同济大学
TONGJI UNIVERSITY

《离散数学》课程实验报告

题目 最优二元树

姓　名 林继申

学　号 2250758

学　院 软件学院

专　业 软件工程

教　师 李　冰

二〇二三 年 十二 月 二十二 日

目录

1 实验目的.....1
2 实验内容.....1
3 实验环境.....1
4 实验原理.....1
4.1 最优二元树（哈夫曼树）.....1
4.2 最优二元树编码（哈夫曼编码）.....2
5 实验过程.....2
5.1 实验思路.....2
5.2 实验设计.....3
5.2.1 数据结构设计.....3
5.2.2 TreeNode 结构体的设计.....3
5.2.2.1 概述.....3
5.2.2.2 结构体定义.....3
5.2.2.3 数据成员.....3
5.2.2.4 构造函数.....4
5.2.3 程序主体架构设计.....4
5.3 程序功能实现.....5
5.3.1 输入最优二元树节点个数与节点值功能的实现.....5
5.3.2 退出程序功能的实现.....6
5.4 核心算法实现.....6
5.4.1 构建最优二元树算法的实现.....6
5.4.2 递归遍历最优二元树算法的实现.....7
6 实验数据分析.....7
7 实验心得.....10
8 程序源文件.....10

1 实验目的

本实验旨在通过实现一个最优二元树（Optimal Binary Tree）来深入理解和应用二叉树的数据结构。实验重点包括：

(1)理解和应用二叉树在存储和处理数据方面的优势；

(2)掌握构建最优二元树的算法和步骤；

(3)学习如何使用树的遍历方法来提取和打印前缀码；

(4)加强对 C++语言和面向对象编程的理解。

2 实验内容

本实验的主要任务是输入一组通信符号的使用频率，并求出各通信符号对应的前缀码。这个过程包括以下几个关键步骤：

(1)数据输入：用户输入通信符号的使用频率；

(2)构建最优二元树：根据输入的频率数据，构建一个最优二元树；

(3)前缀码提取：遍历构建好的二元树，提取出每个通信符号对应的前缀码。

3 实验环境

程序开发语言：C++

集成开发环境：Microsoft Visual Studio 2022 (Release 模式)

编译运行环境：本项目适用于 x86 架构和 x64 架构

4 实验原理

4.1 最优二元树（哈夫曼树）

最优二元树，通常被称为哈夫曼树（Huffman Tree），是一种特殊的二叉树，用于有效编码。在哈夫曼树中，每个叶子节点代表一个字符或通信符号，而每个节点的权值则代表该字符的使用频率或概率。哈夫曼树的主要特点是提供一种高效的编码方法，其中最常用的字符有最短的编码，而不常用的字符则有较长的编码。

哈夫曼树的构建基于这样一个原则：给定一组权值（例如字符频率），构建一棵具有最小带权路径长度的二叉树。带权路径长度指的是树中所有叶子节点的权值与其到根节点路径长度乘积的总和。

最优二元树（哈夫曼树）构建过程：

(1)初始化：根据输入的每个符号及其频率，创建一个森林（每个符号为一个节点，构成一个单节点树）；

(2)构建树：在森林中选取两个权值最小的树合并，作为一棵新树的左右子树，新树的根节点权值为这两棵子树根节点权值之和；

(3)重复合并：重复上述过程，每次合并后更新森林，直到森林中只剩下一棵树为止。这棵树就是最终的哈夫曼树。

4.2 最优二元树编码（哈夫曼编码）

最优二元树编码（哈夫曼编码）是一种变长编码方式，用于无损数据压缩。在哈夫曼编码中，每个符号根据其出现频率被赋予一个唯一的二进制码。符号的编码长度与其频率成反比，即出现频率高的符号有更短的编码。

最优二元树编码（哈夫曼编码）编码原则：

(1)不等长编码：符号的编码长度不同，频率高的符号编码短，频率低的符号编码长；

(2)前缀无歧义：任何符号的编码都不是另一个符号编码的前缀，保证了编码的唯一解码性。

最优二元树编码（哈夫曼编码）编码过程：

(1)构建哈夫曼树：根据符号的使用频率构建哈夫曼树；

(2)生成编码：从根节点到每个叶子节点的路径确定了该节点符号的编码。路径中左分支表示“0”，右分支表示“1”。

哈夫曼编码广泛应用于数据压缩领域，如文件压缩、多媒体数据编码等。它通过有效减少常用符号的编码长度，达到压缩数据的目的。

5 实验过程

5.1 实验思路

本实验的核心思路是利用一维向量 `std::vector<int> frequencies` 存储通信符号的使用频率，并通过构建最优二元树（哈夫曼树）来为每个通信符号生成前缀码。在这个过程中，使用链表来保存这个最优二元树，并通过树的遍历

方法输出前缀码。

链表的概念主要体现在构建哈夫曼树的过程中。链表不是以传统的std::list形式出现，而是通过std::vector<std::shared_ptr<TreeNode>>来实现，其中 TreeNode 是自定义的结构，用于表示树中的每个节点。每个TreeNode 包含指向其左右子节点的指针（left 和 right），这些指针实际上形成了一个链表结构。

树的遍历在本程序中体现在 printCodes 函数中，该函数用于输出哈夫曼编码。遍历是通过递归调用实现的，分别对树的左子树和右子树进行遍历。

5.2 实验设计

5.2.1 数据结构设计

在本实验中，数据结构设计的核心是构建和管理哈夫曼树。关键的数据结构包括：

⑴频率数组：使用 std::vector<int>类型来存储通信符号的使用频率。这个一维向量是构建哈夫曼树的基础，它记录了每个符号的使用频率；

⑵节点向量：利用 std::vector<std::shared_ptr<TreeNode>>来管理哈夫曼树的节点。这种结构允许动态地添加和移除节点，同时通过智能指针自动管理内存，避免内存泄漏；

⑶哈夫曼树：通过连接 TreeNode 结构构成的哈夫曼树。每个节点包含一个数字（表示频率或权值），以及指向其左右子节点的指针。

5.2.2 TreeNode 结构体的设计

5.2.2.1 概述

TreeNode 结构体是实现最优二元树（哈夫曼树）的基础。它用于表示树中的每一个节点，包括节点存储的数据（如频率）和指向子节点的连接。

5.2.2.2 结构体定义

```
struct TreeNode {
    int num;
    std::shared_ptr<TreeNode> left, right;
    TreeNode(int n) : num(n), left(NULL), right(NULL) {}
};
```

5.2.2.3 数据成员

int num：表示节点存储的数据，如频率或权值

`std::shared_ptr<TreeNode> left, right`：智能指针，分别指向该节点的左子节点和右子节点

5.2.2.4 构造函数

```
TreeNode(int n) : num(n), left(NULL), right(NULL) {}
```

构造函数。初始化 `TreeNode` 实例，设置其 `num` 成员为传入的参数值 `n`，将左右子节点指针 `left` 和 `right` 初始化为 `NULL`，表示在创建节点时，它不具有任何子节点。

5.2.3 程序主体架构设计

本程序的主体架构可以分为三个层次：用户交互层、数据处理层和树操作层。每个层次承担不同的职责，共同实现最优二元树（哈夫曼树）的构建和编码。

⑴用户交互层：处理用户输入和输出显示。

①`main`：程序的入口点，管理用户交互流程；

②`inputInteger`：接收和验证用户输入的整数；

③`inputLogicalValue`：接收用户的逻辑输入，如程序退出决策。

⑵数据处理层：处理和准备用于构建树的数据。

①`initNodes`：根据用户输入的频率数据，初始化树节点；

②`insertSorted`：对树节点进行排序，以便于构建最优二元树。

⑶树操作层：执行最优二元树的构建和遍历。

①`constructTree`：构建最优二元树，使用节点排序和合并算法；

②`printCodes`：递归遍历树，打印每个符号的哈夫曼编码。

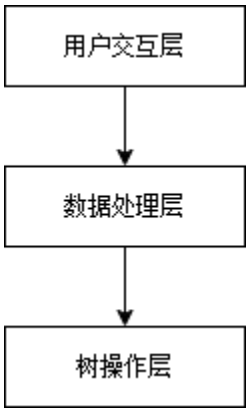

图 5.2.3.1 程序主体架构设计层次图

程序主体架构设计为：

⑴进入程序：用户通过 `main` 函数启动程序，进入用户交互层；

⑵输入数据：用户输入频率数据，通过 `inputInteger` 处理；

⑶初始化节点：使用 `initNodes` 基于输入数据创建树节点；

⑷构建最优二元树：调用 `constructTree` 在树操作层构建最优二元树；

(5)输出编码：使用 printCodes 遍历树并输出哈夫曼编码；

(6)退出程序：用户决定是否退出程序，通过 inputLogicalValue 处理。

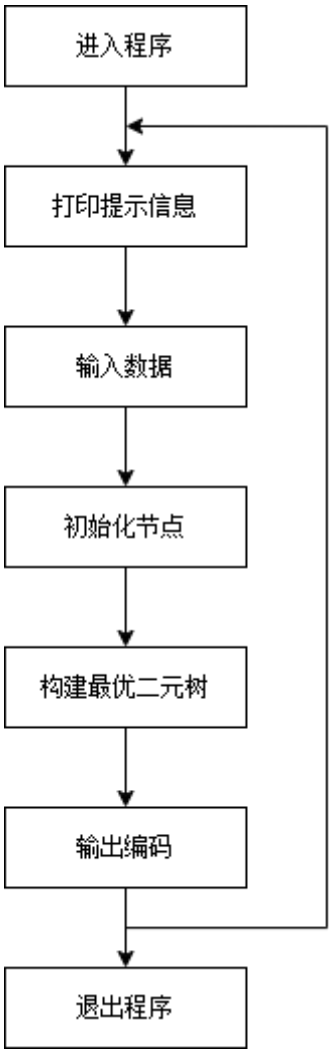

图 5.2.3.2 程序主体架构设计流程图

5.3 程序功能实现

5.3.1 输入最优二元树节点个数与节点值功能的实现

程序通过调用 inputInteger 函数输入最优二元树节点个数与节点值。inputInteger 函数用于获取用户输入的整数，同时限制输入必须在指定的范围内。inputInteger 函数对输入非法的情况进行了处理，代码具体执行逻辑如下：

(1)进入一个无限循环，它会一直运行直到用户提供有效的输入；

(2)用户的输入被读取到 tempInput 变量中，这里采用 double 类型来接收输入以便后续检查；

(3)进行输入验证：std::cin.good()检查输入流的状态是否正常，确保没有发生数据类型输入错误，tempInput==static_cast<int>(tempInput)检查

用户输入是否为整数，通过将其转换为整数再比较，tempInput>=lowerLimit 和 tempInput<=upperLimit 确保输入在指定的范围内；

(4)合法输入处理：如果用户提供了合法的输入，函数会清除输入流的错误状态，丢弃输入缓冲区中的任何剩余内容，然后返回转换后的整数值；

(5)非法输入处理：如果用户提供的输入不合法，函数会输出错误消息，清除输入流的错误状态，丢弃输入缓冲区中的内容，并继续循环以等待用户提供合法的输入。

5.3.2 退出程序功能的实现

退出程序的功能是通过在 main 函数内部的一个循环结构实现的。此功能允许用户在完成操作后选择是否退出程序。

程序功能实现思路：

(1)程序使用 do-while 循环来反复执行程序和询问用户是否退出程序。在每次循环的开始，使用 system("cls")清除屏幕，以提供清晰的界面；

(2)在完成一轮操作后，程序会输出询问用户是否退出程序的提示：“是否退出程序［y/n]：”。此时，程序再次调用 inputLogicalValue 函数，这次用于接收用户的退出决定。参数被设置为'n'（不退出）和'y'（退出）；

(3)如果用户输入'y'(对应真值),则 inputLogicalValue 函数返回 true，导致 do-while 循环结束，程序随之退出。如果用户输入'n'（对应假值），inputLogicalValue 函数返回 false，do-while 循环继续，用户可以进行新一轮操作。

5.4 核心算法实现

5.4.1 构建最优二元树算法的实现

最优二元树（哈夫曼树）是根据输入数据（符号频率）构建的特殊树结构，旨在实现高效编码。此算法通过组合频率最小的两个节点来逐步构建树。

constructTree 函数实现思路如下：

(1)循环直到节点向量只剩一个元素（树的根节点）；

(2)在每次循环中，首先调用 insertSorted 对节点进行排序；

(3)选取两个最小的节点作为左右子节点，创建一个新的父节点，其值为两子节点之和；

(4)将新创建的父节点添加回节点向量中。

算法特点：保证树的最终形态在所有可能的树结构中具有最小的加权路径长度，确保了编码的高效性。

5.4.2 递归遍历最优二元树算法的实现

递归遍历算法用于生成哈夫曼编码。它遍历树的每个节点，根据节点的位置确定编码。

`printCodes` 函数实现思路如下：

(1)从根节点开始，递归遍历每个节点；

(2)每向左遍历一步，编码添加`"0"`，每向右一步，编码添加`"1"`；

(3)当到达叶子节点时，打印节点的值（频率）和累计的前缀码；

(4)使用 `std::setw` 确保输出格式整齐。

算法特点：递归方法简化了遍历过程，使代码更加简洁易读。同时，确保了每个符号的唯一且高效的编码生成。

6 实验数据分析

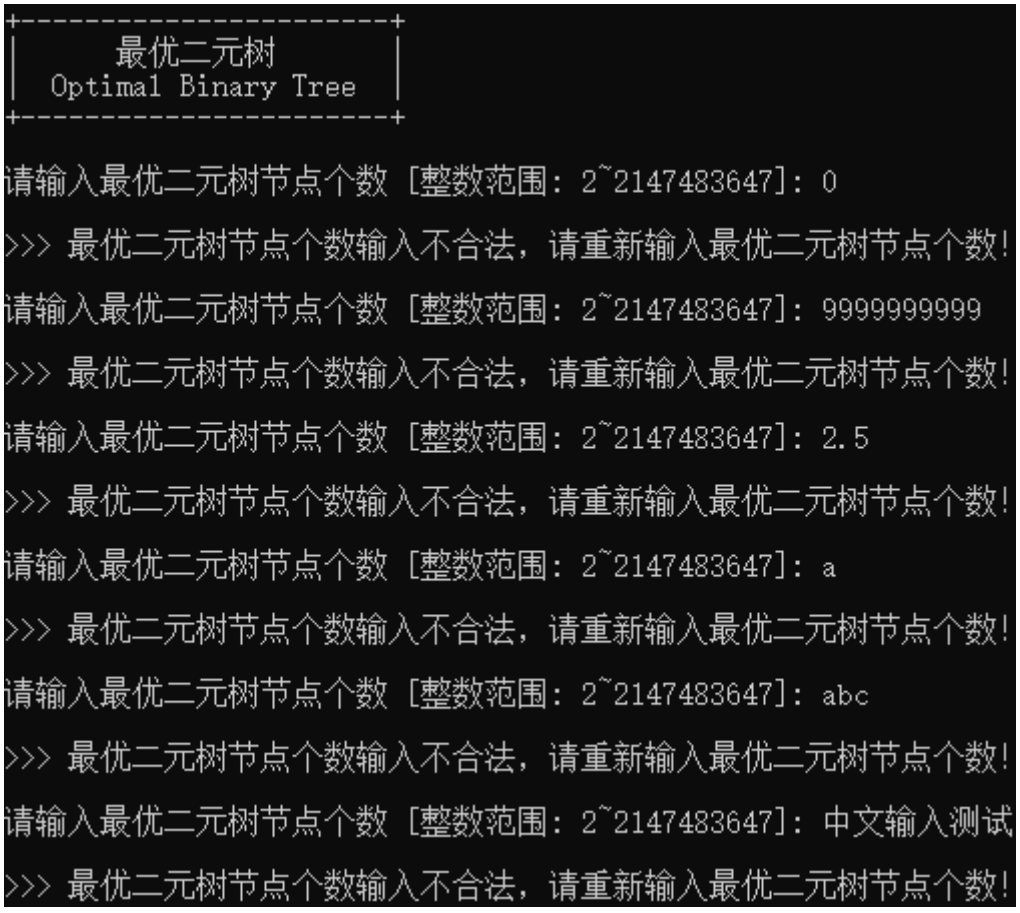

图 6.1 实验数据 1

实验数据分析 1：分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理，并要求用户重新输入最优二元树节点个数。

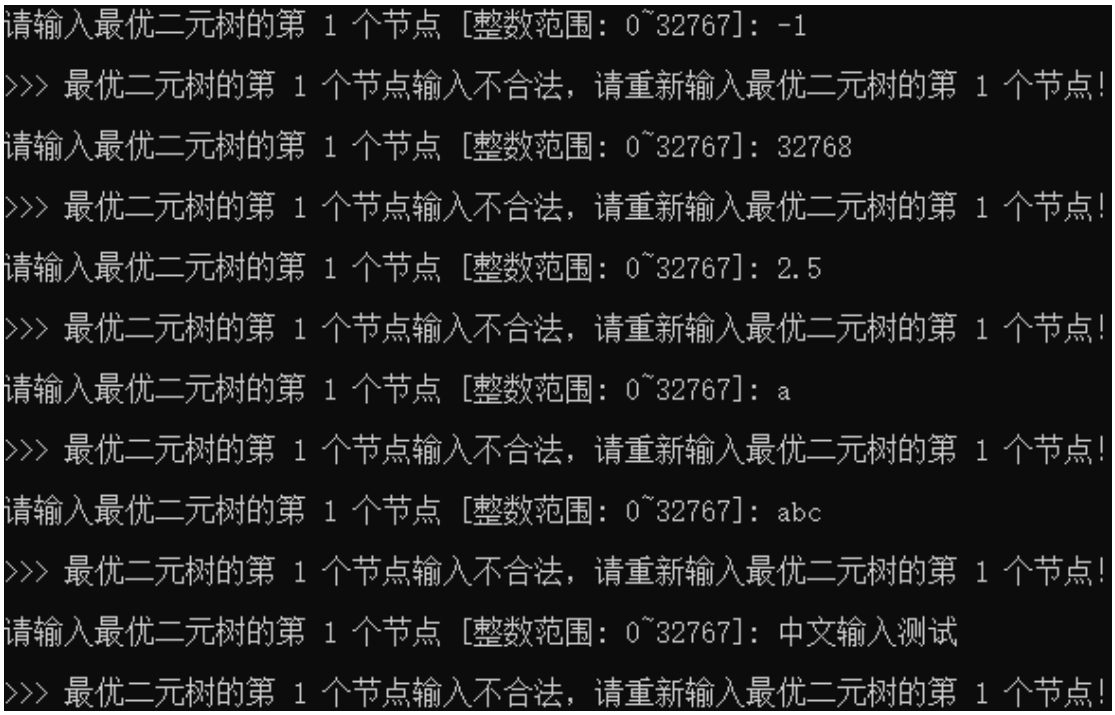

图 6.2 实验数据 2

实验数据分析 2：分别输入超过上下限的整数、浮点数、字符、字符串，可以验证程序对输入非法的情况进行了处理，并要求用户重新输入最优二元树节点值。

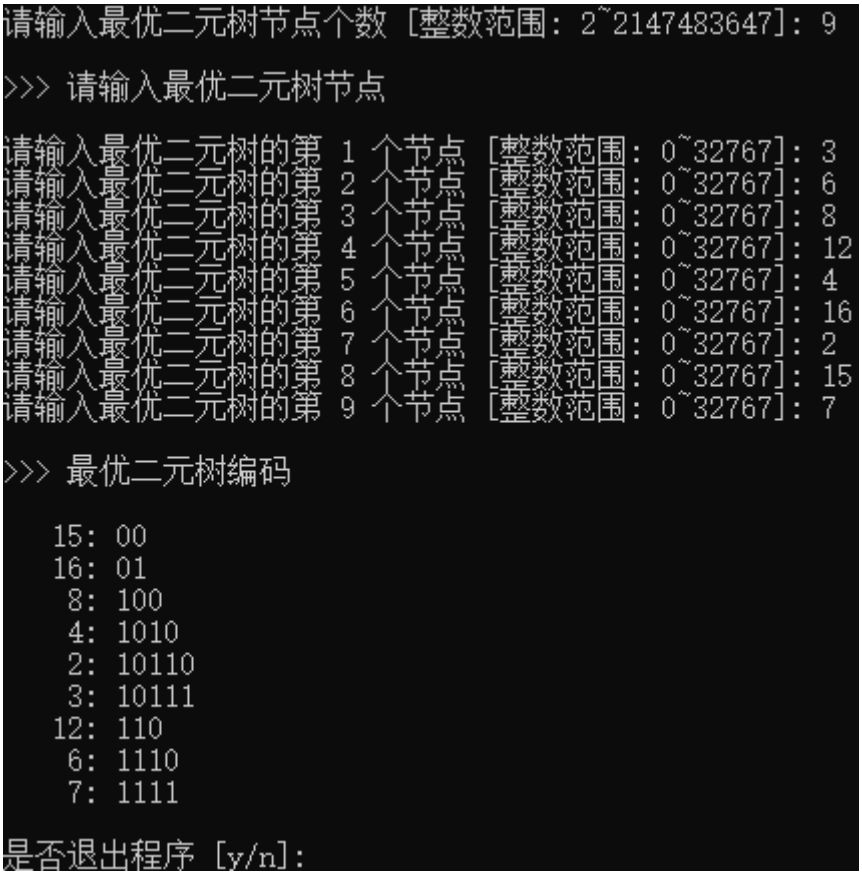

图 6.3 实验数据 3

实验数据分析 3：输入最优二元树的 9 个节点（3、6、8、12、4、16、2、15、7）构建最优二元树（见图 6.4 最优二元树示意图），程序输出最优二元树编码（15：00、16：01、8：100、4：1010、2：10110、3：10111、12：110、6：1110、7：1111）。

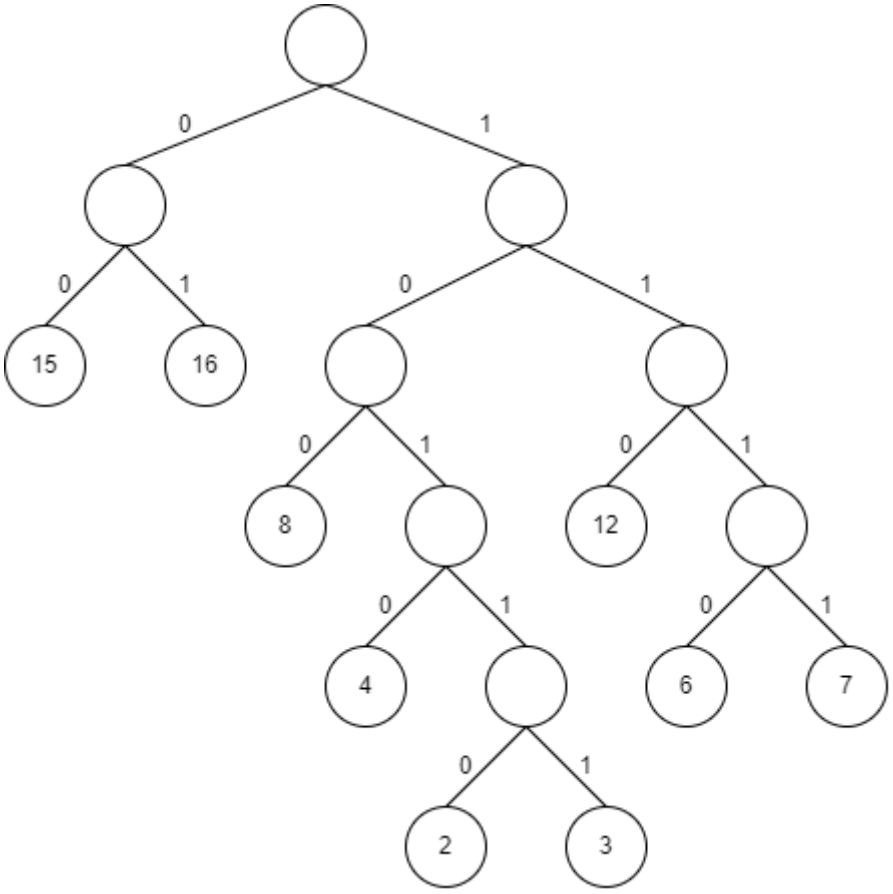

图 6.4 最优二元树示意图

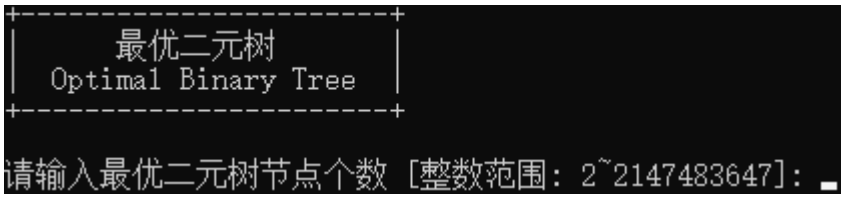

图 6.5 实验数据 4

实验数据分析 4：当输入 n 时，程序清空屏幕并执行下一次最优二元树的构建。

图 6.6 实验数据 5

实验数据分析 5：当输入 y 时，程序退出。

7 实验心得

在本次最优二元树实验中，我获得了许多宝贵的经验和深刻的理解。以下是我的实验心得：

(1)数据结构应用：这次实验使我深入理解了二叉树，特别是哈夫曼树在数据处理和存储中的实际应用。通过亲手构建哈夫曼树，我更加清晰地理解了其数据结构和工作原理。

(2)编程技能提升：在实验中，我使用 C++语言编写代码，这对于增强我的编程技能和理解面向对象的编程概念大有裨益。处理各种数据结构，特别是指针和智能指针的使用，加深了我对内存管理的认识。

(3)算法理解实现：通过本实验，我学习了构建哈夫曼树的算法，并亲自实现了它。这不仅提高了我的算法理解能力，也锻炼了我解决问题和实现复杂算法的能力。在实现哈夫曼树的过程中，我使用递归来遍历树和生成编码，这加深了我对递归概念的理解。

(4)用户交互设计：实验的一个重要方面是设计用户友好的交互界面。我学会了如何有效地收集用户输入并提供清晰的输出，这是软件开发中一个重要的技能。

(5)问题解决调试：在实验过程中，我遇到了各种各样的错误和问题。通过调试和修改代码，我提高了自己的问题解决能力。

(6)理论实践结合：这次实验是理论知识和实践技能结合的极好示例。它使我理解了理论知识在实际编程中的应用，以及如何将复杂的理论转化为实际的代码。

8 程序源文件

```
/*****************************************************************
 * Project Name:  Assignment_5
 * File Name:     assignment_5.cpp
 * File Function: 最优二元树
 * Author:        Jishen Lin (林继申)
 * Update Date:   2023/12/22
 *****************************************************************/

#define _CRT_SECURE_NO_WARNINGS

#include <iostream>
```

```cpp
#include <vector>
#include <string>
#include <algorithm>
#include <memory>
#include <conio.h>
#include <limits>
#include <iomanip>

/* Define TreeNode structure */
struct TreeNode {
    int num;
    std::shared_ptr<TreeNode> left, right;
    TreeNode(int n) : num(n), left(NULL), right(NULL) {}
};

/*
 * Function Name:    initNodes
 * Function:         Initialize tree nodes
 * Input Parameters: std::vector<std::shared_ptr<TreeNode>>& nodes
 *                   const std::vector<int>& frequencies
 * Return Value:     void
 */
void initNodes(std::vector<std::shared_ptr<TreeNode>>& nodes, const
std::vector<int>& frequencies)
{
    for (int frequency : frequencies)
        nodes.push_back(std::make_shared<TreeNode>(frequency));
}

/*
 * Function Name:    insertSorted
 * Function:         Sort a vector of shared pointers to TreeNode in
ascending order
 * Input Parameters: std::vector<std::shared_ptr<TreeNode>>& nodes
 * Return Value:     void
 */
void insertSorted(std::vector<std::shared_ptr<TreeNode>>& nodes)
{
    std::sort(nodes.begin(), nodes.end(), [](const
std::shared_ptr<TreeNode>& a, const std::shared_ptr<TreeNode>& b) {
        return a->num < b->num;
        });
}
```

```
/*
 * Function Name:    constructTree
 * Function:         Construct an optimal binary tree
 * Input Parameters: std::vector<std::shared_ptr<TreeNode>>& nodes
 * Return Value:     a shared pointer to the root node
 */
std::shared_ptr<TreeNode>
constructTree(std::vector<std::shared_ptr<TreeNode>>& nodes)
{
    while (nodes.size() > 1) {
        /* Sort the nodes in the list */
        insertSorted(nodes);

        /* Take the first node as the left child and remove it from the
list */
        auto left = nodes.front();
        nodes.erase(nodes.begin());

        /* Take the next node as the right child and remove it from the
list */
        auto right = nodes.front();
        nodes.erase(nodes.begin());

        /* Create a new parent node whose value is the sum of the left
and right child's values */
        auto parent = std::make_shared<TreeNode>(left->num +
right->num);

        /* Assign the left and right children to the parent node */
        parent->left = left;
        parent->right = right;

        /* Add the parent node back to the list of nodes */
        nodes.push_back(parent);
    }
    return nodes.front();
}

/*
 * Function Name:    printCodes
 * Function:         Print the binary prefix codes
 * Input Parameters: const std::shared_ptr<TreeNode>& node
 *                   const std::string& prefix
 * Return Value:     void
```

```
 */
void printCodes(const std::shared_ptr<TreeNode>& node, const
std::string& prefix)
{
    if (node) {
        printCodes(node->left, prefix + "0");
        if (!node->left && !node->right)
            std::cout << std::setw(5) << node->num << ": " << prefix <<
std::endl;
        printCodes(node->right, prefix + "1");
    }
}

/*
 * Function Name:    inputInteger
 * Function:         Input an integer
 * Input Parameters: int lowerLimit
 *                   int upperLimit
 *                   const char* prompt
 * Return Value:     an integer
 */
int inputInteger(int lowerLimit, int upperLimit, const char* prompt)
{
    while (true) {
        std::cout << "请输入" << prompt << " [整数范围: " << lowerLimit
<< "~" << upperLimit << "]: ";
        double tempInput;
        std::cin >> tempInput;
        if (std::cin.good() && tempInput == static_cast<int>(tempInput)
&& tempInput >= lowerLimit && tempInput <= upperLimit) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            return static_cast<int>(tempInput);
        }
        else {
            std::cerr << std::endl << ">>> " << prompt << "输入不合法，请
重新输入" << prompt << "! " << std::endl << std::endl;
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
        }
    }
}
```

```
/*
 * Function Name:    inputLogicalValue
 * Function:         Input logical value
 * Input Parameters: char falseValue
 *                   char trueValue
 * Return Value:     true / false
 */
bool inputLogicalValue(char falseValue = '0', char trueValue = '1')
{
    while (true) {
        char optn = _getch();
        if (optn == 0 || optn == -32)
            optn = _getch();
        else if (optn == falseValue || optn == trueValue) {
            std::cout << optn << std::endl << std::endl;
            return optn == falseValue ? false : true;
        }
    }
}

/*
 * Function Name:    main
 * Function:         Main function
 * Return Value:     0
 */
int main()
{
    do {
        /* System entry prompt */
        system("cls");
        std::cout << "+----------------------+" << std::endl;
        std::cout << "|      最优二元树       |" << std::endl;
        std::cout << "|  Optimal Binary Tree  |" << std::endl;
        std::cout << "+----------------------+" << std::endl <<
std::endl;

        /* Establish optimal binary tree */
        int num = inputInteger(2, INT_MAX, "最优二元树节点个数");
        std::cout << std::endl << ">>> 请输入最优二元树节点" << std::endl
<< std::endl;
        std::vector<int> frequencies(num);
        for (int i = 0; i < num; i++) {
            char tmp[64];
```

```
            sprintf(tmp, "最优二元树的第 %d 个节点", i + 1);
            frequencies[i] = inputInteger(0, SHRT_MAX, tmp);
        }
        std::vector<std::shared_ptr<TreeNode>> nodes;
        initNodes(nodes, frequencies);
        auto root = constructTree(nodes);

        /* Print the binary prefix codes */
        std::cout << std::endl << ">>> 最优二元树编码" << std::endl <<
std::endl;
        printCodes(root, "");

        /* Whether to exit the program */
        std::cout << std::endl << "是否退出程序 [y/n]: ";
    } while (!inputLogicalValue('n', 'y'));
    return 0;
}
```