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： 江一诺

学 号： 2220220821

指导老师： 韩凤

实验日期： 2024年3月25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.

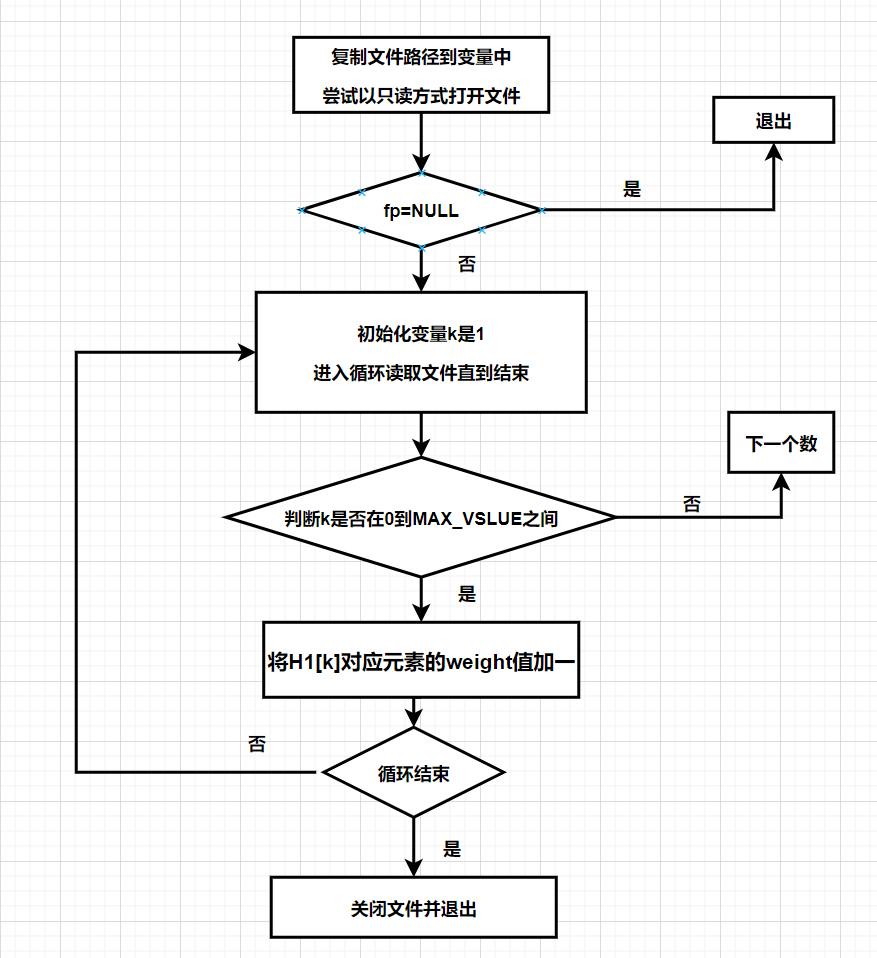
5. Copying work from online resources (Baidu,Chegg, google forums, etc.)

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

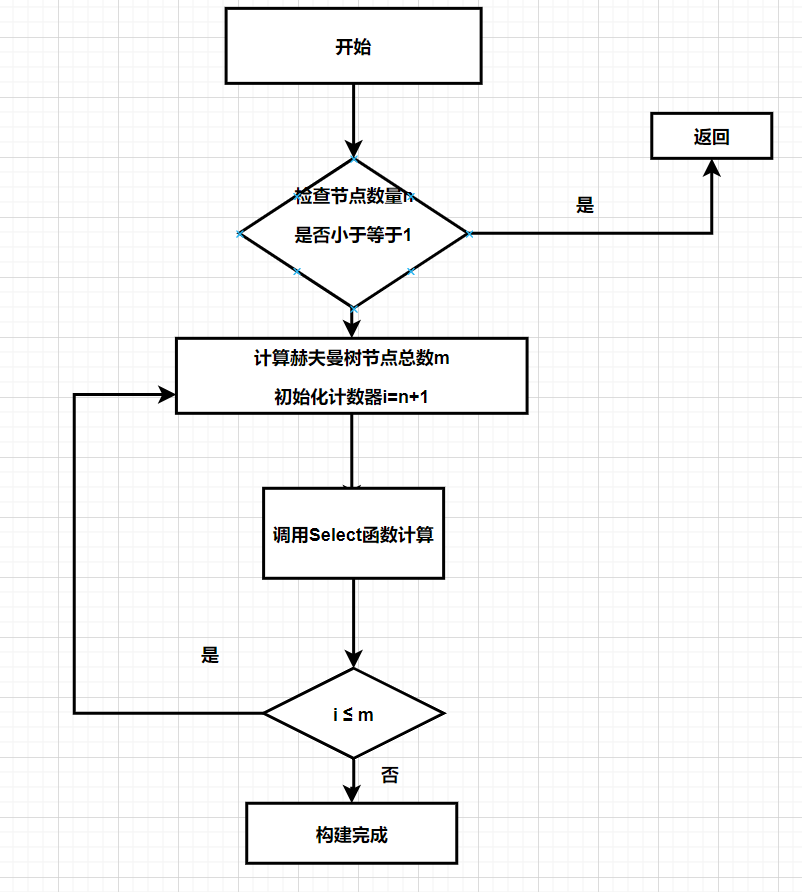
All submissions will be checked for similarity. This code will check each submission for similarity between other student submissions, past student submissions, the solution manual, and online resources and postings. If your submission is flagged for a high level of similarity, it will be turned in for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

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

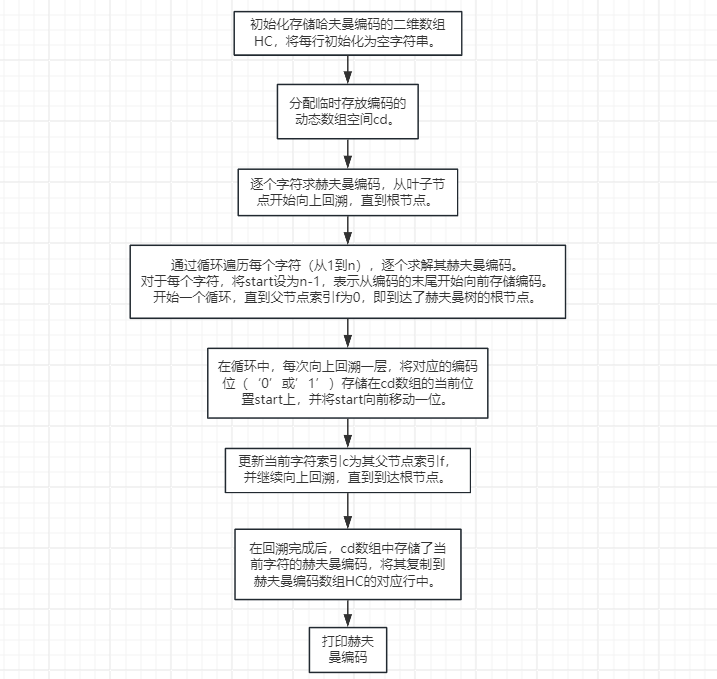
1. **实验目的**
2. 深入理解掌握树的基本概念、二叉树的基本操作、赫夫曼树的含义及其应用。
3. 掌握二叉树（赫夫曼树）生成、遍历等算法的实现。
4. 掌握利用赫夫曼算法对信号进行编解码的实现方法。
5. **实验内容与要求**
6. 设计函数实现对一维信号进行直方图统计。
7. 设计函数实现赫夫曼树构建。
8. 设计函数实现赫夫曼码书构建。
9. 设计函数实现对一维信号的赫夫曼编码与解码。
10. **算法描述与流程**
11. 设计函数实现对一维信号进行直方图

****

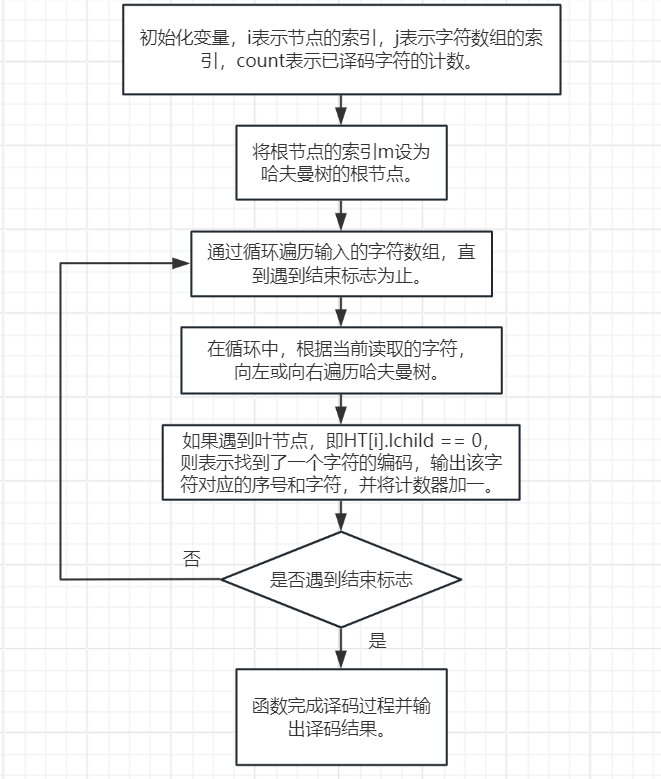
1. 设计函数实现赫夫曼码书的构建

****

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**

*统计直方图*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

*//ASCII码256个，所以长度设置为256*

#define MAX\_WORD\_LENGTH 256

#define NUM\_CHARS 256

typedef struct {

    char data;

    int weight;

    int parent;

    int lch;

    int rch;

} Histogram;

*哈夫曼树*

#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

#include <string.h>

#define MAX\_NODES 100 *// 树的最大节点数*

#define MAX\_LEAFS 50  *// 最大叶子节点数*

typedef struct {

    char data;

    int weight;

    int parent;

    int lchild;

    int rchild;

} HuffmanNode;

typedef char\*\* HuffmanCode;

*// 初始化哈夫曼树节点*

void initHuffmanTree(HuffmanNode\* nodes, int leafCount) {

    int totalNodes = 2 \* leafCount - 1;

    for (int i = leafCount; i < totalNodes; ++i) {

        nodes[i].parent = -1;

        nodes[i].lchild = -1;

        nodes[i].rchild = -1;

    }

}

*// 选择两个权重最小的节点*

void select(HuffmanNode\* nodes, int count, int\* s1, int\* s2) {

    int min1 = INT\_MAX, min2 = INT\_MAX;

    \*s1 = \*s2 = -1;

    for (int i = 0; i < count; ++i) {

*// 选择未连接的节点*

        if (nodes[i].parent == -1) {

*//选权重最小的两个节点*

            if (nodes[i].weight < min1) {

*//更新最小值*

                min2 = min1;

*//更新最小值节点指针*

                \*s2 = \*s1;

                min1 = nodes[i].weight;

                \*s1 = i;

            } else if (nodes[i].weight < min2) {

                min2 = nodes[i].weight;

                \*s2 = i;

            }

        }

    }

}

*// 创建哈夫曼树*

void createHuffmanTree(HuffmanNode\* nodes, int leafCount) {

    initHuffmanTree(nodes, leafCount);

    int totalNodes = 2 \* leafCount - 1;

*//遍历每个叶子节点，连接到两个权重最小的节点*

    for (int i = leafCount; i < totalNodes; ++i) {

        int s1, s2;

        select(nodes, i, &s1, &s2);

        nodes[s1].parent = i;

        nodes[s2].parent = i;

        nodes[i].lchild = s1;

        nodes[i].rchild = s2;

        nodes[i].weight = nodes[s1].weight + nodes[s2].weight;

    }

}

*// 哈夫曼编码*

void createHuffmanCode(HuffmanNode\* nodes, HuffmanCode\* hc, int leafCount) {

    \*hc = (HuffmanCode)malloc((leafCount + 1) \* sizeof(char\*));

*// 编码结束符*

    char\* cd = (char\*)malloc(leafCount \* sizeof(char));

    cd[leafCount - 1] = '\0';

*//遍历每个叶子节点，生成编码*

    for (int i = 0; i < leafCount; ++i) {

        int start = leafCount - 1;

        int c = i;

        int f = nodes[i].parent;

*// 回溯到根节点*

*//只要不是-1就有父节点，就一直回溯，直到根节点*

        while (f != -1) {

            start--;

*// 左子节点0，右子节点1*

            if (nodes[f].lchild == c) {

                cd[start] = '0';

            } else {

                cd[start] = '1';

            }

*//向上更新节点*

            c = f;

            f = nodes[f].parent;

        }

        (\*hc)[i] = (char\*)malloc(leafCount - start);

        strcpy((\*hc)[i], &cd[start]);

    }

    free(cd);  *// 释放临时空间*

}

*// 哈夫曼解码：将编码的字符串解码为原始信号*

void decodeSignal(HuffmanNode\* nodes, int \*signal, char\* encodedStr, int leafCount) {

*//根节点索引=(叶子数-1)\*2*

    int root = 2 \* leafCount - 2;

    int current = root;

    int length = 0;

    signal[length] = '\0';

    while (\*encodedStr) {

*// 左子节点0，右子节点1*

        if (\*encodedStr == '0') {

            current = nodes[current].lchild;

        } else {

            current = nodes[current].rchild;

        }

*// 到达叶子末端，重置当前索引到根节点*

        if (nodes[current].lchild == -1 && nodes[current].rchild == -1) {

            signal[length++] = nodes[current].data;

            current = root;

        }

*// 移动到下一个字符*

        encodedStr++;

    }

    signal[length] = '\0';

}

*// 打印哈夫曼树节点信息*

void printNodes(HuffmanNode\* nodes, int count) {

    for (int i = 0; i < count; ++i) {

        printf("Node %d: [data='%c', weight=%d, parent=%d, lchild=%d, rchild=%d]\n",

               i, nodes[i].data, nodes[i].weight, nodes[i].parent, nodes[i].lchild, nodes[i].rchild);

    }

}

*// 打印哈夫曼编码*

void printHuffmanCodes(HuffmanCode hc, HuffmanNode\* nodes, int leafCount) {

    printf("哈夫曼编码：\n");

    for (int i = 0; i < leafCount; ++i) {

        printf("Data '%c': Code = %s\n", nodes[i].data, hc[i]);

    }

}

1. **实验结果与分析**

**5.1测试用例**

**5.2测试程序**

*统计直方图*

int main() {

    char srcFile[MAX\_WORD\_LENGTH];

    strcpy(srcFile, "data.txt");

    FILE\* fp = fopen(srcFile, "r");

    if (fp == NULL) {

        printf("打开失败\n");

        exit(1);

    }

    Histogram HT[NUM\_CHARS] = {0}; *// 初始化结构体数组*

    for (int i = 0; i < NUM\_CHARS; i++) {

        HT[i].data = (char)i;

        HT[i].weight = 0;

        HT[i].parent = 0;

        HT[i].lch = 0;

        HT[i].rch = 0;

    }

    int ch;

    while ((ch = fgetc(fp)) != EOF) {

        HT[ch].weight++; *// 增加字符的出现次数*

    }

    fclose(fp);

*// 打印输出结果*

    for (int i = 0; i < NUM\_CHARS; i++) {

        if (HT[i].weight > 0) {

            printf("字符: %c 出现的频率: %d\n", HT[i].data, HT[i].weight);

        }

    }

    return 0;

}

*哈夫曼树*

int main() {

    HuffmanNode nodes[MAX\_NODES];

    int leafCount = 0;

*// 打开文件并读取数据*

    FILE \*file = fopen("huffmantree.txt", "r");

    if (file == NULL) {

        perror("Error opening file");

        return EXIT\_FAILURE;

    }

*// 读取文件中的字符和对应频率*

    while (fscanf(file, " %c %d", &nodes[leafCount].data, &nodes[leafCount].weight) == 2) {

        nodes[leafCount].parent = -1;

        nodes[leafCount].lchild = -1;

        nodes[leafCount].rchild = -1;

        leafCount++;

    }

    fclose(file);

*// 确保叶子节点数量没有超过定义的上限*

    if (leafCount > MAX\_LEAFS) {

        fprintf(stderr, "叶子节点数量超出最大限制。\n");

        return EXIT\_FAILURE;

    }

*// 创建哈夫曼树*

    createHuffmanTree(nodes, leafCount);

*// 打印节点信息*

    printf("哈夫曼树节点信息:\n");

    printNodes(nodes, 2 \* leafCount - 1);

*// 生成和打印哈夫曼编码*

    HuffmanCode hc;

    createHuffmanCode(nodes, &hc, leafCount);

    printHuffmanCodes(hc, nodes, leafCount);

    int decodedSignal[MAX\_LEAFS];

    for(int i = 0; i < leafCount; i++) {

    decodeSignal(nodes, decodedSignal, hc[i], leafCount);

    printf("%s 解码后: %c\n", hc[i],decodedSignal[0]);

    }

*// 释放哈夫曼编码存储空间*

    for (int i = 0; i < leafCount; ++i) {

        free(hc[i]);

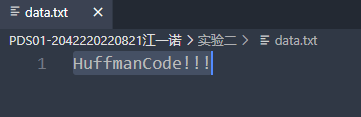
    }

    free(hc);

    return 0;

}

**5.3初始参数设置**





**5.4实验结果**

**5.5实验分析**

1. **算法选择**

* 数组计数法适合已知且有限的字符集，如ASCII
* 哈希表适合未知或大型字符集
* 当前算法适合这个实验的需求

1. **数据结构设计问题**

* 原程序的Histogram结构体包含未使用的字段(parent、lch、rch)，这表明可能是为了后续构建霍夫曼树预留的
* 改进版使用了更简洁的CharCount结构体，只保留必要字段

1. **用户体验问题**

* 原程序使用硬编码文件名，缺乏灵活性
* 原程序对特殊字符的显示不够友好
* 改进版增加了命令行参数支持和更友好的输出格式

1. **实验中遇到的问题**

* 中文或其他非ASCII字符可能无法正确处理
* 不可打印字符在原程序中显示可能会混乱
* 数据文件不存在时的错误处理不够友好

1. **算法局限性**

* 只能处理ASCII字符集
* 不能处理大文件（如果文件太大，可能需要使用分块读取策略）
* 不能处理二进制文件中的所有可能值