数据结构课程设计项目说明文档

——修理牧场

作 者 姓 名： 马威

学 号： 2151294

指 导 教 师： 张颖

学院、 专业： 软件学院 软件工程

****

目 录

[1 项目分析 1](#_Toc495668153)

[1.1 项目背景 1](#_Toc495668154)

[1.2 项目要求 1](#_Toc495668154)

[1.3 项目需求分析 2](#_Toc495668155)

[2 项目设计 3](#_Toc495668156)

[2.1 数据结构设计 3](#_Toc495668157)

[2.2 类设计 3](#_Toc495668158)

2.2.1 最小堆类（MinHeap） 3

2.3 算法设计 5

2.3.1 算法思路 5

2.3.2 性能评估 5

2.3.3 准备工作部分 6

·流程图表示 6

·代码实现 6

2.3.4 求解最小花费部分 8

·流程图表示 8

·代码实现 8

[3 项目测试 1](#_Toc495668161)0

[3.1 样例测试 1](#_Toc495668162)0

[3.2 所有木头不等长 1](#_Toc495668166)0

[3.3 所有木头等长 1](#_Toc495668170)0

[3.4 只需要一根木头 1](#_Toc495668174)1

[3.5 只需要两根木头 1](#_Toc495668178)1

[3.6 木头总数输入错误的情况 1](#_Toc495668182)2

[3.7 木头长度输入错误的情况 1](#_Toc495668182)2

**1.项目分析**

1.1 项目背景

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。

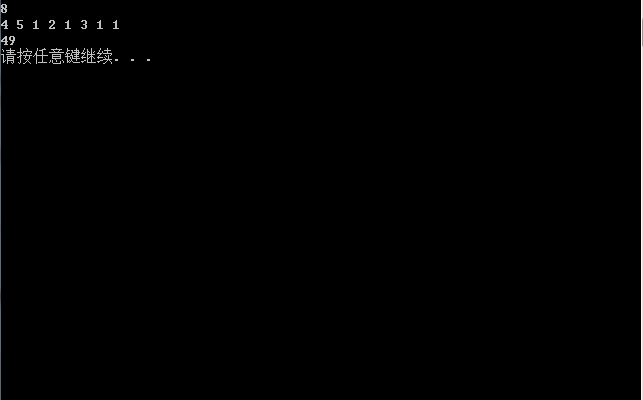
但是农夫自己没有锯子，请人锯木的酬金跟这段木头的长度成正比。为简单起见，不妨就设酬金等于所锯木头的长度。例如，要将长度为20的木头锯成长度为8，7和5的三段，第一次锯木头将木头锯成12和8，花费20；第二次锯木头将长度为12的木头锯成7和5花费12，总花费32元。如果第一次将木头锯成15和5，则第二次将木头锯成7和8，那么总的花费是35（大于32）。

1.2 项目要求

输入格式：输入第一行给出正整数N（N≤104），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。

输出格式：输出一个整数，即将木头锯成N块的最小花费。

测试用例：



1.3 项目需求分析

对于完成修理牧场问题的程序，需考虑以下需求：

**·正确性**

程序应当能够按照要求求解出最小花费并输出

**·健壮性**

程序应当对木头数量、每根木头长度的错误输入进行处理。

程序应当对构建哈夫曼树的特殊情况，如仅有一个或两个元素的情况进行正确的处理。

**·高效性**

问题的求解有特殊要求，需要用特定的求法解决，这就要求要在尽可能短的时间内处理大量的数据。

**2.项目设计**

2.1 数据结构设计

由项目分析可以得出，本项目需要构造一棵哈夫曼树，其叶结点为每根木头的长度，根结点即为木头的总长。而哈夫曼树的构造涉及到选出权值最小的节点，一种优先级队列可以满足此类需求。

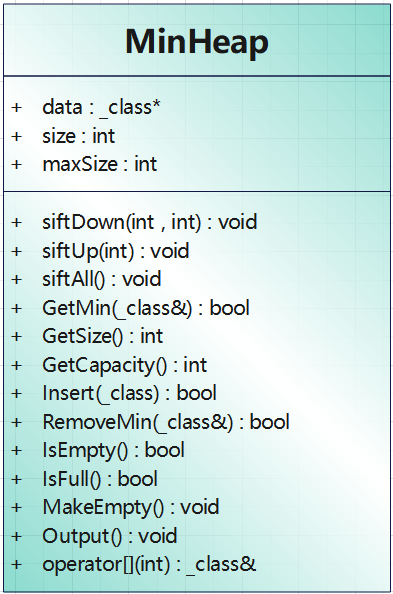
2.2 类设计

**2.2.1 最小堆类（MinHeap）**

最小堆（MinHeap）是一种优先级队列，其逻辑上为二叉树结构，每个结点的值都比其子女小，根结点就是最小值。按照这样的顺序排列，使得最小堆的插入、取出最小值等操作的时间复杂度仅为O(log2n)，性能良好。

每次进行插入或删除操作时，为了保持最小堆的性质，需要频繁地对元素进行访问、比较和交换，而并不关心每个元素的具体位置。因此，虽然最小堆逻辑上是树形结构，但本质上通过数组实现，通过层次往下给出元素的下标，标号为n的左子女（若存在）标号为2n+1，右子女（若存在）标号为2n+2，由此可知，每次进行元素的比较、交换时只需通过下标即可访问元素，非常方便。

其UML图如下：



其中，主要函数如下：

//取堆顶元素，即最小元素

bool GetMin(\_class& x)const;

//取堆的当前大小

int GetSize()const;

//取堆的最大容量

int GetCapacity()const;

//将x插入最小堆中

bool Insert(const \_class x);

//删除堆顶上的最小元素

bool RemoveMin(\_class& x);

//判断堆是否为空

bool IsEmpty()const;

//判断堆是否已满

bool IsFull()const;

//置空堆

void MakeEmpty();

//按数组内存放顺序输出堆内的元素

void Output()const;

//重载函数：下标访问

\_class& operator[](const int pos)const;

2.3 算法设计

**2.3.1 算法思路**

由题意，农夫每切割一次的费用取决于切割后两段木头的长度总和，切出来的两根木头越短，费用就越少。因此，可以采用倒推法，从所需的N根木头出发，每次选择最短的两根木头进行合成（合成后的木头作为一根回到木头的集合中），直到所有木头重新合成为一根为止，即可保证每切一次切出来的都是最短的木头，费用也会是最少的了。

由上述分析可知，该过程是一个构建一棵哈夫曼树的过程，根结点为所需木头的总长，非叶结点为中间长度，叶结点为每根所需木头的长度。在构建过程中同时求解最小花费的方法如下：

（1）取根结点的较小子女

（2）将根结点及其较小子女的权值加到总花费上

（3）从最小堆中移除最小的两个元素

（4）将（3）中新合成的木头放入最小堆中

（5）重复以上步骤直至最小堆中仅有一个元素（该元素即为所需木头的总长）

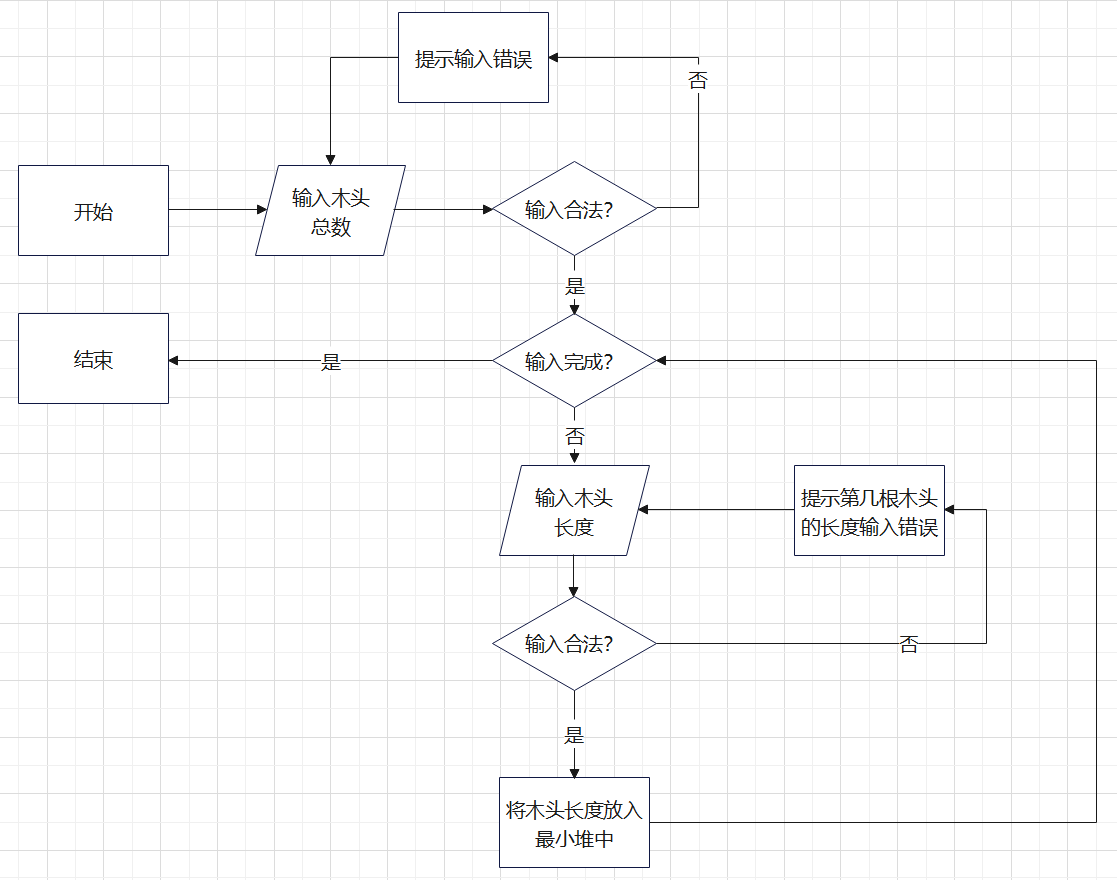
最后将总花费输出，即可得到最小的花费。要注意，若只需要一根木头，从实际出发，农夫不需要再请人切割木头，花费自然为0，上述过程在此情况下得到的花费也为0。

**2.3.2 性能评估**

设总共需要木头数量为n。建立最小堆需要O(nlog**2** n)，每次移除元素、重新插入元素需要O(log**2** n)，每趟的插入、删除操作都是常数次，共n-1趟，所需时间仍然为O(nlog**2** n)。故总的时间复杂度仍为O(nlog**2** n)。

**2.3.3 准备工作部分**

·流程图表示

****

·代码实现

/\*1、输入木头数量\*/

    int num = 0;

    while (1) {

        cin >> num;

        if (cin.good() && num >= 1 && num <= 1000)

            break;

        cin.clear();

        cin.ignore(INT\_MAX, '\n');

        cout << "输入错误，请重新输入！" << endl;

    }

    int\* data = new int[num];

    if (data == NULL) {

        cerr << "存储分配错误！" << endl;

        return 0;

    }

    /\*2、输入每块木头的长度\*/

    for (int i = 0; i < num; i++) {

        int length = -1;

        while (1) {

            cin >> length;

            if (cin.good() && length >= 1 && length <= INT\_MAX) {

                data[i] = length;

                break;

            }

            cin.clear();

            cin.ignore(INT\_MAX, '\n');

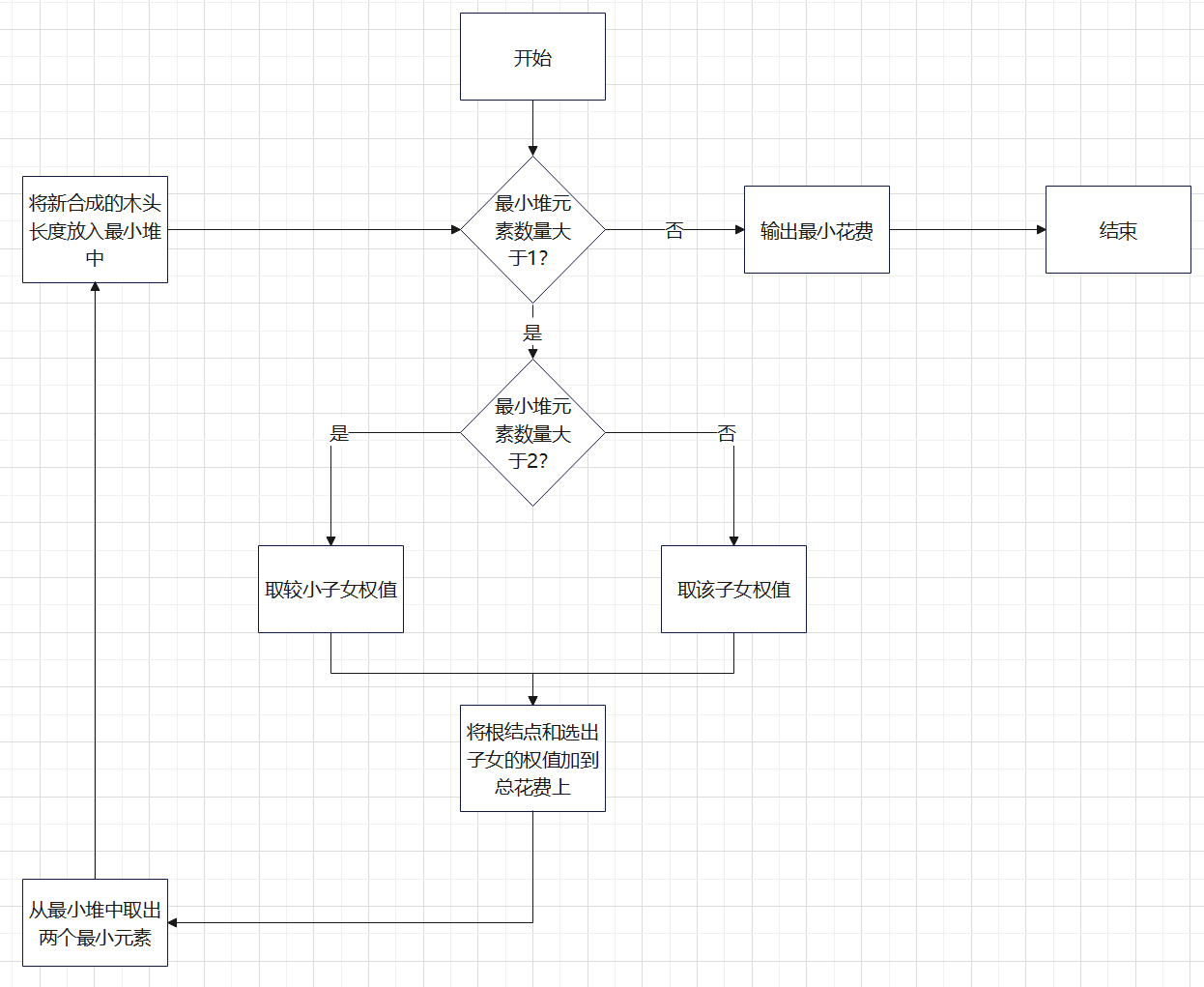
            cout << "第" << i + 1 << "块木头长度输入错误，请重新输入" << endl;

        }

    }

**2.3.4 求解最小花费部分**

·流程图表示



·代码实现

MinHeap<int> woods(data, num);  /\*存储每根木头长度的最小堆\*/

    int cost = 0;  /\*总花费\*/

    int minor\_child\_pos = -1;  /\*小子女所在位置\*/

    while (woods.GetSize() > 1) {

        if (woods.GetSize() > 2)  /\*若最小堆元素大于2，取小子女所在位置\*/

            minor\_child\_pos = (woods[1] > woods[2]) ? 2 : 1;

        else  /\*若最小项仅有一个子女，则取该子女位置\*/

            minor\_child\_pos = 1;

        cost += woods[0] + woods[minor\_child\_pos];  /\*将最小项和小子女加到总花费上\*/

        int new\_wood = woods[minor\_child\_pos] + woods[0];  /\*将最短的两根木头合成为一根\*/

        int get = 0;

        woods.RemoveMin(get);  /\*从最小堆中移除最小的两个元素\*/

        woods.RemoveMin(get);

        woods.Insert(new\_wood);  /\*将新合成的木头放入最小堆中\*/

    }

    cout << cost << endl;  /\*输出最小花费\*/

    delete[] data;

    return 0;

**3.项目测试**

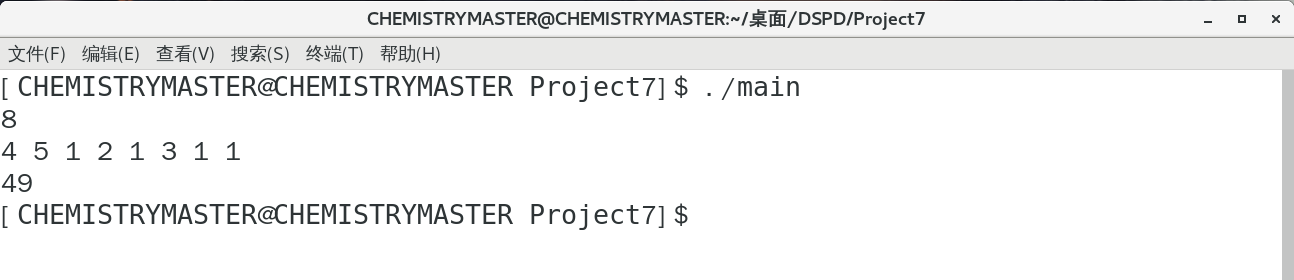
3.1 样例测试

输入内容：8

4 5 1 2 1 3 1 1

预期结果：49

测试结果：



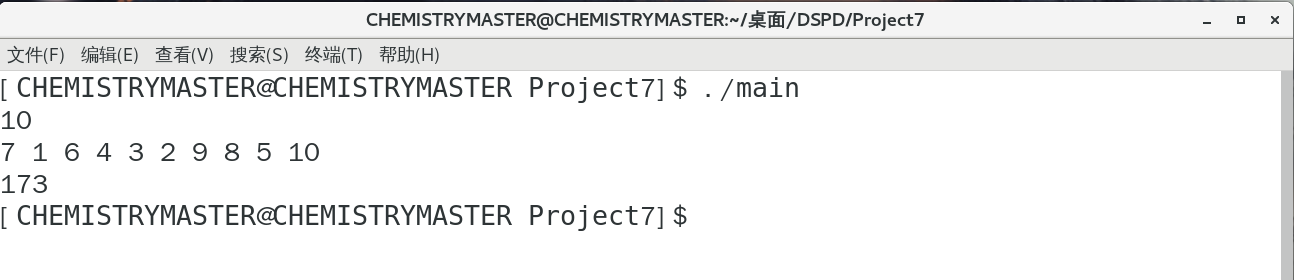
3.2 所有木头不等长

输入内容：10

7 1 6 4 3 2 9 8 5 10

预期结果：173

测试结果：



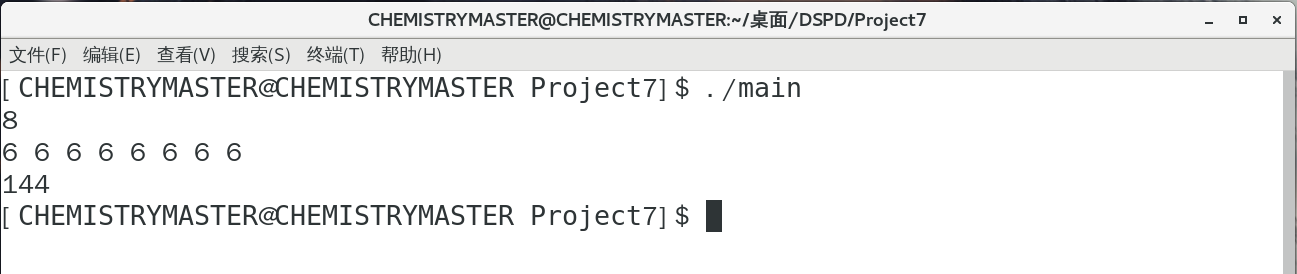
3.3 所有木头等长

输入内容：8

6 6 6 6 6 6 6 6

预期结果：144

测试结果：



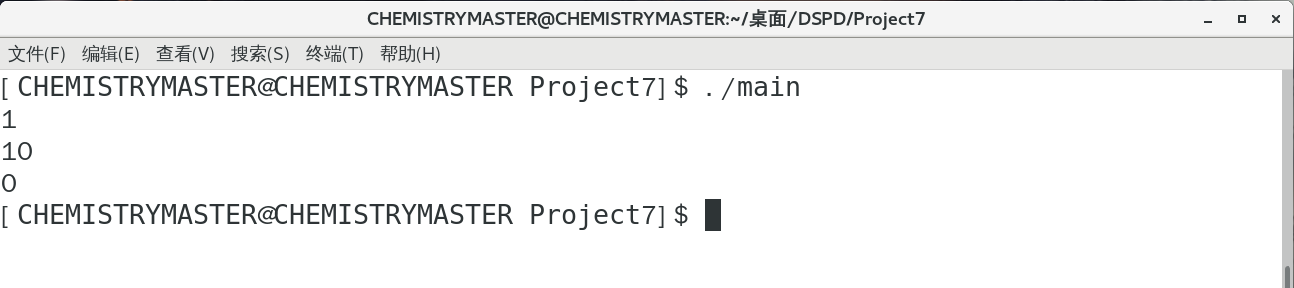
3.4 只需要一根木头

输入内容：1

0

预期结果：0

测试结果：



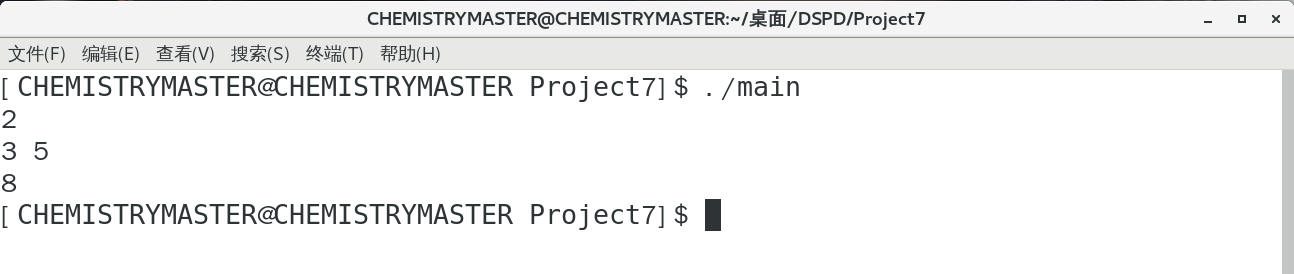
3.5 只需要两根木头

输入内容：2

3 5

预期结果：8

测试结果：

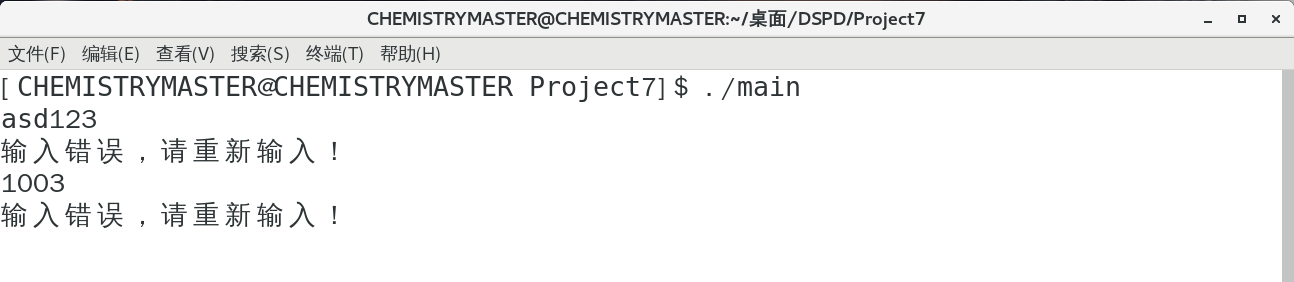


3.6 木头总数输入错误的情况

输入内容：asd123 1003

预期结果：（提示错误，直至输入正确为止）

测试结果：



3.7 木头长度输入错误的情况

输入内容：10

1 3 5 7 9 a 4 6 8 10

预期结果：（提示第几根木头长度输入错误，需从该木头开始继续输入）

测试结果：

