**项目说明文档**

**数据结构课程设计**

**——修理牧场**

### 作 者 姓 名 尹诚成

学 号 2351279

指 导 教 师 张 颖

学 院 专 业 计算机科学与技术学院 软件工程



二〇二三 年 十二 月 十三 日

**目录**

[1.项目分析 1](#_bookmark0)

* 1. .[项目背景分析 1](#_bookmark1)
  2. .[项目需求分析 1](#_bookmark2)
  3. .[项目功能分析 1](#_bookmark3)

1.3.1.建立哈夫曼最小生成树功能 1

1.3.2.异常处理功能 1

[2.项目设计 2](#_bookmark12)

* 1. .[数据结构设计 2](#_bookmark13)
  2. .[结构体与类设计 2](#_bookmark14)

2.2.1 HFNode结构体的设计 2

2.2.2 HFTree类的设计 2

* 1. .[项目主体架构设计 3](#_bookmark32)

[3.项目功能实现 3](#_bookmark33)

3.1.项目主体架构的实现 3

3.2.建立哈夫曼最小生成树功能的实现 4

3.3.打印哈夫曼最小生成树功能的实现 5

3.4.异常处理功能的实现 5

4.项目测试 5

4.1.输入牧场木头数量功能的测试 5

4.2.输入每块木头长度功能的测试 6

4.3.建立哈夫曼最小生成树功能的测试 7

1. **项目分析**
   1. 项目背景分析

在农业和乡村经济中，木材是一种重要的原材料，尤其在牧场或农场的建设与维护中。农夫在维护牧场时需要修理栅栏或其他结构，这通常涉及到木材的切割。然而，木材的切割费用跟切割的长度成正比，因此如何选择合适的切割方式以最小化费用变得至关重要。本项目旨在通过使用哈夫曼树算法，提供一种高效的策略以确定最小的木材切割成本，并让用户了解分割过程。

* 1. 项目需求分析

基于以上背景分析，本项目需要实现需求如下：

(1)设计一个修理牧场的模拟系统，能够输入牧场需要的木头数量和每块木头的长度；

(2)使用算法优化对木头的切割过程，以达到总成本最低；

(3)提供友好的用户界面，方便用户输入数据和查看结果；

(4)系统需要具备良好的稳定性和安全性，能够处理非法输入等异常情况。

* 1. 项目功能分析

本项目旨在通过使用哈夫曼算法建立最小生成树，并考虑用户界面设计，实现修理牧场模拟系统。下面对项目的功能进行详细分析。

* + 1. 建立哈夫曼最小生成树功能

哈夫曼算法是一种用于数据压缩的有效方法，其核心是生成哈夫曼树以实现最优前缀编码。哈夫曼树是一种特殊的二叉树，用于对字符或符号进行编码，确保最常用的字符使用较短的编码，从而减少整体数据的存储空间。哈夫曼算法适用于本项目，原因如下：

(1)最优编码：哈夫曼算法能够确保生成的编码是最优的，即所需的总位数最低。通过根据字符的出现频率来构建哈夫曼树，使得频繁出现的字符拥有较短的编码，从而有效减少了数据大小。

(2)高效处理能力：相较于其他编码方式，哈夫曼算法在处理大量数据时表现出更高的效率。它能够快速地对输入字符的频率进行统计，并生成相应的编码，特别适合用于处理大规模文本或数据集。

(3)灵活性与适用性：哈夫曼算法可以灵活地应用于不同类型的数据压缩，包括图像、音频和文本文件。其生成的前缀编码没有歧义，便于解码，确保了数据传输的可靠性。

通过哈夫曼算法，程序将能够实现高效的数据压缩方案，减小存储需求，并提高数据传输的效率。该算法在实际应用中广泛用于各种数据压缩工具和文件格式中，例如ZIP文件和MP3音频文件的编码等。

* + 1. 异常处理功能

实现异常处理机制，处理用户可能输入的非法信息，确保系统的稳定性和安全性。

1. **项目设计**

2.1.数据结构设计

基于项目分析，在构建哈夫曼树的过程中，本项目采用数组来表示哈夫曼树的节点，原因如下：

(1)适用于频繁访问的场景：哈夫曼树的节点存储了字符及其对应的频率，使用数组可以方便地进行频率统计和节点更新。

(2)便于操作与合并：在构建哈夫曼树的过程中，常需要合并权重最小的两个节点，通过数组可以轻松地进行最小值查找和节点的合并操作，极大地简化了算法的实现。

(3)简化存储与检索：使用数组存储哈夫曼树的节点可以方便地进行索引访问，能够使算法的实现更加直观，提升代码的可读性和维护性。

通过采用数组，本项目能够高效地构建哈夫曼树，生成最优编码，同时保持代码的清晰性与灵活性。

2.2.结构体与类设计

2.2.1. HFNode结构体的设计

HFNode 结构体是一个用于辅助哈夫曼算法设计的结构体，用于表示树中每一个节点包含的数据（如字符的使用频率），左右子节点和父节点。其数据成员，构造函数定义及含义如下：

T key：节点的关键码，如使用频率

E data：节点的数据

int left, right, parent：节点的左子节点，右子节点和父节点

HFNode(){ key = 0; data = 0; left = right = parent = -1; }：构造函数，初始化节点的关键码和数据均为0，且无子节点和父节点。

2.2.2.HFTree 类的设计

HFTree 类是一个用于构建霍夫曼树（Huffman Tree）的模板类，广泛应用于数据压缩等领域。它主要通过构造霍夫曼树实现对数据的最优编码，具有线程安全、效率高的特点。其数据成员、构造函数、析构函数、公有成员函数定义及含义如下：

HFNode<T, E>\* root：指向霍夫曼树的根节点。

HFNode<T, E>\* elem：指向当前操作的节点。

int N：记录树中节点的数量。

HFNode<T, E>\* elemList：存储用于构建霍夫曼树的节点集合。

HFTree()：默认构造函数，用于初始化霍夫曼树的根节点、元素以及相关属性。

void createHFTree()：构建霍夫曼树的核心函数。该函数会根据输入的权重构建最优的霍夫曼树结构，调用合适的算法将节点连接起来。

int calulateTotalMoney()：计算并返回霍夫曼树编码所需的总成本，该成本通常与树的权重及其结构相关。

void outputEachStep(HFNode<T, E>\* node)：用于输出建立霍夫曼树过程中的每一步骤，主要用于调试和分析。

HFNode<T, E>\* getRoot()：返回霍夫曼树的根节点指针，便于外部访问和使用树的结构。

2.3.项目主体架构设计

项目主体架构设计为：

(1)创建牧场；

(2)输入牧场需要木头个数；

(3)输入牧场各个木头的长度；

(4)建立哈夫曼最小生成树；

(5)打印哈夫曼最小生成树；

(6)退出程序。

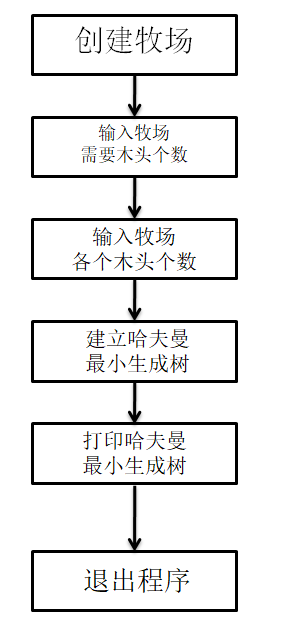


图 2.3 项目主体架构设计流程图

**3.项目功能实现**

3.1.项目主体架构的实现

项目主体架构实现思路为：

(1)创建牧场：在程序的开头，通过打印欢迎信息向用户介绍系统，为用户创建牧场的过程提供指引；

(2)输入牧场需要木头个数：提示用户输入牧场所需木头个数。确保用户输入的是一个有效的整数，并且在指定的范围内。用户输入的数值决定了牧场所需木头个数；

(3)输入牧场各个木头的长度：进入一个循环，每次循环输入每根木头的长度。直到所有木头的长度都被输入。长度数据存储在HFTree 类的实例中；

(4)建立哈夫曼最小生成树：调用HFTree 类的createHFTree函数，建立哈夫曼最小生成树。该方法负责找出成本最低的修理牧场的方法

(5)打印哈夫曼最小生成树：outputEachStep方法被调用以打印出构建的最小生成树。这个步骤在屏幕上展示了每一步该如何切割木头使总成本达到最小。

(6)退出程序。

3.2.建立哈夫曼最小生成树功能的实现

建立哈夫曼最小生成树功能的函数为HFTree 类的成员函数createHFTree，建立哈夫曼最小生成树功能实现的思路为：

(1)初始化元素列表（elemList）：函数开始时，首先将每个输入的元素（通常是待编码的数据及其对应的权重）复制到 elemList 中。对于未初始化的节点（即从 N 到 2 \* N - 1 的节点），将其 key、data 和 parent 属性设置为0，并将左、右子节点和父节点索引初始化为-1，以标记这些节点尚未连接。

(2)创建霍夫曼树：函数进入一个循环，迭代构建霍夫曼树，直到树中只有一个节点为止（即全部元素合并为一棵树）。在每次迭代中：

①寻找最小权重节点：在 elemList 中查找两个权重最小的节点，也就是 key 值最小的两个节点。此时，更新 min1 和 min2 来保持追踪这两个节点及其索引 index1 和 index2。

②链接节点：将找到的两个最小节点的 parent 属性指向新创建的父节点，同时设置新父节点的左右子节点为这两个节点。

③更新权重：新符号的权重（key）为这两个节点权重之和，相关数据（data）也进行相应的更新。

(3)设置根节点：循环结束后，根节点被设置为 elemList 的最后一个元素（即霍夫曼树的根）。

这种构造方式利用了哈夫曼树的特点，通过贪心算法不断选择权重最低的节点合并，最终形成最优的哈夫曼编码树。

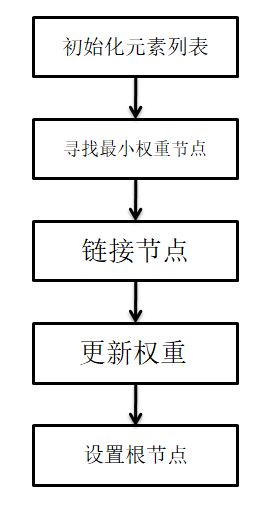


图 3.2 建立哈夫曼最小生成树功能流程图

3.3.打印哈夫曼最小生成树功能的实现

打印哈夫曼最小生成树功能的函数为outputEachStep，打印哈夫曼最小生成树功能实现思路为：

(1)静态变量初始化：使用 static int step 变量来记录当前步骤，这意味着该变量在函数调用之间保持其值，能够方便地跟踪切割步骤的进展。当 step == 1 时，输出修理牧场的最小花费和修理过程的开头信息。调用 calulateTotalMoney 函数来获取总费用，并输出根节点（即木头的总长度）。

(2)如果当前节点是一个叶子节点（即 left 和 right 均为 -1），则表示无进一步的分割，直接返回，终止该分支的递归。

(3)输出当前步骤的信息：输出当前步骤的编号、当前节点的长度、分割后两个部分的新长度以及本次切割的费用。

(4)对当前节点的左子节点和右子节点进行递归调用，逐步输出其分割过程。

outputEachStep 函数通过递归方式展示了霍夫曼树构建的每一步，输出的内容包含节点的长度信息及其分支结果，帮助用户清晰地理解编码过程中各个节点的分割情况和费用。这种结构使得木头分割过程一目了然，有助于提升对霍夫曼树构造及其逻辑的理解。该函数的设计精简而高效，能够很好地为树的构建提供透明度。

3.4.异常处理功能的实现

在进行HFTree类中私有数据成员等的动态内存申请时，程序使用new(std::nothrow)来尝试分配内存。new(std::nothrow)在分配内存失败时不会引发异常，而是返回一个空指针（NULL或nullptr），代码检查指针是否为空指针，如果为空指针，意味着内存分配失败，这时程序将执行以下操作：

(1)向标准错误流std::cerr输出一条错误消息"Error: Memory allocation failed."，指出内存分配失败；

(2)调用exit函数，返回错误码MEMORY\_ALLOCATION\_ERROR（通过宏定义方式定义为-1），用于指示内存分配错误，并导致程序退出。

**4.项目测试**

4.1.输入牧场木头数量功能测试

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。

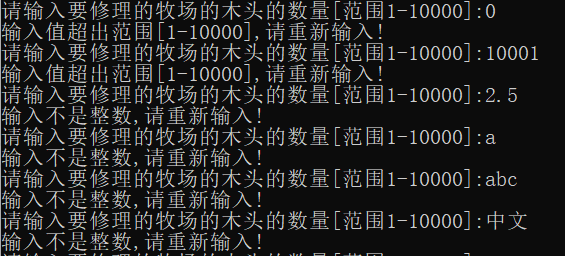


图 4.1.1 输入牧场木头数量功能测试（输入非法）

当输入合法时，程序继续运行。

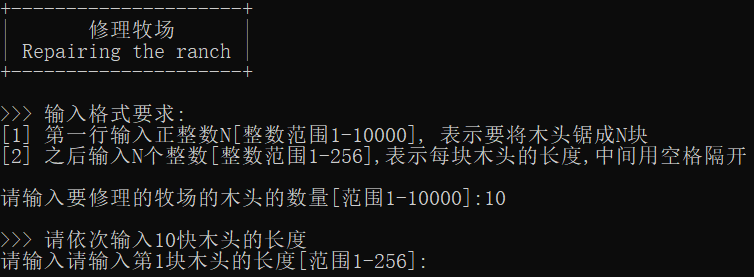


图 4.1.2 输入牧场木头数量功能测试（输入合法）

4.2. 输入每块木头长度功能测试

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。

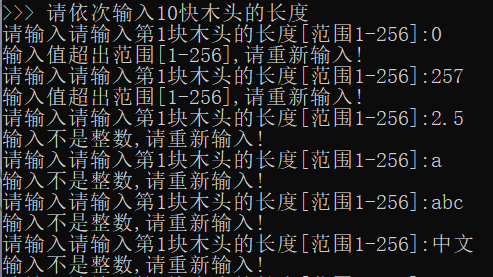


图 4.2.1 输入每块木头长度功能测试（输入非法）

当输入合法时，程序继续运行。

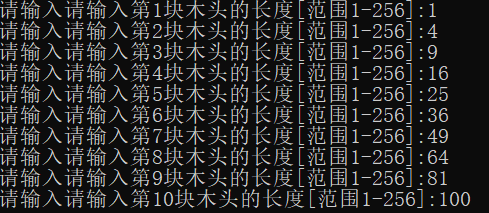


图 4.2.2输入每块木头长度功能测试（输入合法）

4.3.建立哈夫曼最小生成树功能测试

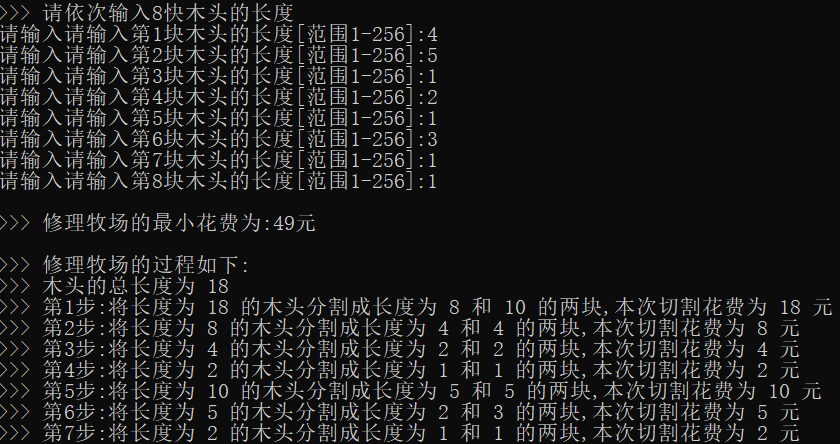


图 4.3.1 建立哈夫曼最小生成树功能测试