实验报告电子版

**数据结构实训**

**——一维数字信号的霍夫曼编码**

姓 名： 刘远明

学 号： 2220212113

指导老师： 韩凤

实验日期： 2024年4月8日

**大连海事大学电子信息科学技术专业**

**Academic Honesty Violations in Practice of Data Structures (PDS)**

In **PDS**, the MINIMUM penalty recommended for a violation of the Academic Honesty Policy will be a ZERO ON THE ASSIGNMENT, PROJECT or EXAM and a LOWERING OF YOUR FINAL GRADE for below what is otherwise earned. You may NOT withdraw from the course if found guilty. Some examples of academic misconduct in **PDS** include but are not limited to the following actions:

1. Picking up and using or discarding another student's written or computer output;

2. Using the computer account of another student;

3. Representing as one's own the work of another on assignments, quizzes, and projects;

4. Giving another student a copy of one's work on an assignment before the due date.

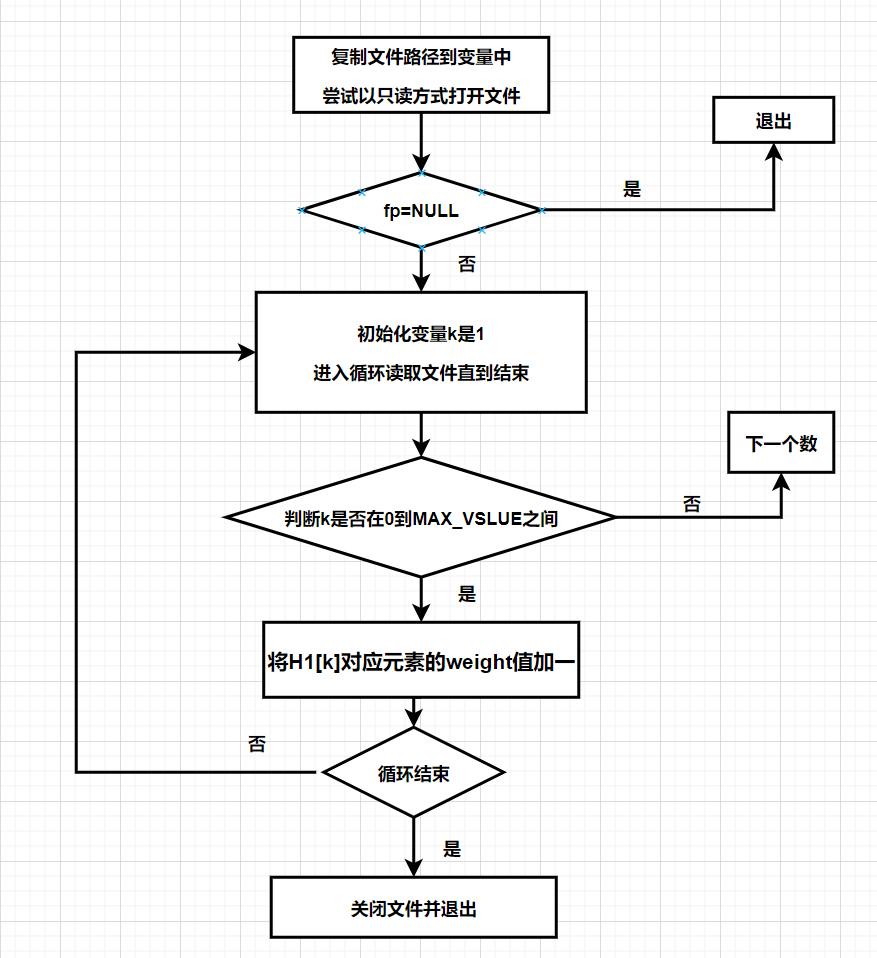
5. Copying work from online resources (Baidu,Chegg, google forums, etc.)

6. Posting work to online resources where other students can view your work.

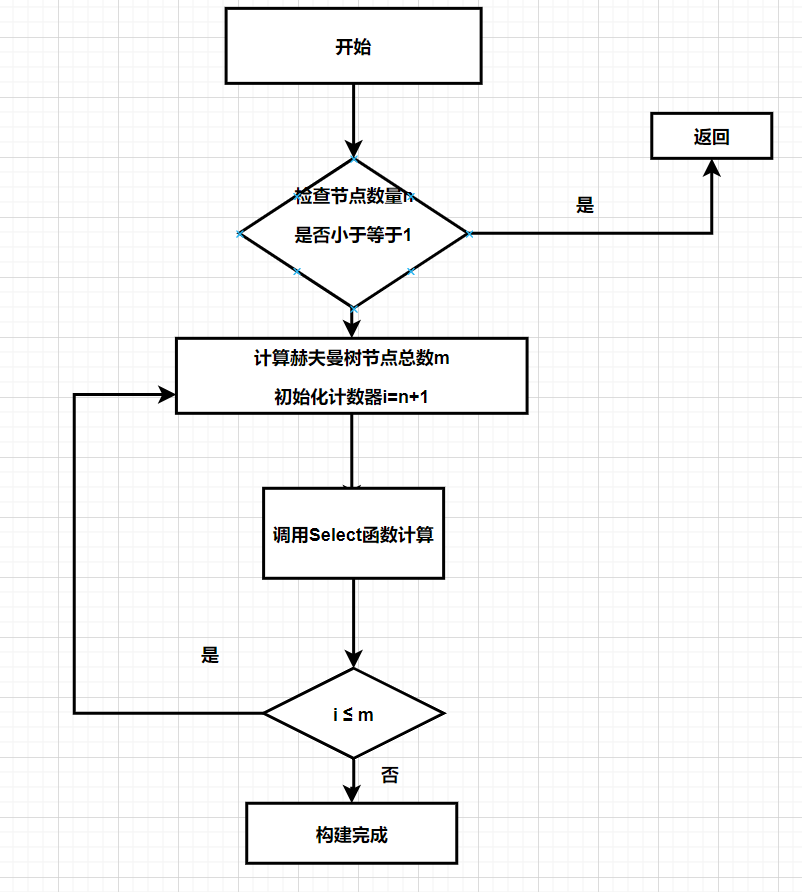
All submissions will be checked for similarity. This code will check each submission for similarity between other student submissions, past student submissions, the solution manual, and online resources and postings. If your submission is flagged for a high level of similarity, it will be turned in for an academic honesty violation if deemed appropriate.

NOTE: Changing variable names, adding comments, or spacing will l result in a violation.

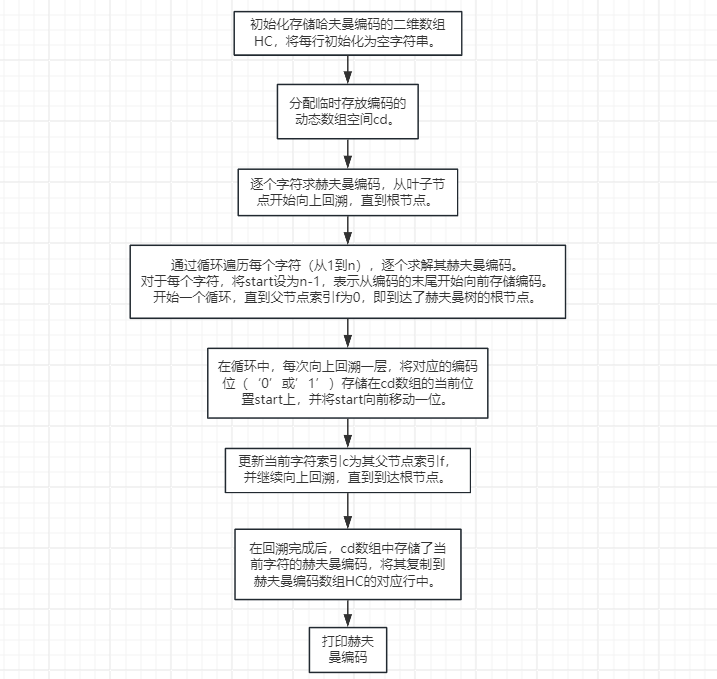
1. **实验目的**
2. 深入理解掌握树的基本概念、二叉树的基本操作、赫夫曼树的含义及其应用。
3. 掌握二叉树（赫夫曼树）生成、遍历等算法的实现。
4. 掌握利用赫夫曼算法对信号进行编解码的实现方法。
5. **实验内容与要求**
6. 设计函数实现对一维信号进行直方图统计。
7. 设计函数实现赫夫曼树构建。
8. 设计函数实现赫夫曼码书构建。
9. 设计函数实现对一维信号的赫夫曼编码与解码。
10. **算法描述与流程**
11. 设计函数实现对一维信号进行直方图

****

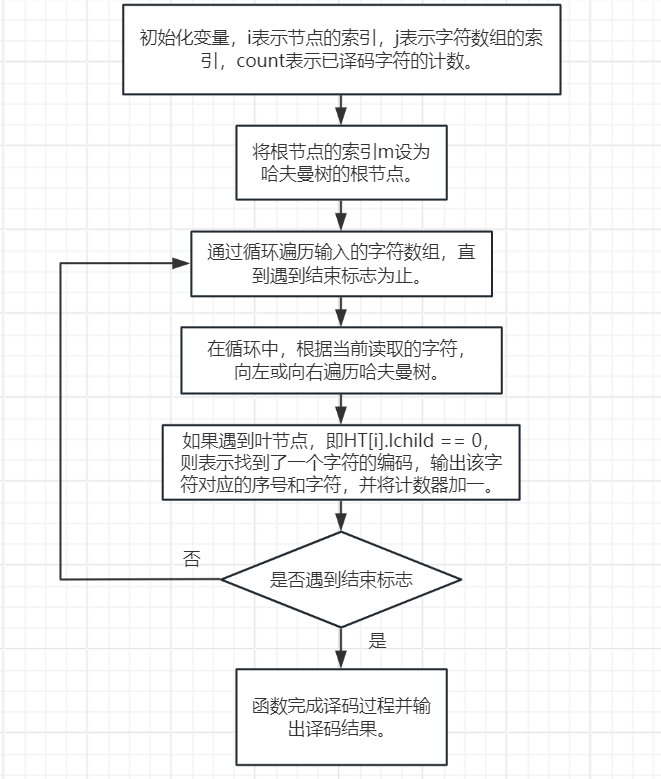
1. 设计函数实现赫夫曼码书的构建

****

1. 设计函数实现对一维信号的赫夫曼编码



1. 设计函数实现对一维信号的赫夫曼解码



1. **算法实现**

**// 标准输入输出库**

**#include <stdio.h>**

**// 标准库**

**#include <stdlib.h>**

**// 内存分配库**

**#include <malloc.h>**

**// 字符串操作库**

**#include <string.h>**

**// 输入输出流库**

**#include <iostream>**

**// 定义 OK 为 0**

**#define OK 0**

**// 定义 ERROR 为 -1**

**#define ERROR -1**

**// 定义最大数组大小为 100**

**#define MAXSIZE 100**

**// 使用标准命名空间**

**using namespace std;**

**// 定义哈夫曼树节点结构体 HTNode**

**typedef struct {**

**int data; // 节点数据**

**int weight; // 节点权重**

**int parent, lch, rch; // 父节点、左子节点、右子节点索引**

**} HTNode, \*HuffmanTree; // 定义哈夫曼树指针类型 HuffmanTree**

**// 定义哈夫曼编码表指针类型 HuffmanCode**

**typedef char\* \*HuffmanCode;**

**// 初始化哈夫曼树函数**

**void InitHuffmanTree(HuffmanTree& HT, int n) {**

**if (n <= 1) return; // 如果 n 小于等于 1，则返回**

**int m = 2 \* n - 1; // 计算哈夫曼树总节点数**

**HT = new HTNode[m + 1]; // 分配哈夫曼树数组内存**

**// 初始化哈夫曼树的每个节点**

**for (int i = 1; i <= m; ++i) {**

**HT[i].parent = 0; // 父节点索引初始化为 0**

**HT[i].lch = 0; // 左子节点索引初始化为 0**

**HT[i].rch = 0; // 右子节点索引初始化为 0**

**}**

**}**

**// 选择哈夫曼树中的两个最小节点函数**

**void select(HuffmanTree& HT, int n, int& s1, int& s2) {**

**int min1 = MAXSIZE, min2 = MAXSIZE; // 初始化 min1 和 min2 为最大值 MAXSIZE**

**// 查找哈夫曼树中的两个最小权重节点**

**for (int i = 1; i <= n; i++) {**

**if (HT[i].weight < min1 && HT[i].parent == 0) {**

**min1 = HT[i].weight;**

**s1 = i;**

**}**

**}**

**int temp = HT[s1].weight; // 暂存 s1 的权重**

**HT[s1].weight = MAXSIZE; // 将 s1 的权重设为最大值，避免重新选择**

**// 查找哈夫曼树中的第二个最小权重节点**

**for (int i = 1; i <= n; i++) {**

**if (HT[i].weight < min2 && HT[i].parent == 0) {**

**min2 = HT[i].weight;**

**s2 = i;**

**}**

**}**

**HT[s1].weight = temp; // 恢复 s1 的原始权重值**

**}**

**// 创建哈夫曼树函数**

**void CreatHuffmanTree(HuffmanTree& HT, int n) {**

**int m, s1, s2;**

**if (n <= 1) return; // 如果 n 小于等于 1，则返回**

**m = 2 \* n - 1; // 计算哈夫曼树总节点数**

**// 通过 n-1 次选择和合并操作来创建哈夫曼树**

**for (int i = n + 1; i <= m; ++i) {**

**// 选择当前哈夫曼树中的两个最小节点**

**select(HT, i - 1, s1, s2);**

**HT[s1].parent = i; // 设置 s1 的父节点为 i**

**HT[s2].parent = i; // 设置 s2 的父节点为 i**

**HT[i].lch = s1; // 设置 i 的左子节点为 s1**

**HT[i].rch = s2; // 设置 i 的右子节点为 s2**

**HT[i].weight = HT[s1].weight + HT[s2].weight; // 计算 i 的权重为左右子节点权重之和**

**}**

**HT[0].data = m; // 将根节点的编号存入 HT[0].data 中**

**}**

**// 创建哈夫曼编码表函数**

**void CreatHuffmanCode(HuffmanTree HT, HuffmanCode& HC, int n) {**

**int i, start, c, f;**

**char \*cd;**

**HC = (HuffmanCode)malloc((n + 1) \* sizeof(char\*)); // 分配 n 个字符编码的头指针变量**

**cd = (char\*)malloc(n \* sizeof(char)); // 分配求编码的工作空间**

**cd[n - 1] = '\0'; // 设置编码结束符位置**

**// 逐个字符求哈夫曼编码**

**for (i = 1; i <= n; i++) {**

**start = n - 1; // 设置起始位置**

**for (c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent) {**

**// 从叶子到根逆向求每个字符的哈夫曼编码**

**if (HT[f].lch == c)**

**cd[--start] = '0';**

**else**

**cd[--start] = '1';**

**}**

**HC[i] = (char\*)malloc((n - start) \* sizeof(char)); // 分配存储编码的空间**

**strcpy(HC[i], &cd[start]); // 将编码拷贝到哈夫曼编码表中**

**}**

**free(cd); // 释放编码工作空间**

**}**

**// 显示哈夫曼编码并存储函数**

**void showHuffmanTree(HuffmanTree HT, HuffmanCode HC, int length, int\* data, char\* show) {**

**int m = 0, index = 0;**

**for (int i = 0; i < length; i++) {**

**int j = 1;**

**// 查找数据对应的哈夫曼编码**

**while (data[i] != HT[j].data) {**

**j++;**

**}**

**// 将哈夫曼编码存储到 show 中**

**for (int count = 0; count < sizeof(HC[j]) / sizeof(char) && HC[j][count]; count++) {**

**show[index++] = HC[j][count];**

**}**

**m++;**

**}**

**show[index] = '#'; // 添加编码结束符**

**// 打印哈夫曼编码**

**for (int x = 0; x < index; x++) {**

**printf("%c", show[x]);**

**}**

**}**

**// 哈夫曼解码函数**

**void decodeHuffmanTree(HuffmanTree HT, char\* show) {**

**int i = HT[0].data;**

**int j = 0;**

**char endflag = '#';**

**// 逐个字符解码**

**while (show[j] != '#') {**

**if (show[j] == '0')**

**i = HT[i].lch;**

**else if (show[j] == '1') {**

**i = HT[i].rch;**

**}**

**// 判断是否为叶子节点，如果是则输出数据**

**if (HT[i].lch == 0) {**

**printf("%d ", HT[i].data);**

**i = HT[0].data;**

**}**

**j++;**

**}**

**printf("\n");**

**}**

**// 读取文件并进行频数统计函数**

**int read(char\* name, int n, int\* data, int\* metaData, int\* weight, int& length) {**

**for (int x = 0; x < n; x++) {**

**data[x] = -1; // 初始化数据数组为 -1**

**}**

**int i = 0, x = 0;**

**int result = 0;**

**FILE\* fp = fopen(name, "r"); // 打开文件**

**char c[50];**

**if (fp == NULL) {**

**puts("不能打开文件！\n"); // 输出错误信息**

**return ERROR;**

**}**

**if (fgets(c, sizeof(c), fp)) {**

**char\* p = strtok(c, ",");**

**if (p) {**

**result = atoi(p); // 将字符串转换为整数**

**data[i] = result;**

**metaData[x] = result;**

**length++;**

**weight[i] = 1;**

**}**

**while (p = strtok(NULL, ",")) {**

**int flag = 0;**

**result = atoi(p); // 将字符串转换为整数**

**metaData[++x] = result;**

**length++;**

**// 统计频数**

**for (int j = 0; j < n; j++) {**

**if (result == data[j]) {**

**weight[j]++;**

**flag = 1;**

**break;**

**}**

**}**

**if (flag == 0) {**

**i++;**

**data[i] = result;**

**weight[i] = 1;**

**}**

**}**

**}**

**fclose(fp); // 关闭文件**

**return OK;**

**}**

**// 将字符串转换为整数函数**

**int atoi(char\* str) {**

**int value = 0;**

**while (\*str >= '0' && \*str <= '9') {**

**value \*= 10;**

**value += \*str - '0';**

**str++;**

**}**

**return value;**

**}**

**// 冒泡排序函数，对数据和频数进行排序**

**void bubble\_sort(int\* data, int\* weight, int size) {**

**int i = 0;**

**int j = 0;**

**int temp = 0;**

**for (i = 0; i < size - 1; i++) {**

**for (j = 0; j < size - 1 - i; j++) {**

**if (data[j] > data[j + 1]) {**

**// 交换数据和频数**

**temp = data[j];**

**data[j] = data[j + 1];**

**data[j + 1] = temp;**

**temp = weight[j];**

**weight[j] = weight[j + 1];**

**weight[j + 1] = temp;**

**}**

**}**

**}**

**}**

**// 主函数**

**int main() {**

**int n = 5; // 定义数组大小**

**int length = 0; // 定义长度变量**

**int input;**

**char c[50];**

**int\* data = (int\*)malloc(n \* sizeof(int)); // 分配数据数组内存**

**int\* weight = (int\*)malloc(n \* sizeof(int)); // 分配频数数组内存**

**int\* metaData = (int\*)malloc(20 \* sizeof(int)); // 分配元数据数组内存**

**HuffmanTree HT;**

**InitHuffmanTree(HT, n); // 初始化哈夫曼树**

**read("E:\\zuoye\\2source.txt", n, data, metaData, weight, length); // 读取文件并进行频数统计**

**// 输出信源数据**

**printf("信源数据如下：\n");**

**for (int i = 0; i < length; i++) {**

**printf("%d ", metaData[i]);**

**}**

**// 对数据进行冒泡排序并进行直方图统计**

**bubble\_sort(data, weight, n);**

**printf("\n\n一维数组直方图统计+冒泡排序:\n");**

**for (int i = 1; i <= n; i++) {**

**HT[i].data = data[i - 1];**

**HT[i].weight = weight[i - 1];**

**printf("信源符号为：%d, 权重为：%d\n", HT[i].data, HT[i].weight);**

**}**

**// 输出码书**

**printf("\n码书为：\n");**

**HuffmanCode HC;**

**CreatHuffmanTree(HT, n);**

**CreatHuffmanCode(HT, HC, n);**

**for (int i = 1; i <= n; i++)**

**printf("信源符号为：%d，码字为：%s\n", HT[i].data, HC[i]);**

**// 编码**

**printf("请输入需要编码的信源（范围0~4,以逗号分隔）,若不需要请输入-1，系统默认输出信源文件的编码及转换。\n");**

**gets(c);**

**char\* show = (char\*)malloc(200);**

**if (strcmp(c, "-1") != 0) {**

**length = 0;**

**char\* p = strtok(c, ",");**

**int num = 0;**

**if (p) {**

**if (atoi(p) < 0 || atoi(p) > 4) {**

**printf("输入信源范围有误，范围应在0~4\n");**

**return ERROR;**

**}**

**metaData[num] = atoi(p);**

**length++;**

**}**

**while (p = strtok(NULL, ",")) {**

**if (atoi(p) < 0 || atoi(p) > 4) {**

**printf("输入信源范围有误，范围应在0~4\n");**

**return ERROR;**

**}**

**metaData[++num] = atoi(p);**

**length++;**

**}**

**}**

**printf("\n码流为：\n");**

**showHuffmanTree(HT, HC, length, metaData, show);**

**// 解码**

**printf("\n\n解码后：\n");**

**decodeHuffmanTree(HT, show);**

**return OK;**

**}**

1. **实验结果与分析**

**5.1测试用例**

**1 1 0 2 2 2 3 3 3 3 3 4 4 4 4 4 4 4 4**

**5.2测试程序**

*int main() {*

*int n = 5; // 定义数组大小*

*int length = 0; // 定义长度变量*

*int input;*

*char c[50];*

*int\* data = (int\*)malloc(n \* sizeof(int)); // 分配数据数组内存*

*int\* weight = (int\*)malloc(n \* sizeof(int)); // 分配频数数组内存*

*int\* metaData = (int\*)malloc(20 \* sizeof(int)); // 分配元数据数组内存*

*HuffmanTree HT;*

*InitHuffmanTree(HT, n); // 初始化哈夫曼树*

*// 输出信源数据*

*printf("信源数据如下：\n");*

*for (int i = 0; i < length; i++) {*

*printf("%d ", metaData[i]);*

*}*

*// 对数据进行冒泡排序并进行直方图统计*

*bubble\_sort(data, weight, n);*

*printf("\n\n一维数组直方图统计+冒泡排序:\n");*

*for (int i = 1; i <= n; i++) {*

*HT[i].data = data[i - 1];*

*HT[i].weight = weight[i - 1];*

*printf("信源符号为：%d, 权重为：%d\n", HT[i].data, HT[i].weight);*

*}*

*// 输出码书*

*printf("\n码书为：\n");*

*HuffmanCode HC;*

*CreatHuffmanTree(HT, n);*

*CreatHuffmanCode(HT, HC, n);*

*for (int i = 1; i <= n; i++)*

*printf("信源符号为：%d，码字为：%s\n", HT[i].data, HC[i]);*

*// 编码*

*printf("请输入需要编码的信源（范围0~4,以逗号分隔）,若不需要请输入-1，系统默认输出信源文件的编码及转换。\n");*

*gets(c);*

*char\* show = (char\*)malloc(200);*

*if (strcmp(c, "-1") != 0) {*

*length = 0;*

*char\* p = strtok(c, ",");*

*int num = 0;*

*if (p) {*

*if (atoi(p) < 0 || atoi(p) > 4) {*

*printf("输入信源范围有误，范围应在0~4\n");*

*return ERROR;*

*}*

*metaData[num] = atoi(p);*

*length++;*

*}*

*while (p = strtok(NULL, ",")) {*

*if (atoi(p) < 0 || atoi(p) > 4) {*

*printf("输入信源范围有误，范围应在0~4\n");*

*return ERROR;*

*}*

*metaData[++num] = atoi(p);*

*length++;*

*}*

*}*

*printf("\n码流为：\n");*

*showHuffmanTree(HT, HC, length, metaData, show);*

*// 解码*

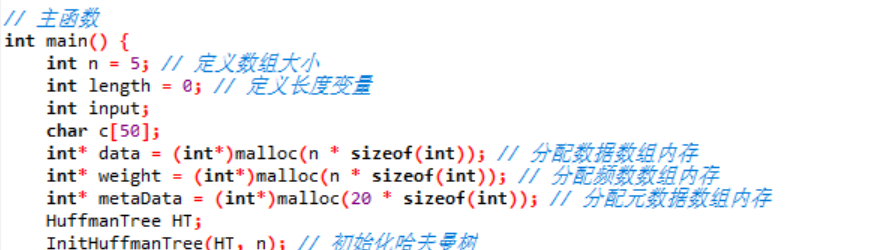
*printf("\n\n解码后：\n");*

*decodeHuffmanTree(HT, show);*

*return OK;*

*写出为解决所给问题需额外编写的测试程序或者对第四节算法程序所做的修改。*

**5.3初始参数设置**

****

**5.4实验结果**

****

**5.5实验分析**

霍夫曼编码是一种可变字长编码方法，基于信源的概率统计模型进行编码。其核心思想是，出现概率高的信源符号使用较短的编码，而出现概率低的信源符号则使用较长的编码。通过这种方式，霍夫曼编码能够降低编码后的平均码长，从而实现数据压缩的目的。在一维数字信号中，每个数字可以视为一个信源符号，其出现的概率可以根据信号的统计特性进行计算。通过构建霍夫曼树，并根据树的路径进行编码，可以得到每个数字的唯一编码表示。通过对比原始信号和压缩后的信号，可以分析霍夫曼编码的压缩效果。一方面，可以计算压缩率，即压缩后数据与原始数据的比值，以评估压缩效果的好坏。另一方面，可以观察压缩后数据的解码质量，即解码后数据与原始数据的差异程度，以验证霍夫曼编码的无损压缩特性。

实验优化：（1）改进频率统计方法：通过更精细的统计方法，如滑动窗口统计或分段统计，提高频率估计的准确性。

（2）优化霍夫曼树构建算法：采用更高效的算法构建霍夫曼树，如使用优先队列或堆数据结构来加速节点的选择和合并过程。