项目文档

谈瑞

项目七：牧场栅栏问题

专业班级：软件工程 4 班 学号：1452775 课目：数据结构课程设计
电话：18936361545 电子邮件：tanrui106@163.com

目录

项目简介 3
项目概要 3
项目功能及要求 3
项目结构 4
基本思路 4
示例说明 5
项目类的实现 6
Woods 类 6
主要代码分析 7
一些思考 8
明确用户输入的数据到底是什么 8
编译器不同，执行代码行为亦不同 8
Lambda 函数 8
运行测试 10
控制台输入 10
文件输入 10
错误检测 11

项目简介

项目概要

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）.

项目功能及要求

此问题的数学建模如下，即对于一个正整数数列，每次选取 2 个数相加，将和添加到数列中去，然后删除这 2 个数，如此循环知道数列中最后剩下一个数。而此题就要求出使得最后一个数最小的情况。很容易想到需要构建 Huffman 树来实现此功能。

具体做法是：先将输入的 N（N<=104）个整数排序，使其按照从小到大的顺序排列（目的是使数组满足 Huffman 树的基本条件），实际上，构成 Huffman 树的元素并不是输入的整数数组，而应是锯成 N 块木头的前一个状态，此时的 ceil（N/2）（ceil 指向上取整）个元素才是 Huffman 树的真正的叶子结点。要得到这样的叶子结点，即是将 N 个整数两两相加（这里要注意 N 若是奇数则会多出来一个数，应对措施是先从总价钱 cost 中减去这个数，然后将其视作叶子结点，将其按照大小放置于正确位置即对 ceil（N/2）个整数再排序），此时用得到的 ceil（N/2）个整数构建 Huffman 树，而后遍历 Huffman 树，每遍历一个结点将其中的长度作为价格加到 cost 上，遍历完成即得到最终的花费，而此花费也正是各个叶子结点的权值与其到根结点的路径乘积的和，某结点到根结点的路径长度即可理解为农夫从整块木头到第一次砍出来这一块木头的次数。

此项目支持控制台输入及文件输入，初始文档中已经有 1000 个随机数，初测能得出正确花费。

项目结构

基本思路

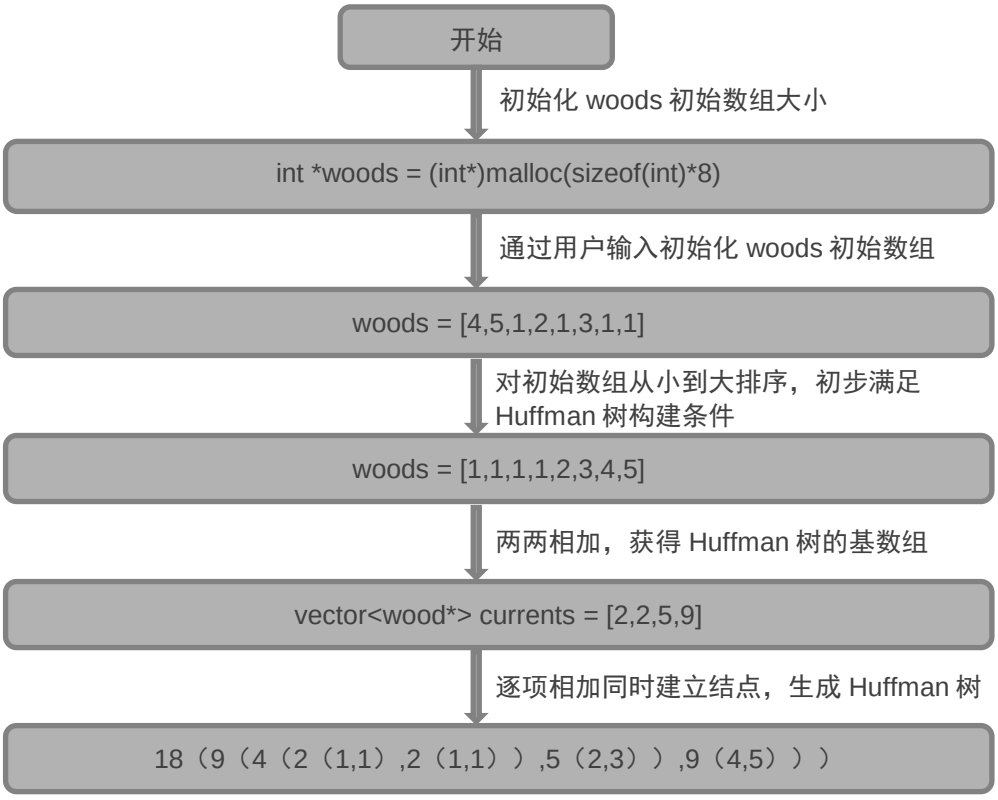
本项目采用 Huffman 树结构，依照 Huffman 树的“带权路径长度最小的二叉树应是权值大的外结点离根结点最近的扩充二叉树特性”，将木头长度视作结点权值，构建 Huffman 树，而后遍历树得到结果。

此题特殊点在于，构建 Huffman 树的基础数组并非用户输入的数组，具体原因在于，Huffman 树在求解带权路径长度时是将叶子结点的权值乘以其到根结点的路径，若采用输入的整数数组作为基数组，则会求解时会多加一遍这个数组，当然若这样做之后将总结果减去基数组的总和也是不可取的，应该使最后存储的结构在去掉基数组中的木头后成为一个标准的 Huffman 树，才能得到正确的结果。

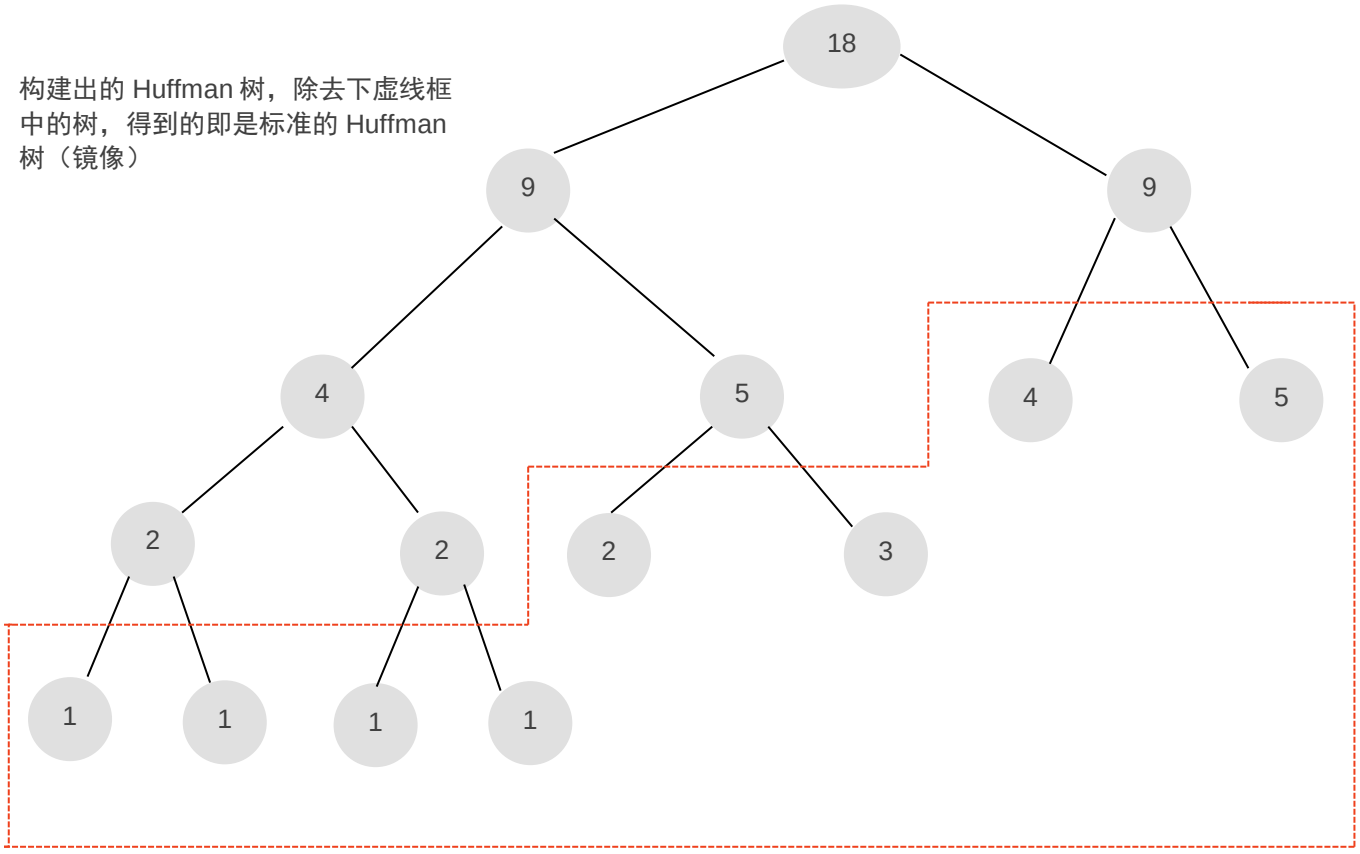
当然，为何排了序后的输入数组两两相加就能得到想要的正确的基数组呢，这是数学证明的范畴了，我这里也是假设以及通过对比验证默认认为此种方法得出的基数组而生成的 Huffman 树在所有可能的 Huffman 树中的总花费最小。

示例说明

针对输入 8:（4,5,1,2,1,3,1,1）讲述工作原理



构建出的 Huffman 树，除去下虚线框中的树，得到的即是标准的 Huffman 树（镜像）



项目类的实现

Woods 类

采用 Huffman 树数据结构

	类成员	作用
public 成员	struct Wood	Woods 类的结点，包含当前结点（即木头）的长度，以及左右子节点
	static long cost = 0	砍出所需木头的总花费，设置为静态变量，非类成员
	Woods()	默认构造函数
	Woods(istream &in)	带有输入流参数的构造函数
	~Woods()	默认析构函数
	void getWoods(istream &in = cin)	获取用户输入的主要数据，默认输入流为标准输入流
	void setWoodsTree()	通过初始数组构建基数组，而后根据基数组构建 Huffman 树
	void getCost()	获取最终花费
	void printWoodsHuffmanTree (Wood *wood)	打印 Huffman 树，观察其层次结构
private 成员	void PreOrder(Wood *current,void (*visit)(Wood *wood))	前序遍历函数，只在类内部调用
	void sortWoods();	对初始数组进行排序，采用希尔排序算法（从项目十中直接拷过来用的 Hmm..）
	Wood* mergeWood(Wood *lwood, Wood *rwood)	通过左右子结点构建二叉树，只在类内部调用
	bool ifInputValid(string str)	检测用户输入的初始数据是否有效
	int amount;	内部成员，木头的数量
	Wood* root	Huffman 树的根，该节点的 length 值一定是整块木头的长度
	int* woods	初始数组，储存用户输入的最终木头长度
	bool isOdd	存储木头数量是否为奇数，在后面确定 Huffman 构建行为中起到一些作用

主要代码分析

```
void Woods::setWoodsTree() {
    // 通过读入的数组，为 Huffman 树创建基数组，这是本项目的核心函数
    sortWoods();
    // 先将数组排序，方便构建 Huffman 树的基本条件
    if (amount == 1) {
        cout << "你不需要任何花费！\n";
        return;
    }
    // 当只需要 1 根木头时，那就是不需要砍，花费为 0
    isOdd = amount % 2 == 0 ? false : true;
    // 对于用户输入的数组，可能有奇数项和偶数项两种情况，用个变量进行标记，项目文档中会讨论其工作原理
    vector<Wood *> currents(amount / 2 + isOdd);
    //Wood *currents[amount/2+isOdd];
    //一开始在 clion 中直接使用数组，而转移到 vs 中，发现 vs 不支持在声明数组大小时使用非常量参数，就很头疼，于是改用 vector 存储
    // 之所以选择 vector 存储，一是因为上一行所述，二是因为 algorithm 中自带标准容器排序算法，美滋滋
    Wood *currentL, *currentR;
    for (int i = 0; i < amount / 2; i++) {
        currentL = new Wood(woods[2 * i], nullptr, nullptr);
        currentR = new Wood(woods[2 * i + 1], nullptr, nullptr);
        currents[i] = mergeWood(currentL, currentR);
    }
    if (isOdd) {
        currents[amount / 2] = new Wood(woods[amount - 1], nullptr,
nullptr);
        sort(currents.begin(), currents.end(), [](Wood *wood1, Wood
*wood2) {
            return wood1->length < wood2->length;
        });
        //自定义比较函数（lambda 函数），原因在于若是奇数组，可能最后一个数未参与前面的两两相加，而其又并不是 Huffman 数奇数组中最大的一个，故需排序以获取正确数列
    }
    Wood *currentP = mergeWood(currents[0], currents[1]);
    for (int j = 2; j < amount / 2 + isOdd; j++) {
        currentP = mergeWood(currentP, currents[j]);
    }
    // 一个循环构建 Huffman 树，事实上，这里构建出的树与标准 Huffman 树互成镜像，但是原理以及行为啥的都是如出一辙
    root = currentP;
    getCost();
    cout << "最小花费为：" << cost << "RMB!\n";
    // 获取最后的花费
    printWoodsHuffmanTree(root);
    // 将构建出的 Huffman 树打印出来可通过括号包含关系看出其层次结构
}
```

一些思考

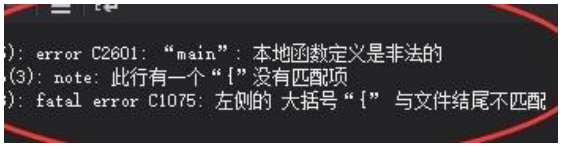
此函数是本项目中的主要函数，主要循着接受用户输入→创建基数组→构建 Huffman 树→运算得出最终花费。在编写过程中，有以下几点思考：

明确用户输入的数据到底是什么

看到题设时，一开始考虑使用用户输入的初始数组直接创建 Huffman 树而后直接通过求 Huffman 树路径长度得出总花费，然而结果与示例差别很大，后来想想可能是因为重复加了初始数组，于是减去了初始数组的值，发现与示例答案相似了但还是差了几块钱，这说明这种创建的 Huffman 树的方法根本就是错的。于是翻书重新查阅了 Huffman 树的特性，发现 Huffman 树求路径时每个叶子结点应是直接参与路径长度计算中，而这里的初始数组（暂且称初始木头吧）是不直接参与最终花费的运算的，继续想，直接参与最终花费的最底层木头应该使“砍出初始木头的那些木头”，即代码中的 `vector<Wood *> currents;`。因此代码中创建的以 root 为根的二叉树实际上是在 Huffman 树的基础上再让他每个叶子结点（即叶子木头）左右子女链接上其最后砍出的木头（即初始木头），这样的结构在 print 后可很清晰看出。

编译器不同，执行代码行为亦不同

此项目我一开始是在 Fedora 上用 C-lion 写的，应该说一路写下来很顺畅，但是等回到宿舍 pull 到台式机上使用 visual studio 调试却产生了一大堆错误，图中列出一部分错误（什么“常量中不能包含换行符”、“声明数组不能使用非常量表达式”、“大括号不匹配”等等），可以说很是崩溃了，扔到 Google 问了一下，发现罪魁祸首还是跨平台问题，visual studio 采用的是 vc++编译器（当然编译只是 vc++的一小部分功能），而 Fedora 上的 C-lion 使用的是 g++，多多少少会在字符编码上有点差别，况且这还是跨系统平台，本身 Linux 和 Windows 的换行符就是不一样的。这里我选择了笨办法，就是不直接复制文件，而是创建新文件而后复制文件内容。希望在以后的学习中，能够学会在代码中对编译器进行检测，从而让计算机根据编译器选择编译哪一段代码（此坑先放放）。



Lambda 函数

Lambda 函数是 C++11 中提出的一种定义匿名函数对象的方法（详见 https://msdn.microsoft.com/zh-cn/library/dd293608.aspx），这种方法我在项目九中运用的较多，即各种顺序遍历二叉树时，第二个参数传入的是一个函数指针，而在调用其时，就使用 lambda 函数来明确遍历结点时的具体操作。此项目中运用时还不是很了解其详细用

```
inline void InOrder(BSortTreeNode *_current, void(*visit)(BSortTreeNode *p)) {
```

```
inline void printTreeInOrder(BSortTreeNode *_current) {
    InOrder(_current, [](BSortTreeNode *p) {
        cout << p->data << "->";
    });
};
```

```
void Woods::getCost() {
    // 调用前序遍历函数，并创建lambda函数，对于除用户输入的最终木头长度数组外的所
    PreOrder(root, [](Wood *wood) {
        if (wood->lchild != nullptr && wood->rchild != nullptr) {
            cost += wood->length;
        }
    });
}
```

法，一开始 cost 变量是声明在类中的私有成员，在 lambda 函数中调用 cost 时提示 lambda 函数无法获取 cost 变量，因此干脆将其声明为静态全局成员，后来查阅 lambda 函数资料得知 capture 子句可选择按值或按引用来获取匿名函数外部的变量，传入 lambda 函数内部。还是无知啊！（但是因为懒也还是没有改回来惹，继续声明 cost 为静态全局变量）

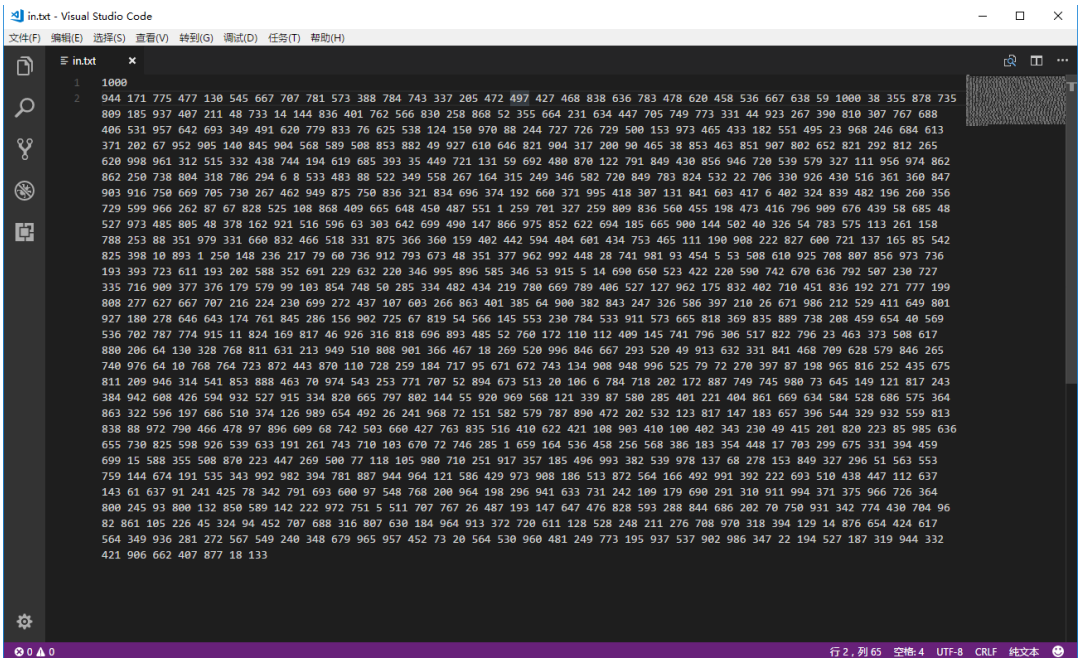
运行测试

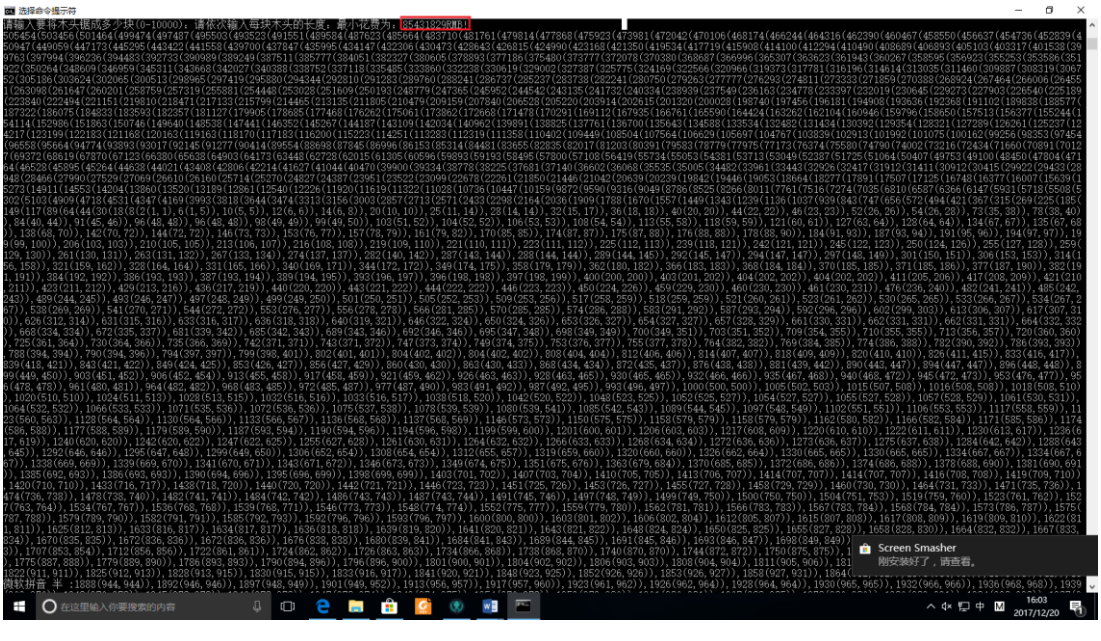
控制台输入

```
D:\tanrui\DataStructure\object7>main.exe
请选择输入方法（0-控制台输入，1-文件输入）:0
请输入要将木头锯成多少块(0-10000)：8
请依次输入每块木头的长度：4 5 1 2 1 3 1 1
最小花费为：49RMB!
18(9(4(2(1,1),2(1,1)),5(2,3)),9(4,5))
```

文件输入

输入的文件内容中包含木头的数量（此文件中是 1000），以及每块木头的长度（此文件中为 1000 个随机生成的数），据此得出最小花费。





错误检测

本项目输入的错误主要会出现在木头数量的输入中，在此添加了错误检测。而对木头的长度的上界并未加限制。如图所示，木头数量应在 0-10000 开区间内，而木头的长度限制在正整数。另外，还有两种情况，记输入的木头数量为 amount，输入的每块木头加起来的数量为 amount2，一是 amount>amount2 时，此时取 amount2 为真正的木头数量，如图中红框所示，二是 amount<amount2 时，此时取 amount 为真正的木头数量，如图中黄框所示。