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工程专业**

**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.)

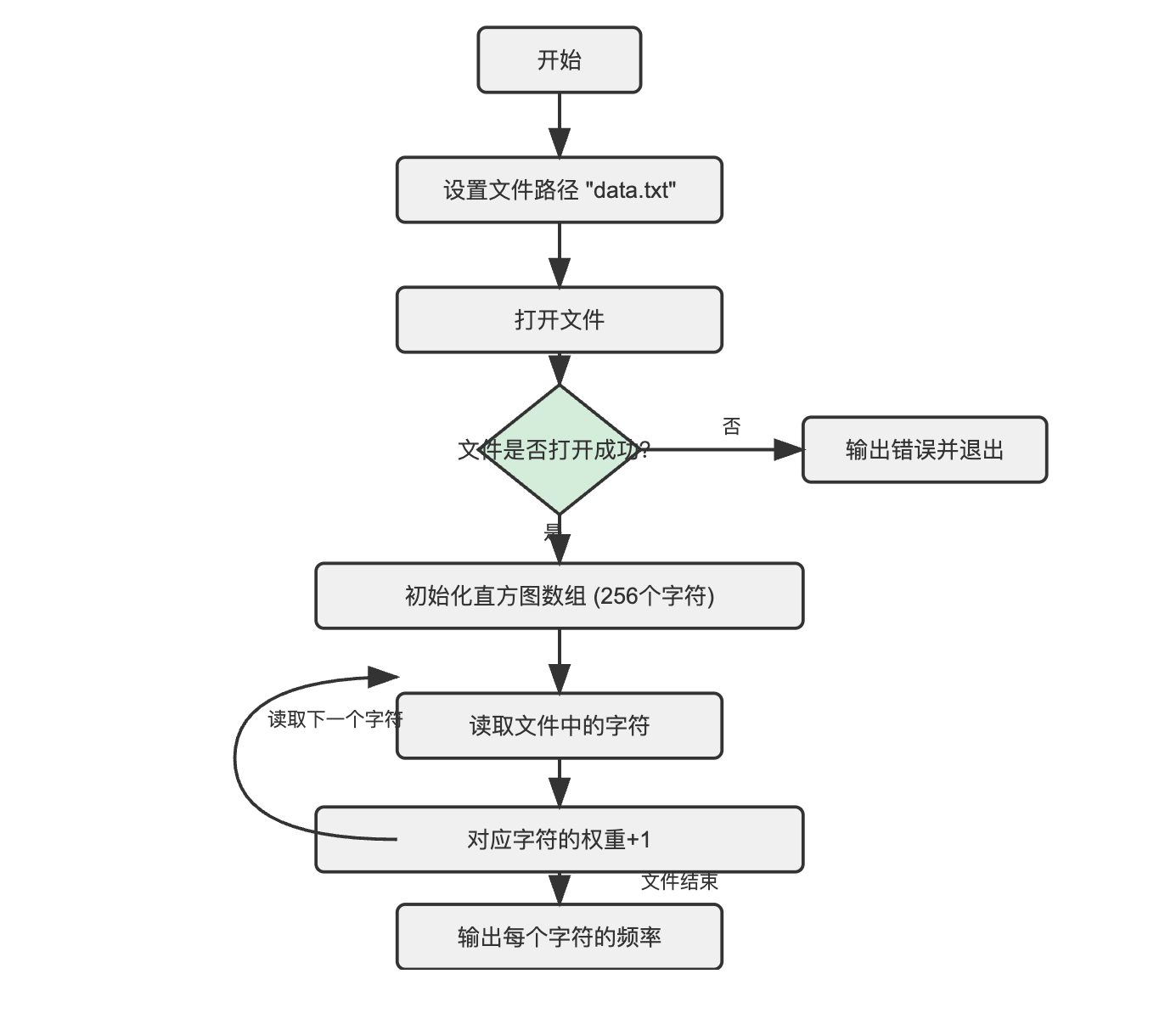
6. Posting work to online resources where other students can view your work.

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.

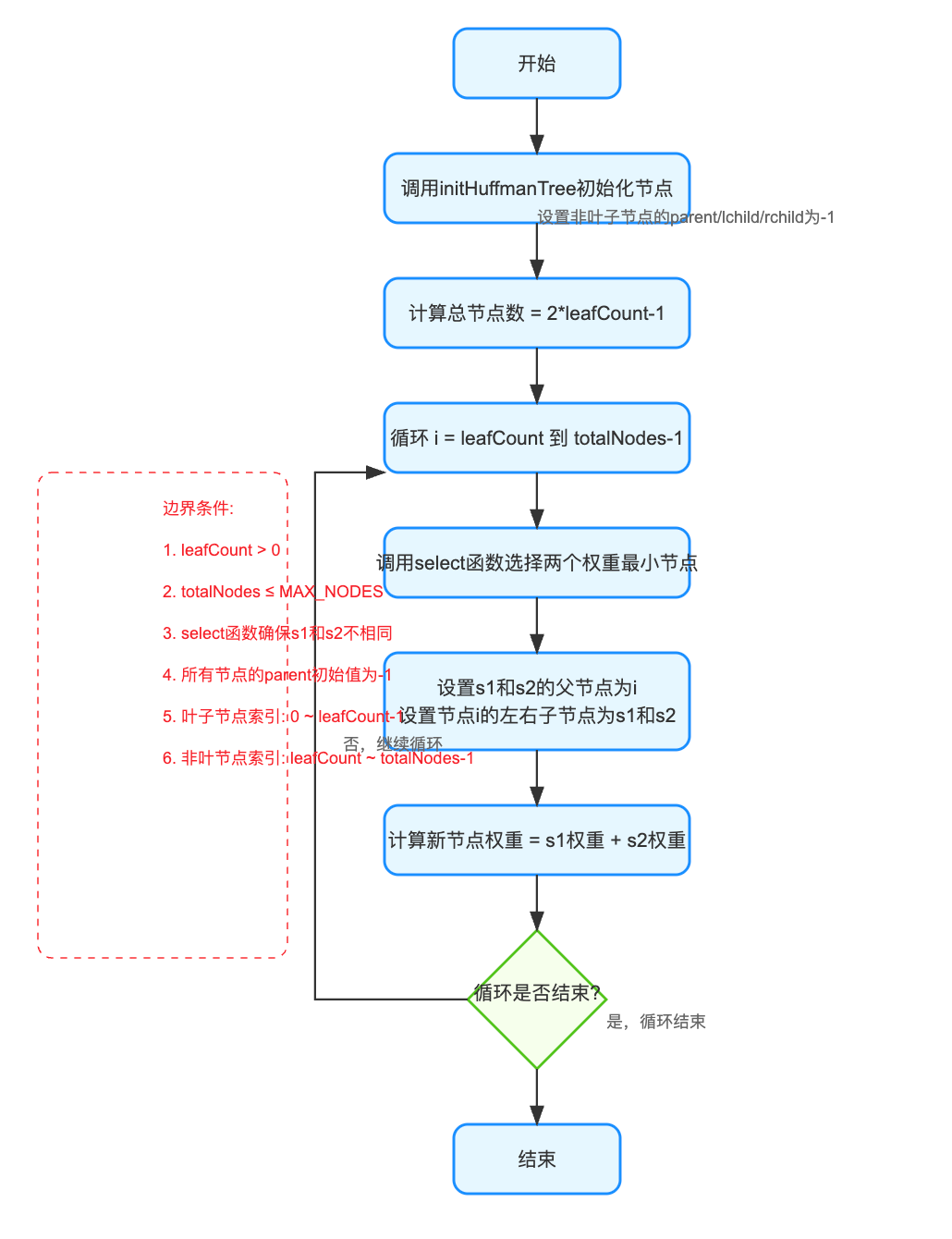
NOTE: Changing variable names, adding comments, or spacing will l result in a violation.

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**

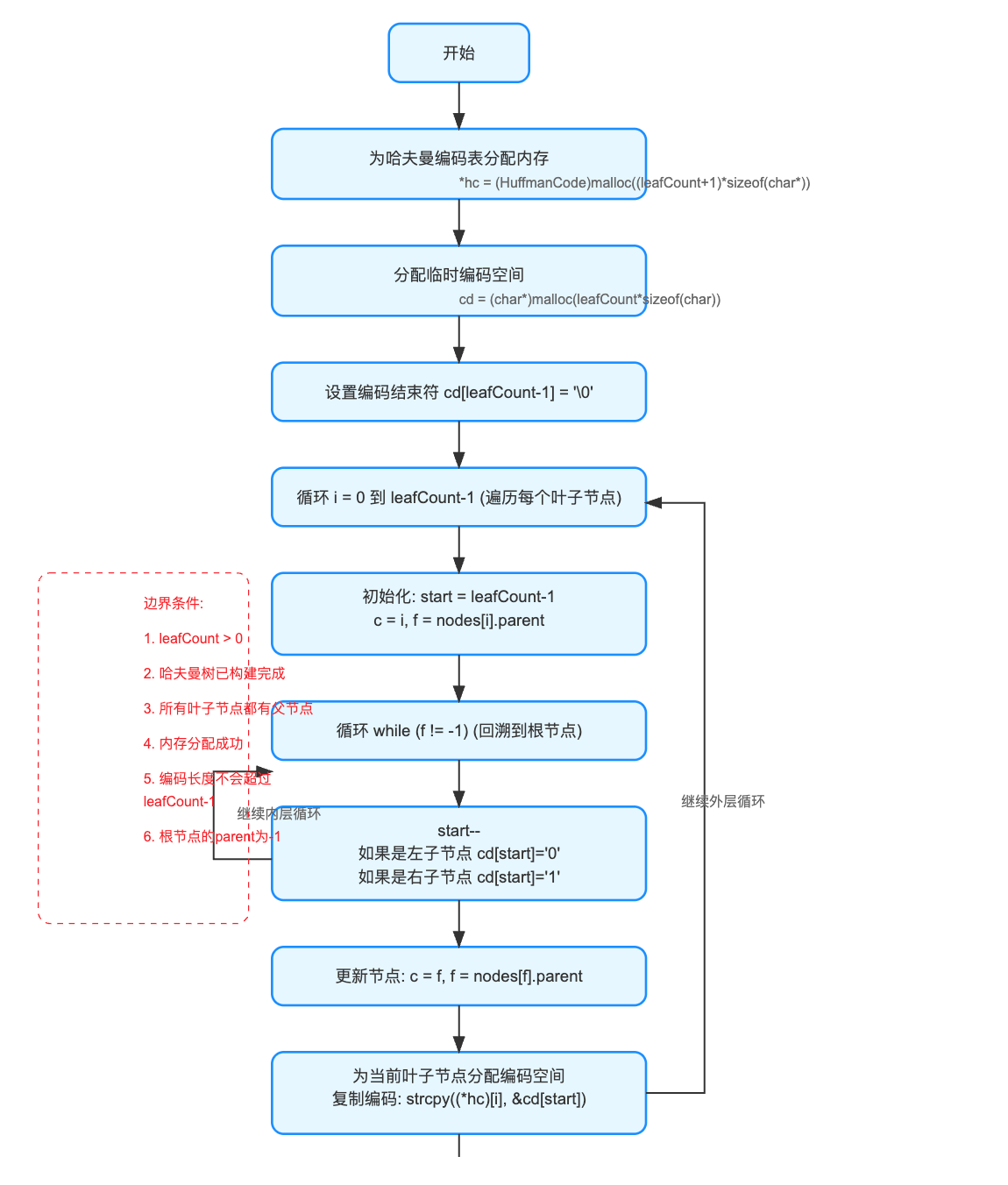
设计函数实现对一维信号进行直方图统计



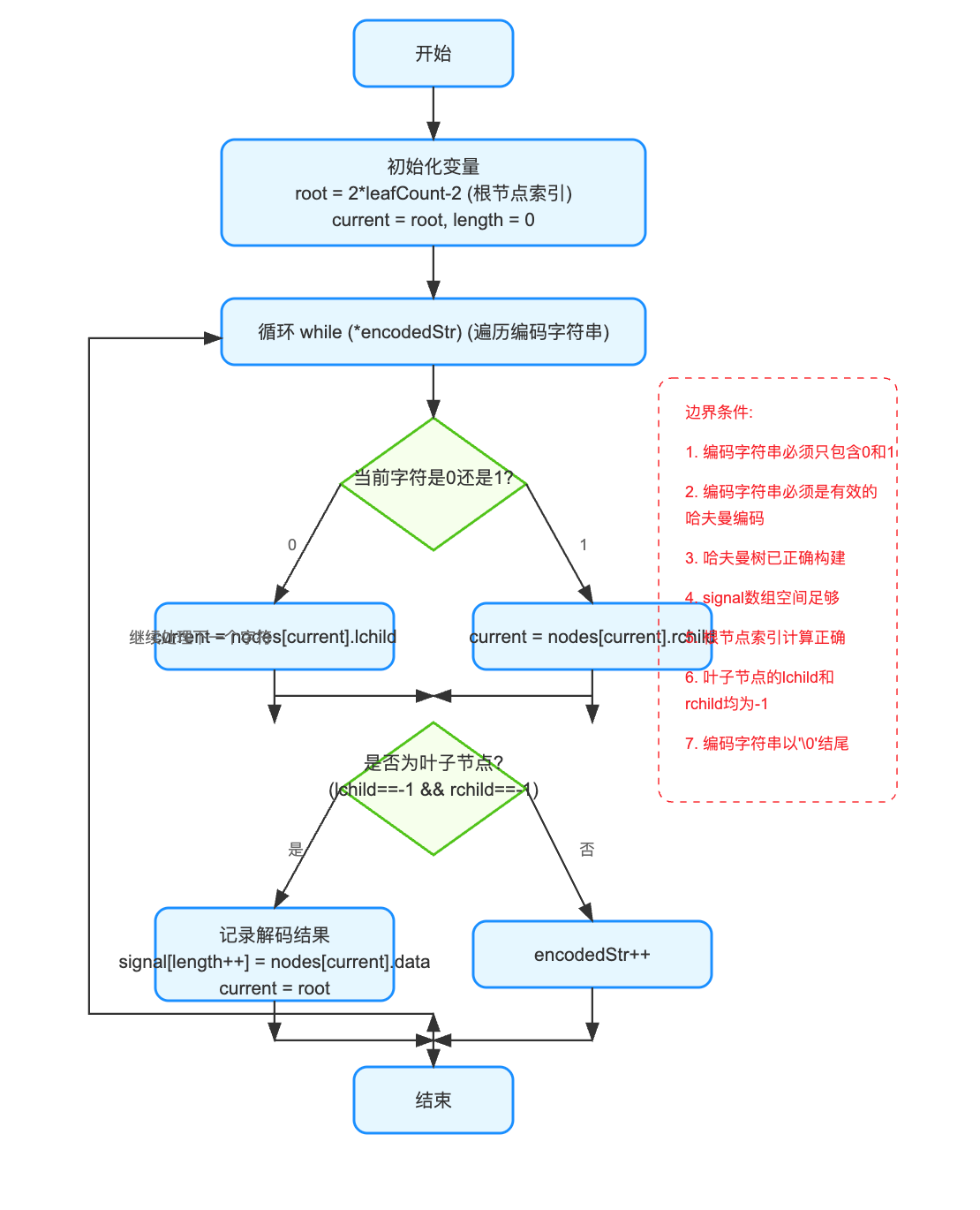
1. 设计函数实现赫夫曼码书的构建



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]);

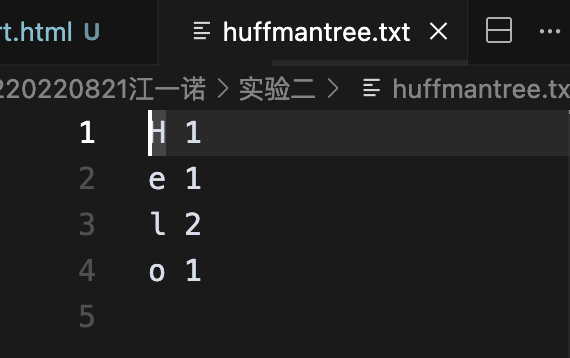
    }

}

1. **实验结果与分析**

### 5.1测试用例（初始参数设置）





### 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);

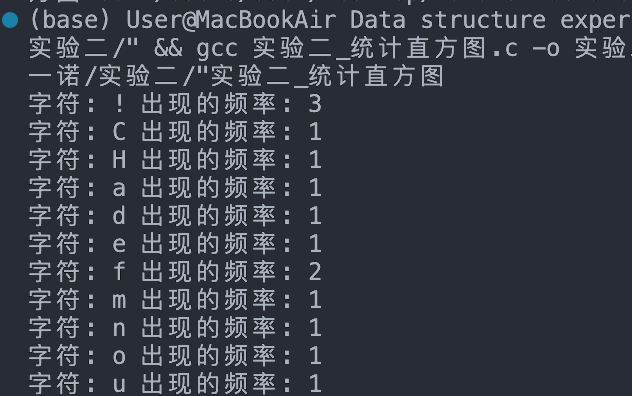
    return 0;

}

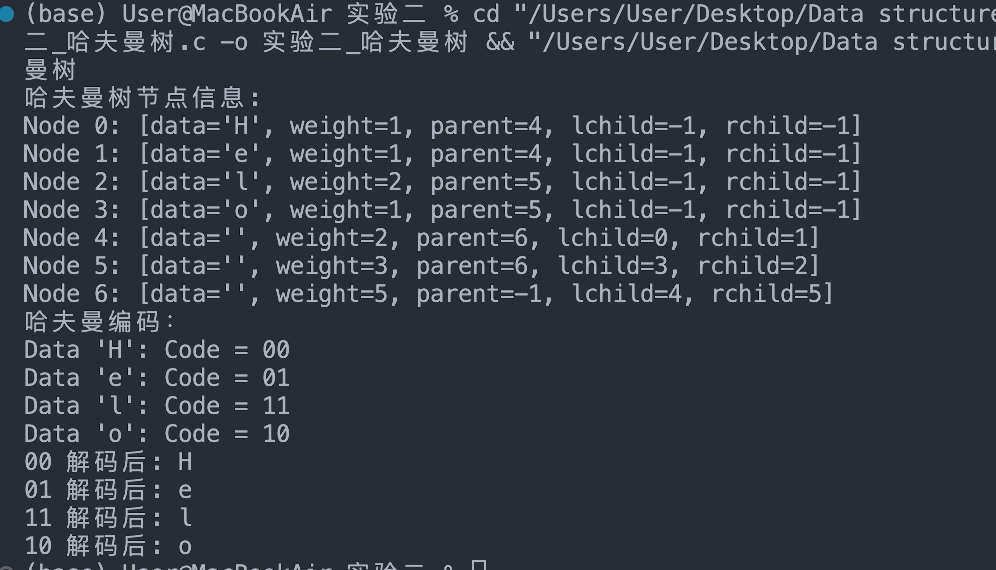
### 5.3实验结果

终端运行结果：

统计直方图



哈夫曼树



### 5.4实验分析

1. 时间和空间复杂度分析

时间复杂度

初始化哈夫曼树节点 (initHuffmanTree): O(n)，需要遍历所有节点初始化父子关系。

选择两个权重最小的节点 (select): O(n^2)，每次选择需要遍历所有未连接节点。

创建哈夫曼树 (createHuffmanTree): O(n^2)，需要多次调用选择函数。

生成哈夫曼编码 (createHuffmanCode): O(n^2)，每个叶子节点回溯到根节点。

解码信号 (decodeSignal): O(m)，m为编码字符串长度。

空间复杂度

整体数据结构: O(n)，n为节点数量。

各操作额外空间: O(n)，如生成编码时的临时存储。

(2) 边界条件分析

文件读取:

确保文件成功打开，读取字符和频率。

叶子节点数量不超过最大限制。

编码和解码:

确保编码字符串正确解码为原始信号。

处理空编码字符串的情况。

(3) 实验中出现的问题与改进方案

内存管理:

释放哈夫曼编码存储空间，但未处理异常退出时的内存泄漏。

应添加异常处理机制，确保内存安全释放。

错误处理不足:

文件操作错误处理简单，需增加详细错误信息。

选择和创建树的过程中缺乏错误返回机制。

(4) 实验结论

数据结构选择:

哈夫曼树适合处理字符频率编码问题。

通过节点的父子关系简化了编码和解码过程。

算法局限性:

选择节点的过程效率较低，适合小规模数据。

编码生成过程对节点数量敏感，可能影响性能。

优化方向:

使用更高效的数据结构（如优先队列）优化节点选择。

增加对不同编码方案的支持，提高灵活性。