项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 张诚睿

学 号： 2150998

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](#_Toc122644739)

[1.1 项目背景分析 1](#_Toc122644740)

[1.2 功能分析 1](#_Toc122644741)

[2 设计 1](#_Toc122644742)

[2.1 数据结构设计 1](#_Toc122644743)

[2.2 类结构设计 1](#_Toc122644744)

[2.3 类具体成员与操作设计 2](#_Toc122644745)

[2.4 main函数设计 2](#_Toc122644746)

[3 实现 3](#_Toc122644747)

[3.1 求解最小花费功能的实现 3](#_Toc122644748)

[3.1.1 功能流程图 3](#_Toc122644749)

[3.1.2 核心代码 4](#_Toc122644750)

[4 测试 4](#_Toc122644751)

[4.1 功能测试 4](#_Toc122644752)

[4.1.1一般锯木要求处理结果输出功能测试 4](#_Toc122644753)

[4.2 边界测试 5](#_Toc122644754)

[4.2.1 要锯成1或2块木头 5](#_Toc122644755)

[4.3 出错测试 5](#_Toc122644756)

[4.3.1 木头总数不是正整数 5](#_Toc122644757)

[4.3.2 木头总数是字符 6](#_Toc122644758)

[4.3.3 有木头长度是字符或负数 6](#_Toc122644759)

# 1 分析

## 1.1 项目背景分析

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，请人锯木的酬金等于所锯木头的长度。

现需要求解将木头锯成N块要求长度的最小花费

## 1.2 功能分析

本项的目的在于实现对输入的木头总段数及每段长度要求求解最小花费锯木方案。分析可知让短的木段反复参与未完成锯木操作的部分会让酬金更低，求解思路贴近于贪心算法，用优先级队列实现每次合并最短的两根木头并求价格，直至合成一根木头，即可获得最小酬金数值。

# 2 设计

## 2.1 数据结构设计



核心数据结构设计如上，通过类-PriorityWoodSolution实现控制一个优先级队列（含带头结点），队列是利用struct Node构成的链表模拟的，结点中用int length来存放木头长度，求解时，链表不断缩短直至仅有两个（不含带头结点） ，最后求解出答案

## 2.2 类结构设计

如上所述，本项目采用struct Node描述链表结点类（LNode），这样使得链表结点类PriorityWoodSolution（LinkList）可以访问链表结点。

## 2.3 类具体成员与操作设计

**链表结点类**

struct Node {

int length;

Node\* next;

};

**链式栈类**

class PriorityWoodSolution {

private:

Node\* head; // 含有首元结点

int cost;

int Num; // 木头数量

void MakeEmpty();

void InitQueue(int\* LengthTmp);

bool IsEmpty() { return !bool(head); };

public:

PriorityWoodSolution() :head(NULL), cost(0), Num(0) {};

~PriorityWoodSolution() { MakeEmpty(); };

void Init();

void Solve();

};

## 2.4 main函数设计

首先建立PriorityWoodSolution对象，之后获取木头总段数及每段要求的长度，如输入错误，则提示重新输入直至输入正确，最后计算结果并输出，主函数返回0。

# 3 实现

## 3.1 求解最小花费功能的实现

### 3.1.1 功能流程图

### 3.1.2 核心代码

while (NodeNum > 2)

{

cost += head->next->length;

cost += head->next->next->length;

// 计算好价格

Node\* New = new Node;

New->length = head->next->length + head->next->next->length;

// 两块最小的木头合到一起

Node\* tmp = head->next; // 记录第一个节点

// 去除前两个节点

head->next = head->next->next->next;

delete tmp->next;

delete tmp;

tmp = head;

while (tmp->next != NULL && New->length > tmp->next->length)

{

tmp = tmp->next; // 新节点比当前节点大，就往后找，直到找到结尾或者比当前节点小于等于

}

/\* 插入在tmp后 \*/

New->next = tmp->next;

tmp->next = New;

NodeNum--;

}

# 4 测试

## 4.1 功能测试

### 4.1.1一般锯木要求处理结果输出功能测试

**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 要锯成1或2块木头

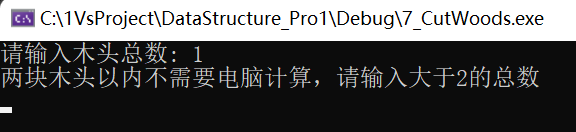
**测试用例：**

1

**预期结果：**

1或2块木头不需要计算最优方案，给出提示信息，要求重新输入

**实验结果：**



## 4.3 出错测试

### 4.3.1 木头总数不是正整数

**测试用例：**

-4

**预期结果：**

输出提示信息，要求重新输入直至正确，程序不崩溃

**实验结果：**



### 4.3.2 木头总数是字符

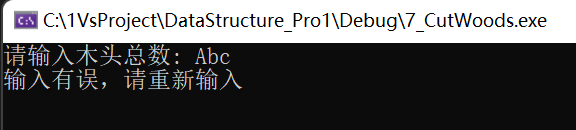
**测试用例：**

Abc

**预期结果：**

输出提示信息，要求重新输入直至正确，程序不崩溃

**实验结果：**



### 4.3.3 有木头长度是字符或负数

**测试用例：**

4

3 a -1 5

**预期结果：**

输出提示信息，要求重新输入直至正确，程序不崩溃

**实验结果：**

