计算机与信息工程学院实验报告

••••••••••••••••••••••••••••••••• 密 ••••••••••••••••••••••••••••••••• 封 ••••••••••••••••••••••••••••••••• 线 •••••••••••••••••••••••••••••••••

姓名：\_杨佳森\_学号：2112080106 专业：\_数据科学与大数据技术年级：\_2021\_\_

课程： 数据结构 主讲教师：\_\_刘成\_ 辅导教师：\_\_\_\_无\_\_\_

实验时间：\_2023\_\_年 5月 2\_日 下\_午17\_\_时至\_19\_时，实验地点\_计算机大楼213\_

实验题目： 赫夫曼树的综合应用——发报机模拟器

实验目的： 通过该实验，使学生理解赫夫曼树的概念，掌握赫夫曼树及赫夫曼编码的构造过程，体会网络发送端和接收端编码和译码过程及其工作原理。

实验环境（硬件和软件） PC和vs code

实验内容：

1) 输入HuffmanTree的参考标准底稿：输入一段字符串，作为发送方和接收方进行编码的统一标准，统计出现的字符种类数和出现的频度。

2）初始化HuffmanTree参数：给定报文中26个字母a-z及空格的出现频率{64, 13, 22, 32, 103, 21, 15, 47, 57, 1, 5, 32, 20, 57, 63, 15, 1, 48, 51, 80, 23, 8, 18, 1, 16, 1, 168}。

3）创建HuffmanTree和编码表：根据1）或2）的输入，构建赫夫曼树并为这些字符编制赫夫曼编码，并输出。

5）模拟发送端，从键盘输入字符串，在屏幕上输出输入串的编码

6）模拟接收端，从键盘上输入0-1赫夫曼编码串，翻译出对应的原文。

实验步骤：

1、输入哈夫曼树的参考标准底稿

    // 输入哈夫曼树的参考标准底稿

    int freq[] = { 64, 13, 22, 32, 103, 21, 15, 47, 57, 1, 5, 32, 20, 57, 63, 15, 1, 48, 51, 80, 23, 8, 18, 1, 16, 1, 168 };

    unordered\_map<char, int> freqMap;

    for (int i = 0; i < 26; i++) {

        freqMap[char('a' + i)] = freq[i];

    }

    freqMap[' '] = freq[26];

2、初始化哈夫曼树的参数

    // 初始化哈夫曼树的参数

    priority\_queue<Node> pq;

    for (auto p : freqMap) {

        pq.push(Node(p.first, p.second));

    }

    while (pq.size() > 1) {

        Node\* left = new Node(pq.top());

        pq.pop();

        Node\* right = new Node(pq.top());

        pq.pop();

        pq.push(Node(left, right));

    }

    Node\* root = new Node(pq.top());

    pq.pop();

3、输出哈夫曼编码表

    for (char c = 'a'; c <= 'z'; c++) {

        if (codeTable.find(c) != codeTable.end()) {

            cout << c << ": " << codeTable[c] << endl;

        }

    }

    if (codeTable.find(' ') != codeTable.end()) {

        cout << " : " << codeTable[' '] << endl;

    }

4、模拟发送端

            // 模拟发送端

            string message;

            cin.ignore();

            getline(cin, message);

            string encodedMessage;

            for (char c : message) {

                encodedMessage += codeTable[c];

            }

            cout << "编码后的报文: " << encodedMessage << endl;

5、模拟接收端

    // 模拟接收端

    Node\* current = root;

    string encoded, decodedMessage;

    cin >> encoded;

    for (char bit : encoded) {

        current = bit == '0' ? current->left : current->right;

        if (current->left == nullptr && current->right == nullptr) {

            decodedMessage += current->data;

            current = root;

        }

    }

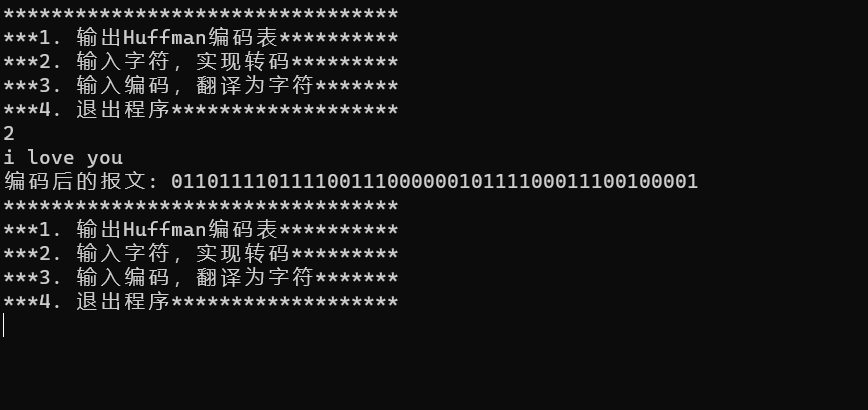
    cout << "解码后的报文: " << decodedMessage << endl;

实验数据记录：

1、模拟发送端

输入：i love you

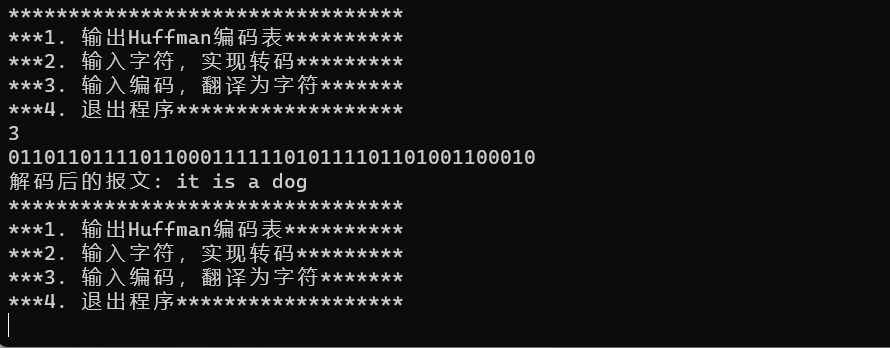
输出：01101111011110011100000010111100011100100001



2、模拟接收端

输入：01101101111011000111111010111101101001100010

输出：it is a dog



问题讨论：

1.权值越大的叶子节点越靠近根节点，权值越小的叶子节点越远离根节点。

2.只有度为0（叶子节点）和度为2（分支节点）的节点，没有度为1的节点。

**源代码：**

#include <iostream>

#include <unordered\_map>

#include <queue>

#include<string>

using namespace std;

struct Node {

    char data;

    int freq;

    Node\* left = nullptr;

    Node\* right = nullptr;

    Node(char d, int f) : data(d), freq(f) {}

    Node(Node\* l, Node\* r) : freq(l->freq + r->freq), left(l), right(r) {}

    bool operator<(const Node& other) const {

        return freq > other.freq;

    }

};

void generateCode(Node\* root, string code, unordered\_map<char, string>& table) {

    if (root == nullptr) {

        return;

    }

    if (root->left == nullptr && root->right == nullptr) {

        table[root->data] = code;

        return;

    }

    generateCode(root->left, code + "0", table);

    generateCode(root->right, code + "1", table);

}

//销毁哈夫曼树

void deleteTree(Node\* root) {

    if (root == nullptr) {

        return;

    }

    deleteTree(root->left);

    deleteTree(root->right);

    delete root;

}

void show() {

    cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*1. 输出Huffman编码表\*\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*2. 输入字符，实现转码\*\*\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*3. 输入编码，翻译为字符\*\*\*\*\*\*\*" << endl;

    cout << "\*\*\*4. 退出程序\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

}

int main() {

    // 输入哈夫曼树的参考标准底稿

    int freq[] = { 64, 13, 22, 32, 103, 21, 15, 47, 57, 1, 5, 32, 20, 57, 63, 15, 1, 48, 51, 80, 23, 8, 18, 1, 16, 1, 168 };

    unordered\_map<char, int> freqMap;

    for (int i = 0; i < 26; i++) {

        freqMap[char('a' + i)] = freq[i];

    }

    freqMap[' '] = freq[26];

    // 初始化哈夫曼树的参数

    priority\_queue<Node> pq;

    for (auto p : freqMap) {

        pq.push(Node(p.first, p.second));

    }

    while (pq.size() > 1) {

        Node\* left = new Node(pq.top());

        pq.pop();

        Node\* right = new Node(pq.top());

        pq.pop();

        pq.push(Node(left, right));

    }

    Node\* root = new Node(pq.top());

    pq.pop();

    // 生成Huffman编码表

    unordered\_map<char, string> codeTable;

    generateCode(root, "", codeTable);

    int choice;

    while (true) {

        show();

        cin >> choice;

        switch (choice) {

        case 1:

        {

            // 输出Huffman编码表

            for (char c = 'a'; c <= 'z'; c++) {

                if (codeTable.find(c) != codeTable.end()) {

                    cout << c << ": " << codeTable[c] << endl;

                }

            }

            if (codeTable.find(' ') != codeTable.end()) {

                cout << " : " << codeTable[' '] << endl;

            }

            break;

        }

        case 2:

        {

            // 模拟发送端

            string message;

            cin.ignore();

            getline(cin, message);

            string encodedMessage;

            for (char c : message) {

                encodedMessage += codeTable[c];

            }

            cout << "编码后的报文: " << encodedMessage << endl;

            break;

        }

        case 3:

        {

            // 模拟接收端

            Node\* current = root;

            string encoded, decodedMessage;

            cin >> encoded;

            for (char bit : encoded) {

                current = bit == '0' ? current->left : current->right;

                if (current->left == nullptr && current->right == nullptr) {

                    decodedMessage += current->data;

                    current = root;

                }

            }

            cout << "解码后的报文: " << decodedMessage << endl;

            break;

        }

        case 4:

        {

            cout << "程序已退出" << endl;

            deleteTree(root);

            exit(0);

            break;

        }

        default:

        {

            cout << "无效的选择，请重新输入" << endl;

            break;

        }

        }

    }

    return 0;

}