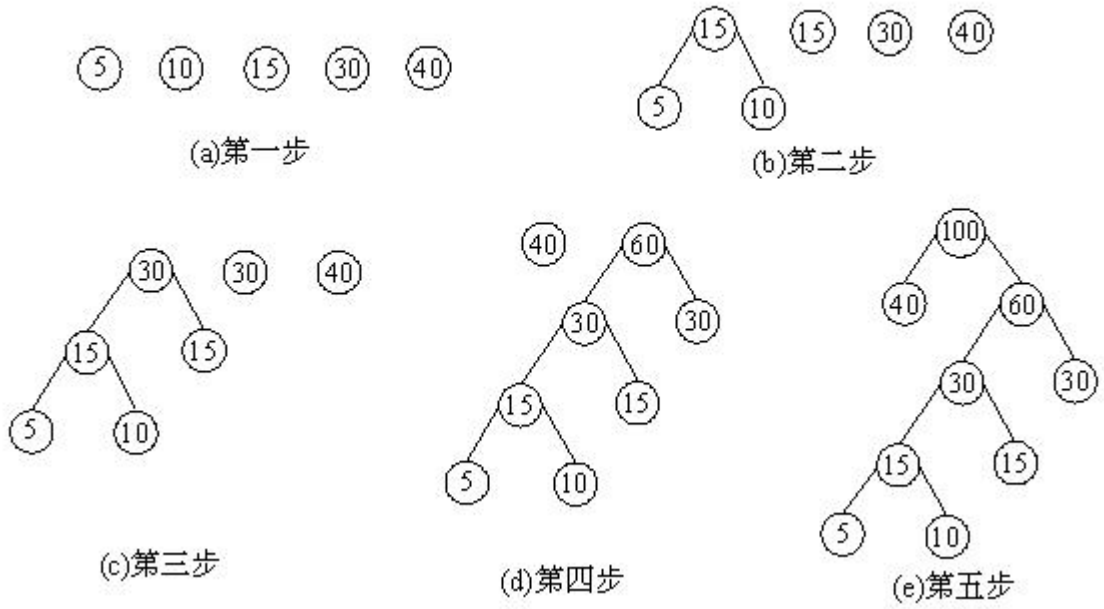
首先，将字符按照频率插入一个优先级队列，频率越低越靠近队头，然后循环执行下面的操作：

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 Huffman 树，示意图如下




**Node.java**

```
package MidTest;
```

```
//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 MidTest;

//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 MidTest;
```

```
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 MidTest;
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,"j");
        Node n2 = new Node(1,"i");
        Node n3 = new Node(2,"n");
        Node n4 = new Node(1,"y");
        Node n5 = new Node(1,"a");
        Node n6 = new Node(1,"g");
        Node n7 = new Node(1,"sp");
        queue.insert(n1);
        queue.insert(n2);
        queue.insert(n3);
        queue.insert(n4);
        queue.insert(n5);
        queue.insert(n6);
        queue.insert(n7);

        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());
        }
 }
}
```

测试结果：

统计字符串”jin yang” 中每个字符出现的次数，并加入优先队列

```
Node n1 = new Node(1,"j");
        Node n2 = new Node(1,"i");
        Node n3 = new Node(2,"n");
        Node n4 = new Node(1,"y");
        Node n5 = new Node(1,"a");
        Node n6 = new Node(1,"g");
        Node n7 = new Node(1,"sp");
        queue.insert(n1);
        queue.insert(n2);
        queue.insert(n3);
        queue.insert(n4);
        queue.insert(n5);
        queue.insert(n6);
```

```
        queue.insert(n7);
```

```
优先级队列:          sp:1    g:1     a:1     y:1     i:1     j:1     n:2
霍夫曼编码结果:
a—>110
g—>001
i—>100
y—>111
j—>101
sp—>000
n—>01
```

这样原本的”jin yang”为 64bit，现在转换为二进制”10110001000111111001001”共 22bit，且实现了无损压缩。

MyCompData=101100 01000111 11001001.（二进制）

经过查阅 acsii 码表即为字符串 MyCompData=”,GÉ”；

2. 实现对 MyCompData 的加密 ，得到加密的压缩数据（记为 MyEncCompData）；

引用上学期计算机网络-ap3.1 的加密程序：

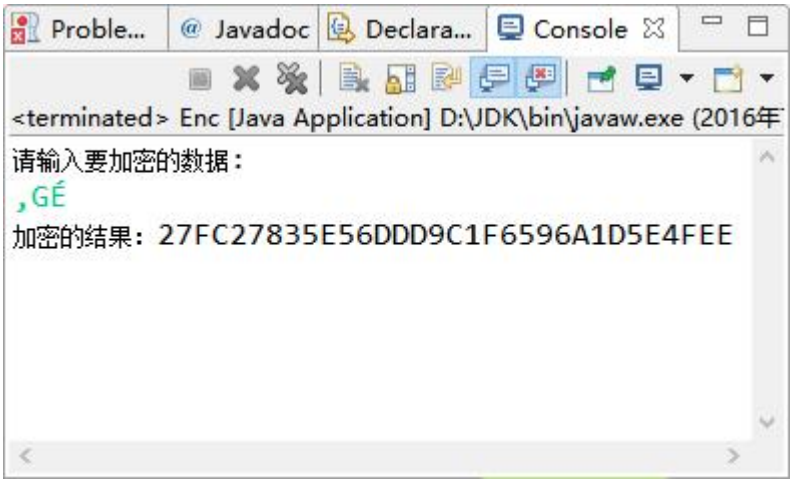
MyCompData=1011000100011111001001 是一串二进制比特流，查阅 acsii 码表即为字符串”,GÉ”

**Enc.java**

```
package MidTest;
import java.io.*;//包含了 java 输入和输出流的包
import java.util.Scanner;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;


/**
 *
 * @author 金洋
 *
```

```
 */
public class Enc {

	public static void main(String[] args) throws Exception{

		BufferedReader input=new BufferedReader(new
InputStreamReader(System.in));
		String message=new String();
		String MyEncCompData=new String();
		System.out.println("请输入要加密的数据:");
		message=input.readLine();
	  /*加密数据*/
	  byte[] encryptResult = encrypt(message, "12345678"); //加密
password,第二个参数是加密密钥
		String encryptResultStr = parseByte2HexStr(encryptResult); //
将加密后的数组转化为字符串便于输出.但是不能强制转换，需要将二进制字节数组转化为
十六进制字符串

		MyEncCompData=encryptResultStr;
		System.out.println("加密的结果: "+encryptResultStr);

	}



	/**将二进制转换成 16 进制
	 * @param buf
	 * @return
	 */
	public static String parseByte2HexStr(byte buf[]) {
		StringBuffer sb = new StringBuffer();
		for (int i = 0; i < buf.length; i++) {
			String hex = Integer.toHexString(buf[i] & 0xFF);
			if (hex.length() == 1) {
				hex = '0' + hex;
			}
			sb.append(hex.toUpperCase());
		}
		return sb.toString();
	}



	/**
	 * 加密
	 *
```

```
     * @param content 需要加密的内容
     * @param password  加密密钥
     * @return
     */
    public static byte[] encrypt(String content, String password) {
            try {
                    KeyGenerator kgen = KeyGenerator.getInstance("AES");
                    kgen.init(128, new SecureRandom(password.getBytes()));
                    SecretKey secretKey = kgen.generateKey();
                    byte[] enCodeFormat = secretKey.getEncoded();
                    SecretKeySpec key = new SecretKeySpec(enCodeFormat,
"AES");
                    Cipher cipher = Cipher.getInstance("AES");// 创建密码
器
                    byte[] byteContent = content.getBytes("utf-8");
                    cipher.init(Cipher.ENCRYPT_MODE, key);// 初始化
                    byte[] result = cipher.doFinal(byteContent);
                    return result; // 加密
            } catch (NoSuchAlgorithmException e) {
                    e.printStackTrace();
            } catch (NoSuchPaddingException e) {
                    e.printStackTrace();
            } catch (InvalidKeyException e) {
                    e.printStackTrace();
            } catch (UnsupportedEncodingException e) {
                    e.printStackTrace();
            } catch (IllegalBlockSizeException e) {
                    e.printStackTrace();
            } catch (BadPaddingException e) {
                    e.printStackTrace();
            }
            return null;
    }
}
```




即 MyEncCompData=$(27FC27835E56DDD9C1F6596A1D5E4FEE)_{16}$

经过查阅 acsii 码表即为字符串 MyEncCompData="ƒü'ƒ^VÝÙÁöYj ^Oî"；

(3) 实现对 MyEncCompData 的差错控制，得到受差错控制保护的加密的压缩数据（记为 MyEccEncCompData）。

使用循环冗余校验来对 MyEncCompData 进行差错控制。
先计算正确的数据的校验值，然后修改部分字符来判断后来的字符串是否出现差错。若出现差错则要求重传。

**CRC.java**

```
package MidTest;
import java.io.*;
public class CRC {


      public static void main(String[] args) throws IOException {

            System.out.println("请输入 MyEccEncCompData:");
            BufferedReader input=new BufferedReader(new
InputStreamReader(System.in));

            String MyEccEncCompData=new String();
            MyEccEncCompData=input.readLine();

            int crc2 = FindCRC(MyEccEncCompData.getBytes());
            String crc16 = Integer.toHexString(crc2);//把 10 进制的结果转化为
16 进制
            //如果想要保证校验码必须为 2 位，可以先判断结果是否比 16 小，如果比 16
```

```
小，可以在 16 进制的结果前面加 0
            if(crc2 < 16 ){
                  crc16 = "0"+crc16;
            }
            System.out.println(crc16);

            if (!crc16.equals("84"))
                  System.out.println("数据错误，重传！");
      }
      public static int FindCRC(byte[] data){
            int CRC=0;
            int genPoly = 0Xaa;
            for(int i=0;i<data.length; i++){
                  CRC ^= data[i];
                  for(int j=0;j<8;j++){
                        if((CRC & 0x80) != 0){
                              CRC = (CRC << 1) ^ genPoly;
                        }else{
                              CRC <<= 1;
                        }
                  }
            }
            CRC &= 0xff;//保证 CRC 余码输出为 2 字节。
            return CRC;
      }

}
```

测试结果


即 CRC 校验码为 84；

若对 MyEncCompData 进行改动
①增添：ƒü'ƒ^VÝÙÁöYj ^Oîjinyang


②删除：ƒü'ƒ^VÝÙÁöYj_x001D_^O


③修改


可见程序对目前为止的输入改动数据都作出了差错检验.

**五、实验体会**

**（请认真填写自己的真实体会）**

1.String 和 byte[]转换可以按如下方式进行

①string 转 byte[]

```
String str = "Hello";
byte[] srtbyte = str.getBytes();
```

②byte[] 转 string

```
byte[] srtbyte;
String res = new String(srtbyte);
System.out.println(res);
```

③设定编码方式相互转换

```
String str = "hello";
byte[] srtbyte = null;
try {
   srtbyte = str.getBytes("UTF-8");
   String res = new String(srtbyte,"UTF-8");
   System.out.println(res);
} catch (UnsupportedEncodingException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
```

2. 加密后的 byte 数组是不能强制转换成字符串的，换言之：字符串和 byte 数组在这种情况下不是互逆的；要避免这种情况，我们需要做一些修订，可以考虑将二进制数组转换成十六进制字符串，将十六进制字符串转换为二进制数组。

**六、参考文献**

1. 主讲课英文教材

***2. https://1024tools.com/ascii***

2. 我夕. java 对称加密——直接代码中加密. [EB/OL]. [2012-05-15]. http://blog.csdn.net/sdefzhpk/article/details/7568777

3.hbcui1984. JAVA 实现 AES 加密. [EB/OL]. [2010-01-16]. http://blog.csdn.net/hbcui1984/article/details/5201247