项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 谢宇翔

学 号： 1951708

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

同济大学

Tongji University

目 录

[1 分析 1](#_Toc2903)

[1.1 背景分析 1](#_Toc27886)

[1.2 功能分析 1](#_Toc13427)

[2 设计 1](#_Toc7930)

[2.1 数据结构设计 1](#_Toc19027)

[2.2 类结构设计 1](#_Toc15597)

[2.3 成员与操作设计 2](#_Toc9954)

[2.4 系统设计 2](#_Toc1193)

[3 实现 3](#_Toc13714)

[3.1 读取木头的数量和每段木头的长度 3](#_Toc11401)

[3.1.1 功能流程图 3](#_Toc12607)

[3.1.2 核心代码 4](#_Toc4670)

[3.2 构建霍夫曼树 6](#_Toc24076)

[3.2.1 功能流程图 6](#_Toc14768)

[3.2.2 构建霍夫曼树核心代码 6](#_Toc1872)

[3.3 计算价格功能的实现 9](#_Toc25529)

[3.3.1 计算价格功能流程图 9](#_Toc2513)

[3.3.2 计算价格功能核心代码 9](#_Toc1019)

[3.3.3 计算功能截图示例 10](#_Toc13528)

[4 测试 10](#_Toc22047)

[4.1 功能测试 10](#_Toc24805)

[4.2 边界测试 10](#_Toc5684)

[4.2.1 只有一块木头 10](#_Toc20751)

[4.2.2 所有木头一样长 11](#_Toc15644)

[4.3 出错测试 11](#_Toc23631)

[4.3.1 木头个数错误 11](#_Toc26902)

[4.3.2 木头长度错误 12](#_Toc1615)

# 1 分析

## 1.1 背景分析

农夫要修理牧场的一段栅栏，他测量了栅栏，发现需要N块木头，每块木头长度为整数*Li*个长度单位，于是他购买了一个很长的，能锯成N块的木头，即该木头的长度是*Li*的总和。

但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。为简单起见，不妨就设酬金等于所锯木头的长度。

请通过计算得出花费最小的锯木头的方案

## 1.2 功能分析

修理农场程序首先应能允许用户输入木头要锯成的总块数，并能读入每块木 头的长度。之后应该能够使用恰当的算法计算出花费最少的情况并输出给用户。 并应该能够对用户输入的木头长度做简短的判断，保证题目的正确性，如果 木头长度有误应该提示用户输入错误，并能正确终止程序。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，该系统要频繁的寻找当前花费最少的两块木头，算出它们的总花费，并把合成的木头重新加入总木头中。 因此，该题采用 Huffman 树作为数据结构，不仅可以使核心的算法更突出， 而且还可以使代码更简洁美观。

## 2.2 类结构设计

设计Huffman类，对于有N块木头的程序，开辟2\*N的空间，从N+1的位置，每次将2个较小的节点合并存入存到此位置，到2\*N-1刚好能全部合并结束，再将所有没孩子节点节点的weight加起来便是所需要的费用。

## 2.3 成员与操作设计

**霍夫曼树类**

class Huffman

公有成员：

int weight;//权值

int parent;//双亲结点

int lchild;

int rchild;//左右孩子

公有操作：

Huffman()//构造函数

{

this->weight = 0;

this->parent = -1;

this->lchild = -1;

this->rchild = -1;

}

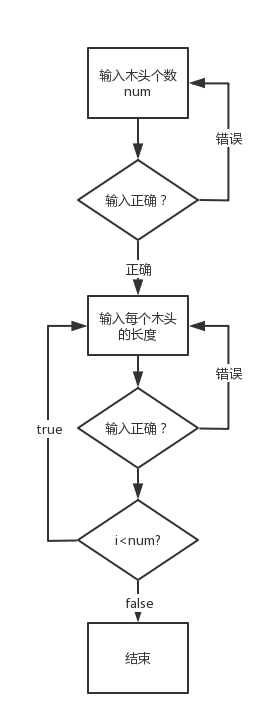
## 2.4 系统设计

首先输入木头有多少段，检查合法性，然后逐个输入木头的长度存储进huffman数组里，然后对huffman数组进行操作，对N到N+1的数组赋值，得出一个huffman结构的数组。然后对数组进行相加输出。

# 3 实现

## 3.1 读取木头的数量和每段木头的长度

### 3.1.1 功能流程图



### 3.1.2 核心代码

int num;

while (1)

{

cout << "请输入木头有多少段:(小于等于10000)" << endl;

cin >> num;//段数

if (!cin.good())

{

cout << "输入错误请重新输入！" << endl;

cin.clear();

cin.ignore(100, '\n');

}

else if (num <= 0||num>10000)

{

cout << "请输入一个大于0小于10000的整数！" << endl;

cin.ignore(1000, '\n');

}

else

break;

}

cout << "请输入每段木头的长度:" << endl;

int\* array = new int[num];

for (int i = 0; i < num; i++)//输入每段长度

{

//cin >> array[i];

while (1)

{

cin >> array[i];

if (cin.fail())

{

cout << "请输入正确数值!请从错误处开始继续输入木头长度:" << endl;

cin.clear();

cin.ignore(1);

}

else if (array[i] <= 0)

{

cout << "请输入正整数！请从错误处开始继续输入木头长度:" << endl;

cin.ignore(1);

}

else

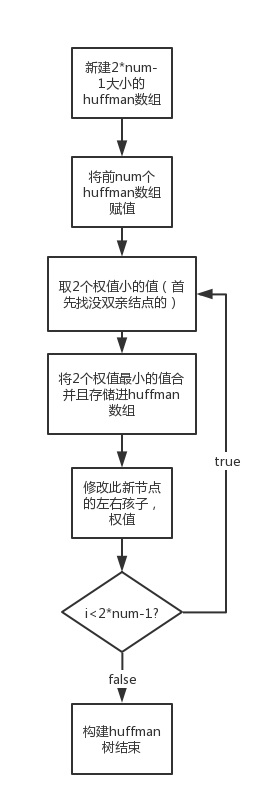
break;

}

}

## 3.2 构建霍夫曼树

### 3.2.1 功能流程图



### 3.2.2 构建霍夫曼树核心代码

void find2min(Huffman\* huffman, int n,int& min1, int& min2)

{

for (int i = 0; i < n; i++)//先找还没双亲的节点

{

if (huffman[i].parent == -1)

{

min1 = i;

break;

}

}

for (int i = 0; i < n; i++)

{

if (huffman[i].parent == -1 && huffman[i].weight < huffman[min1].weight)

{

min1 = i;

}

}

for (int i = 0; i < n; i++)

{

if (huffman[i].parent == -1 && i != min1)

{

min2 = i;

break;

}

}

for (int i = 0; i < n; i++)

{

if (huffman[i].parent == -1 &&

huffman[min2].weight > huffman[i].weight &&

i != min1)

{

min2 = i;

}

}

}

void createhuffman(Huffman\* huffman, int num, int array[])

{

for (int i = 0; i < num; i++)

{

huffman[i].weight = array[i];

}

for (int i = num; i < 2 \* num - 1; i++)

{

int min1 = 0;

int min2 = 0;

find2min(huffman, i,min1, min2);

huffman[i].weight = huffman[min1].weight + huffman[min2].weight;

huffman[i].lchild = min1;

huffman[i].rchild = min2;

huffman[min1].parent = i;

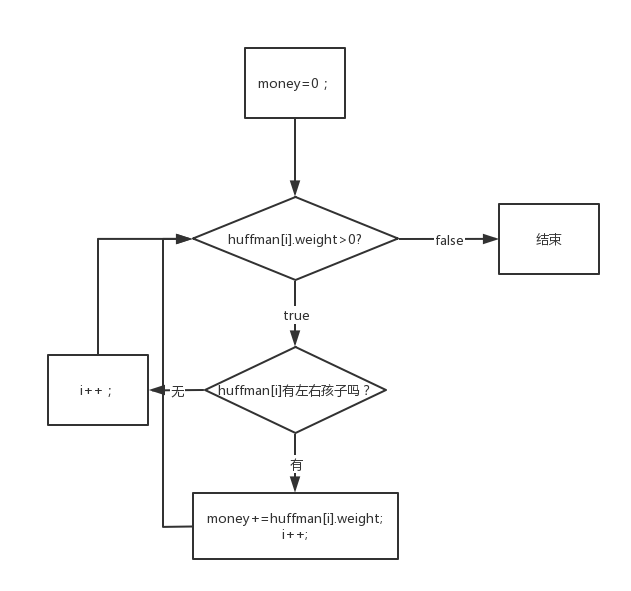
huffman[min2].parent = i;

}

}

## 3.3 计算价格功能的实现

### 3.3.1 计算价格功能流程图



### 3.3.2 计算价格功能核心代码

void caltree(Huffman\* huffman)

{

for (int i = 0; huffman[i].weight > 0; i++)

{

if (huffman[i].lchild != -1 && huffman[i].rchild != -1)

{

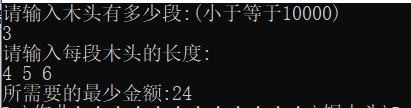
money += huffman[i].weight;

}

}

}

### 3.3.3 计算功能截图示例



# 4 测试

## 4.1 功能测试

**测试用例**：

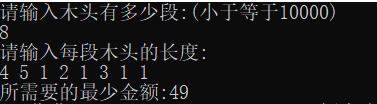
8

4 5 1 2 1 3 1 1

**预期结果**：

49

**实验结果**



## 4.2 边界测试

### 4.2.1 只有一块木头

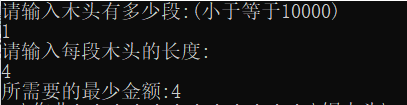
**测试用例：**

**1**

**4**

**预期结果：输出4**

**实验结果：**



### 4.2.2 所有木头一样长

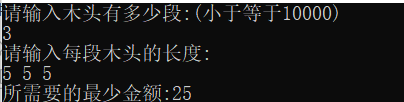
**测试用例：**

**3**

**5 5 5**

**预期结果：**输出25

**实验结果：**



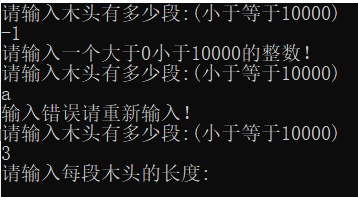
## 4.3 出错测试

### 4.3.1 木头个数错误

**测试用例：**输入木头个数错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**

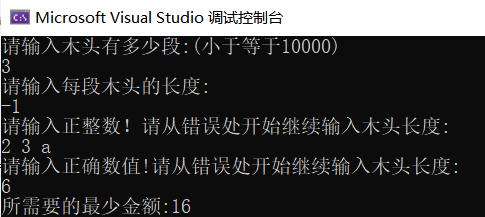


### 4.3.2 木头长度错误

**测试用例：**输入木头长度错误

**预期结果：**程序给出提示信息，程序正常运行不崩溃。

**实验结果：**



# 