项目说明文档

数据结构课程设计

——修理牧场

作 者 姓 名： 张喆

学 号： 1754060

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](#_Toc532922869)

[1.1 背景分析 1](#_Toc532922870)

[1.2 功能分析 1](#_Toc532922871)

[2 设计 2](#_Toc532922872)

[2.1 数据结构设计 2](#_Toc532922873)

[2.2 类结构设计 2](#_Toc532922874)

[2.3 成员与操作设计 2](#_Toc532922875)

[2.4 系统设计 2](#_Toc532922876)

[3 实现 3](#_Toc532922877)

[3.1 求最小花费功能的实现 3](#_Toc532922878)

[3.1.1 求最小花费功能流程图 3](#_Toc532922879)

[3.1.2 求最小花费功能核心代码 3](#_Toc532922880)

[3.1.3 求最小花费功能截屏示例 4](#_Toc532922881)

[3.2 总体系统的实现 5](#_Toc532922882)

[3.2.1 总体系统流程图 5](#_Toc532922883)

[3.2.2 总体系统核心代码 5](#_Toc532922884)

[3.2.3 总体系统截屏示例 6](#_Toc532922885)

[4 测试 7](#_Toc532922886)

[4.1 功能测试 7](#_Toc532922887)

[4.1.1 求最小花费功能测试 7](#_Toc532922888)

[4.2 边界测试 8](#_Toc532922889)

[4.2.1 只有一块木头 8](#_Toc532922890)

[4.2.2 每一块木头的长度都一样 8](#_Toc532922891)

[4.3 出错测试 8](#_Toc532922892)

[4.3.1 木头的长度非法 8](#_Toc532922893)

[4.3.2 木头总块数输入非法 9](#_Toc532922894)

# 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。例如，要将长度为20的木头锯成长度为8，7和5的三段，第一次锯木头将木头锯成12和8，花费20；第二次锯木头将长度为12的木头锯成7和5花费12，总花费32元。如果第一次将木头锯成15和5，则第二次将木头锯成7和8，那么总的花费是35（大于32）。

这个项目从小的方面说对农夫的帮助很大，可以替他决策出最优的解从而花最少的前修理牧场的栅栏；往大的方面说对很多类似的问题都有很多借鉴意义，可以决策出更优秀的同类型问题。

## 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C++ STL库中的优先队列(priority\_queue)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

设计了农场类（Ranch），在农场类中维护了一个优先队列，并有求最小花费的方法，可以通过农场类对应的对象很简单的处理修理农场的问题。

## 2.3 成员与操作设计

**农场类的构造函数：**

Ranch() = default; //默认构造函数

Ranch(int n); //总块数为n的构造函数

**私有成员：**

int N;

bool judge = false; //用于判断木头的长度是否合理

priority\_queue<int, vector<int>, greater<int> > woods;

//使用优先队列存储

**公有操作：**

bool isCorrect() { return !judge; } //判断木头长度是否输入合法

int minRepairCost(void); //最小花费

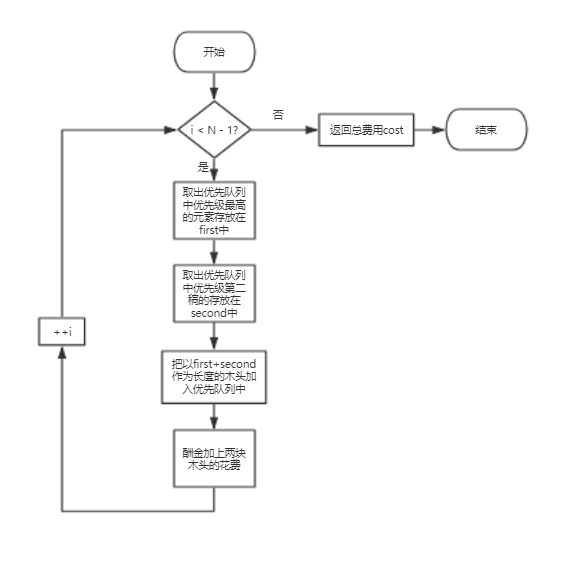
## 2.4 系统设计

系统首先读入用户指定的总木块数量，之后建立参数为总木块数量的农场类，最后调用求最小花费的成员函数求解最小花费问题。

# 3 实现

## 3.1 求最小花费功能的实现

### 3.1.1 求最小花费功能流程图



### 3.1.2 求最小花费功能核心代码

int Ranch::minRepairCost(void)

{

int cost = 0;

int first, second;

for (int i = 0; i < N - 1; ++i)

{

/\*每次取出最小的两块木头\*/

first = woods.top();

woods.pop();

second = woods.top();

woods.pop();

/\*把这两块木头拼成的长木头长度重新加入优先队列中\*/

woods.push(first + second);

/\*酬金加上这两块木头的长度\*/

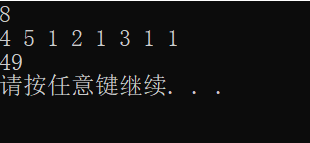
cost += first + second;

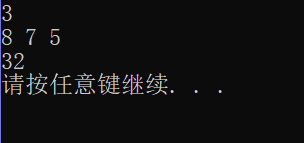
}

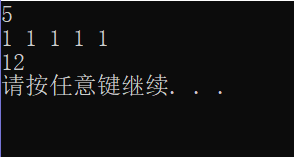
return cost;

}

### 3.1.3 求最小花费功能截屏示例

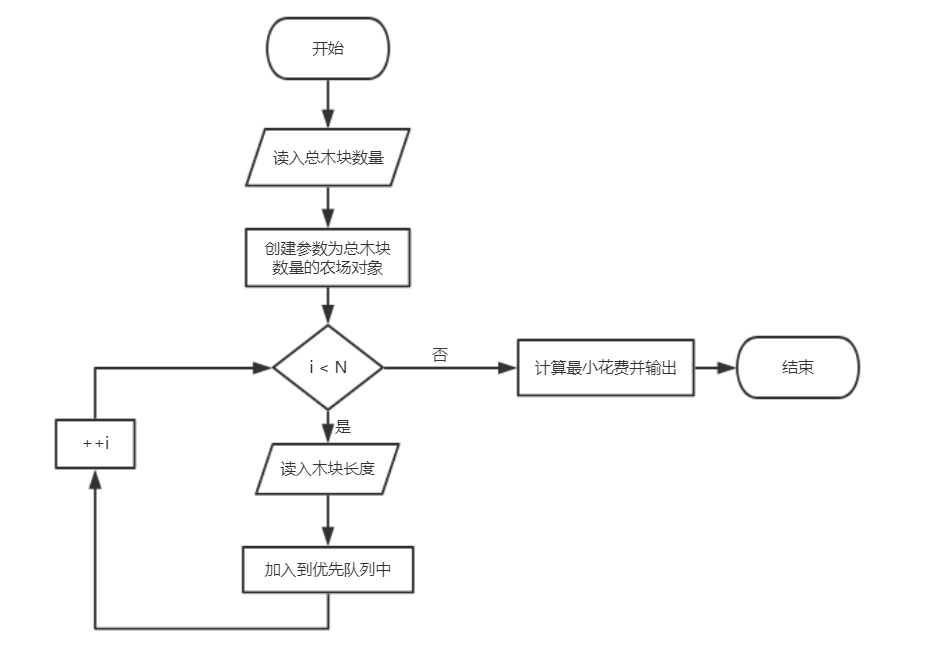






## 3.2 总体系统的实现

### 3.2.1 总体系统流程图



### 3.2.2 总体系统核心代码

int N;

while (1)

{

cin >> N;

if (N > 0) { break; }

else

{

cerr << "总块数错误！请重新输入" << endl;

}

}

Ranch ranch(N);

if (ranch.isCorrect())

{

cout << ranch.minRepairCost() << endl;

}

else

{

cerr << "木头长度输入有误！" << endl;

}

Ranch::Ranch(int n)

:N(n)

{

int buf;

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

{

cin >> buf;

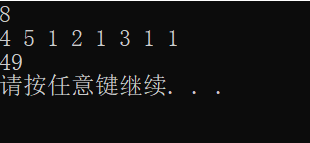
if (buf <= 0) { judge = true; }

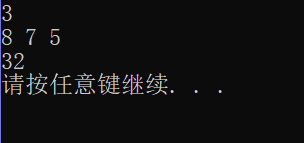
woods.push(buf);

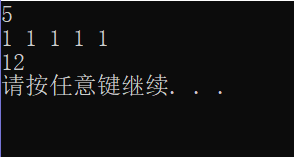
}

}

### 3.2.3 总体系统截屏示例







# 4 测试

## 4.1 功能测试

### 4.1.1 求最小花费功能测试

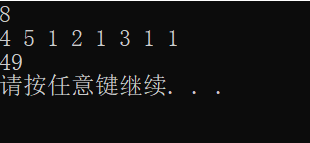
**测试用例**：

8

4 5 1 2 1 3 1 1

**预期结果**：最小花费为49

**实验结果：**



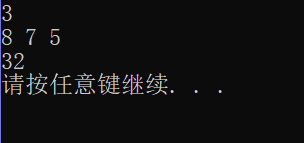
**测试用例**：

3

8 7 5

**预期结果**：最小花费为32

**实验结果：**



## 4.2 边界测试

### 4.2.1 只有一块木头

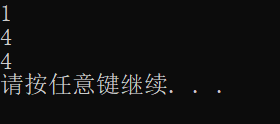
**测试用例：**

1

4

**预期结果：**输出这一块木头的长度作为总花费

**实验结果：**



### 4.2.2 每一块木头的长度都一样

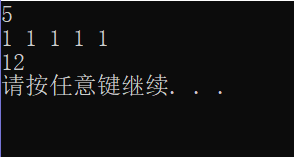
**测试用例：**

5

1 1 1 1 1

**预期结果：**总花费为12

**实验结果：**



## 4.3 出错测试

### 4.3.1 木头的长度非法

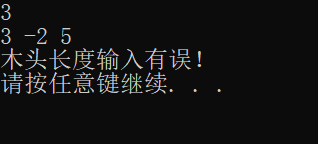
**测试用例：**

3

3 -2 5

**预期结果：**程序不崩溃，并输出提示信息木头长度输入有误

**实验结果：**



### 4.3.2 木头总块数输入非法

**测试用例：**

-4

0

3

5 1 3

**预期结果：**程序不崩溃，并输出提示信息木头总块数输入有误，并允许用户从新输入

**实验结果：**

