项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 罗佳瑞

学 号： 1952528

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](#_Toc9776)

[1.1 背景分析 1](#_Toc8340)

[1.2 功能分析 1](#_Toc32387)

[2 设计 2](#_Toc32017)

[2.1 数据结构设计 2](#_Toc31044)

[2.2 类结构设计 2](#_Toc6724)

[2.3 成员与操作设计 2](#_Toc16109)

[2.4 系统设计 3](#_Toc8355)

[3 实现 4](#_Toc6235)

[3.1 创建功能的实现 4](#_Toc30424)

[3.1.1 创建功能流程图 4](#_Toc30179)

[3.1.2 创建功能核心代码 4](#_Toc9960)

[3.2 遍历功能的实现 6](#_Toc20572)

[3.2.1 遍历功能流程图 6](#_Toc2476)

[3.2.2 遍历功能核心代码 6](#_Toc15142)

[3.3 总体系统的实现 7](#_Toc21290)

[3.3.1 总体系统流程图 7](#_Toc30644)

[3.3.2 总体系统截屏示例 8](#_Toc11912)

[4 测试 9](#_Toc7719)

[4.0 功能测试 9](#_Toc20136)

# 1 分析

## 1.1 背景分析

农夫要修理牧场的一段栅栏，测量栅栏后发现需要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。设酬金等于所锯木头的长度，编写程序生成锯木方案的最优解，得出最少花费。

## 1.2 功能分析

由题意可知，由于一次只能锯一刀，每次的酬金等于总长度，因此可以通过倒推，将所需长度从最小的两部分开始组，直到合成最终的一整根木头，保证被重复收取酬金的长度最短，以达到分割时酬金最少的目的。

# 2 设计

## 2.1 数据结构设计

如上功能分析所述，这一构想的实现，需要多次链接、遍历等操作，在计算机程序中可以选择采用链表存储最优二叉树的数据结构。

## 2.2 类结构设计

树结构的链表一般包括两个抽象数据类型——树结点类（Node）与二叉树类（Tree），而两个类之间的耦合关系可以采用嵌套关系。为方便处理，本程序采用class描述树结点类（Node），使二叉树类（Tree）可以访问树结点。

## 2.3 成员与操作设计

**树结点类（Node）**

**公有成员：**

int num;//结点权值

Node\* Lnode;//左结点指针

Node\* Rnode;//右结点指针

**公有操作：**

Node::Node() :num(-1),Lnode(NULL),Rnode(NULL);//默认构造函数

Node::Node(int n,Node\*L,Node\*R);//构造函数

**友元函数：**

friend void preorder(Node\* p,int &sum);//递归遍历操作

**二叉树类（Tree）**

**私有成员：**

int n;//结点数量

Node\* head;//头结点指针

**公有操作：**

Tree(int m) :n(m),head(NULL) {}//构造函数

~Tree() { delete head; }

void Tree::sort(Node\*\* t, int j); //调整序列函数

void Tree::construct\_tree(Node\*\* t); //生成最优树函数

void Tree::print(); //打印最小费用函数

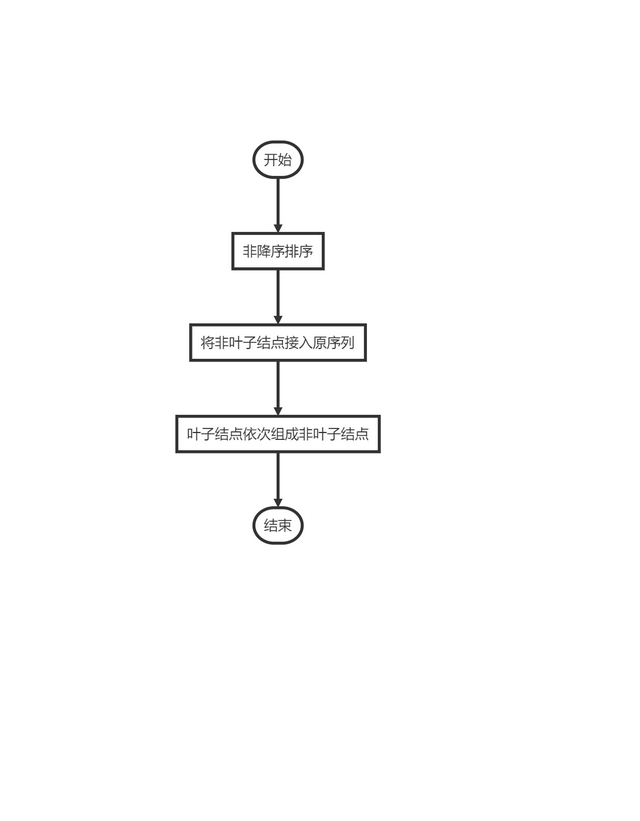
## 2.4 系统设计

系统首先提示用户输入，完成结点权值和结点的创建和输入数据工作，初始化树后，调用二叉树函数（construct\_tree）将其调整为最优二叉树，遍历（preorder）二叉树求取最小耗费，最后调用打印函数（print）输出所求最小耗费。

# 3 实现

## 3.1 创建功能的实现

### 3.1.1 创建功能流程图



### 3.1.2 创建功能核心代码

//生成最优树

void Tree::construct\_tree(Node\*\* t) {

int i=0;

for (;i < n - 1;i++)//非降序排序

for (int j=i+1;j < n ;j++)

if (t[i]->num > t[j]->num)

swap(t[j]->num, t[i]->num);

for (i = 1;i < n;i++)

{

Node\* pt = new Node(t[i - 1]->num + t[i]->num, t[i - 1], t[i]); //生成非叶子结点为从当前最小两值开始的和值;小的为左子女，大的为右子女

t[i] = pt; //用该结点替换掉原子女结点

sort(t, i);//调整序列

}

head = t[n - 1];

//从第n个开始调整序列

void Tree::sort(Node\*\* t, int j) {

int i;

Node\* temp;

for (i = j;i < n - 1;i++)//与后一个树调整,直至倒数第二个

if (t[i]->num > t[i + 1]->num)

{

temp = t[i + 1];

t[i + 1] = t[i];

t[i] = temp;

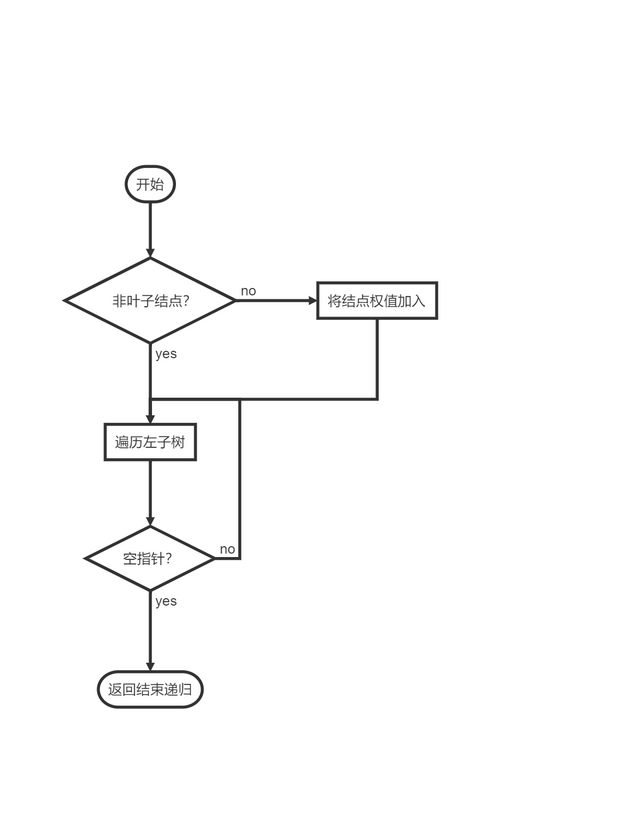
}

}

}

## 3.2 遍历功能的实现

### 3.2.1 遍历功能流程图



### 3.2.2 遍历功能核心代码

void preorder(Node\* p,int&sum)

{

if (p != NULL)//若为空指针直接返回上一层递归

{

if (p->Lnode != NULL&&p->Rnode!=NULL)

sum += p->num;

preorder(p->Lnode,sum);

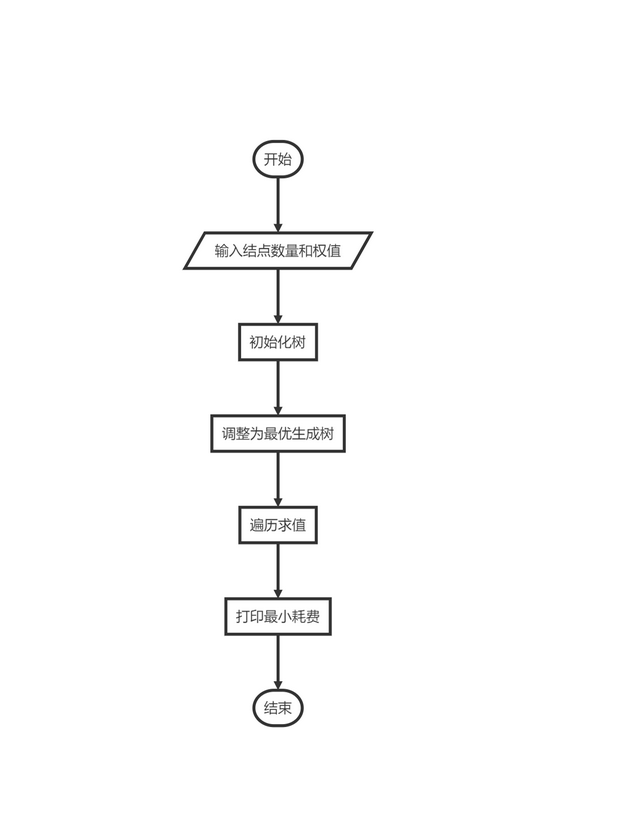
preorder(p->Rnode,sum);

}

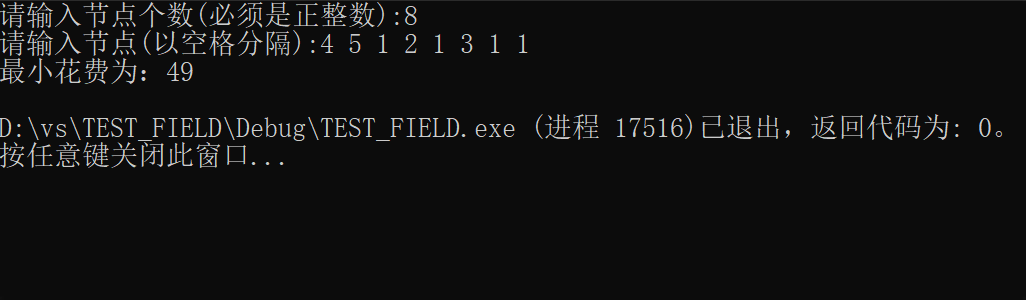
}

## 3.3 总体系统的实现

### 3.3.1 总体系统流程图



### 3.3.2 总体系统截屏示例



# 4 测试

### 4.0 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：**

