**项目说明文档**

**数据结构课程设计**

**——修理牧场**

作 者 姓 名： 张梓瀚

学 号： 2051943

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 项目分析 - 2 -](#_Toc121317891)

[1.1项目背景 - 2 -](#_Toc121317892)

[1.2 项目要求 - 3 -](#_Toc121317893)

[1.2.1 功能要求 - 3 -](#_Toc121317894)

[1.2.2 输入格式 - 3 -](#_Toc121317895)

[1.2.3 输出格式 - 3 -](#_Toc121317896)

[1.2.4 项目示例 - 3 -](#_Toc121317897)

[2 项目设计 - 4 -](#_Toc121317898)

[2.1 修理牧场算法 - 4 -](#_Toc121317899)

[2.1.1 主要思想 - 4 -](#_Toc121317900)

[2.1.2代码 - 5 -](#_Toc121317901)

[2.2 数据结构设计 - 6 -](#_Toc121317902)

[2.3 类设计 - 6 -](#_Toc121317903)

[2.3.1 优先队列类（priQue） - 6 -](#_Toc121317904)

[3 项目测试 - 7 -](#_Toc121317905)

[3.1 一般情况 - 7 -](#_Toc121317906)

[3.2 木头长度全相等 - 7 -](#_Toc121317907)

[3.3 只需锯成两段 - 8 -](#_Toc121317908)

[3.4 最终还是一段 - 8 -](#_Toc121317909)

# 1 项目分析

## 1.1项目背景

## 1.2 项目要求

### 1.2.1 功能要求

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）.

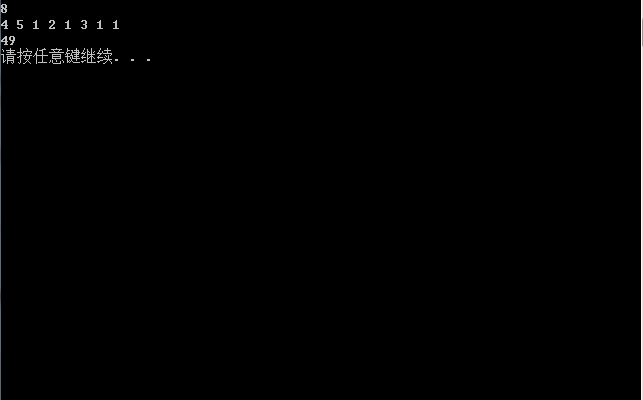
### 1.2.2 输入格式

输入第一行给出正整数N（N<），表示要将木头锯成N块。第二行给出N个正整数，表示每块木头的长度。

### 1.2.3 输出格式

输出一个整数，即将木头锯成N块的最小花费。

### 1.2.4 项目示例



# 2 项目设计

## 2.1 修理牧场算法

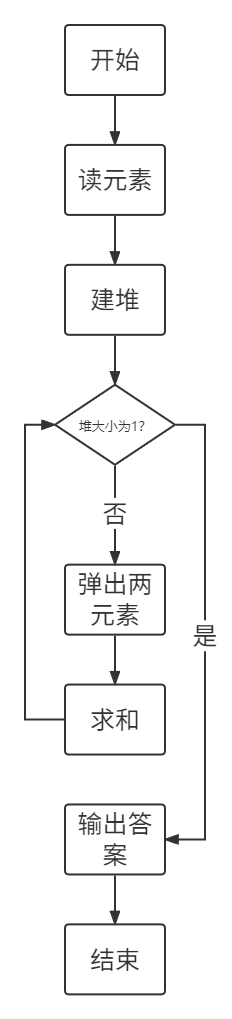
### 2.1.1 主要思想

从题意直接入手找不到切入点，因此尝试逆向思维，按照输入格式将木头先分为最终需要的若干段，再将小段木头两两拼合，最终拼成原有的一长段木头，每次拼接代价为两段短木头的长度之和。不难证明，这样逐个合并的最终代价与题目要求的锯成小段的总代价相同（两者互为逆过程）。

当前有N块木头，最终需要合成为1块，因此每次合并对答案的贡献都是1，但代价不相同，假设现有三块木头（从左到右分别编号为1、2、3）：

如果我们先将较短的1、3合并，再与2合并，那么总合并代价为（1号长度\*2+3号长度\*2+2号长度），如果第一次先合并2号，那么无论怎样合并，代价都会比上述代价更大，因此我们可以得出一个结论：**让长木头参与合并的次数尽可能少，可以使其对答案的贡献最小，以达到我们求得最小代价的目标。**

有了这样的结论，我们可以用优先队列维护短木块的大小有序，每次取出长度最小的两个木块合并，再加入木块集合中，重复这一过程N-1次，每次计算合并代价并求和，就得到了最小代价。



### 2.1.2代码

#include "priQue.h"

#include <iostream>

using namespace std;

int main(){

int num;

cin >> num;

while (cin.fail() || num <= 0)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "请输入正确的正整数N" << endl;

cin >> num;

}

priQue<int> heap(num);

for (int i = 0, j; i < num;i++){

cin >> j;

while (cin.fail() || j <= 0)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "请输入正确的正整数长度" << endl;

cin >> j;

}

heap.push(j);

}

int ans = 0;

while (heap.size() > 1)

{

int t1 = heap.top();

heap.pop();

int t2 = heap.top();

heap.pop();

ans += t1 + t2;

if (heap.empty())

break;

else

heap.push(t1 + t2);

}

cout << ans << endl;

return 0;

## }2.2 数据结构设计

为了维护短木块的大小有序，我们可以采用小顶堆这一数据结构来存储木块长度的数据，每次取堆顶两个元素相加，将其出堆，再将合并后的长度加到堆中，就完成了一次合并操作。

## 2.3 类设计

本题需要实现优先队列（priQue）类，包括初始化堆，弹出堆顶、插入新元素、元素上浮和下沉等操作。

### 2.3.1 优先队列类（priQue）

//优先队列（小顶堆）类

template<typename T>

class priQue

{

private:

//数据

T\* pArray;

//堆大小

int m\_length;

//结点下沉

void swim(int k);

//结点上浮

void sink(int k);

public:

//默认构造函数

priQue();

//指定堆大小的构造函数

priQue(int N);

//取堆顶

T top();

//弹出堆顶

void pop();

//将元素入堆

void push(const T& v);

//判断堆是否为空

bool empty();

//获取堆大小

int size();

};

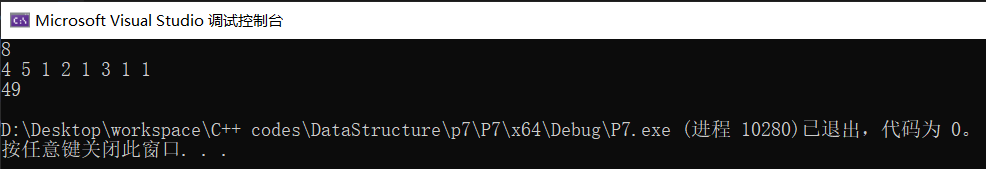
# 3 项目测试

## 3.1 一般情况

测试用例：8 4 5 1 2 1 3 1 1

预期结果：49

实验结果：49

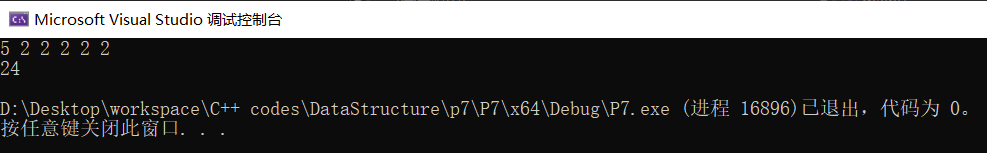


## 3.2 木头长度全相等

测试用例：5 2 2 2 2 2

预期结果：24

实验结果：24

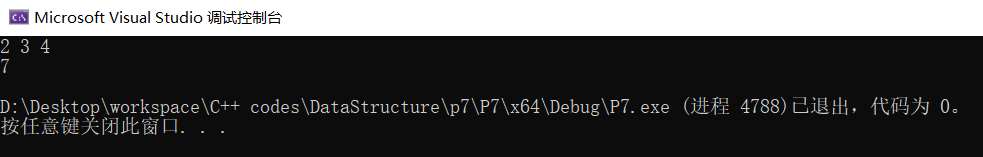


## 3.3 只需锯成两段

测试用例：2 3 4

预期结果：7

实验结果：7



## 3.4 最终还是一段

测试用例：1 10

预期结果：0

实验结果：0

