同濟大學
TONGJI UNIVERSITY

项目说明文档

修理牧场

指导老师：张颖

1851009 沈益立

目录

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。

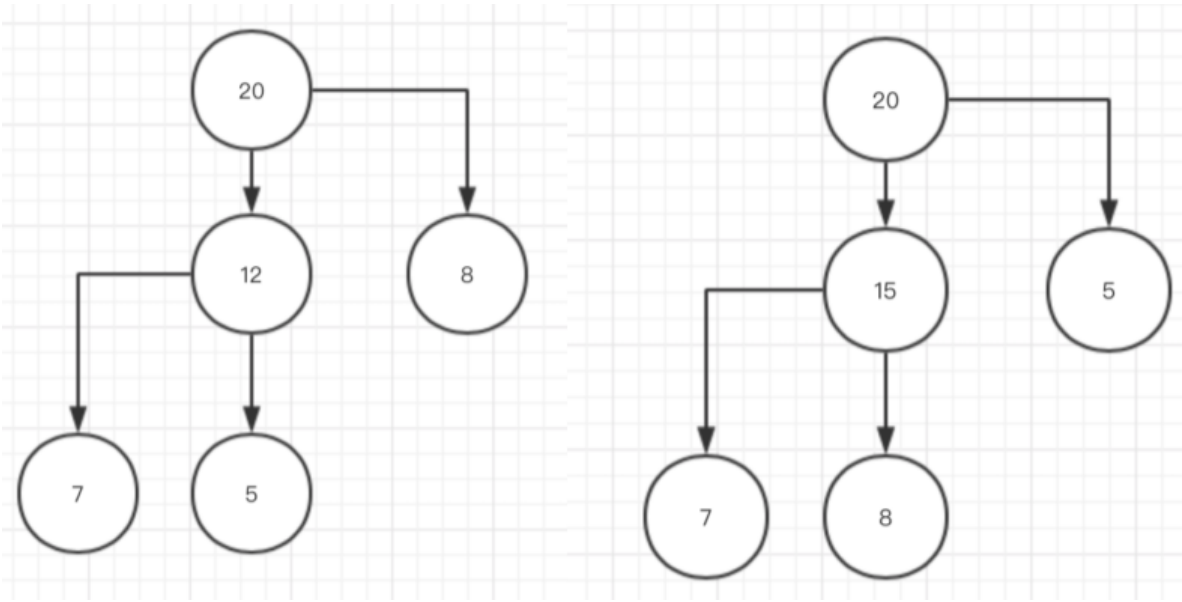
1.2 功能分析

本次设计的输入分为两行，第一行会给出所需要的锯成木头的节数，随后一行给出各段木头的长度，为了满足最基本的题目需求，需要输出他们所需要的最小花费。

2.设计

2.1 算法设计

由题意，只需要给出最后锯木头的最小花费即可。题目的样例中给出将长度为20的木头聚成8、7、5三段的情况，很容易能想到如下两种锯法：



其中，左侧锯法的花费为32，而右侧锯法的花费为35，在计算时很容易发现，深度为`i`的叶子节点将会被重复累加`i`次。

这种数据结构恰好满足带权路径长度(Weighted Path Length / WPL)的要求，而求解最短带路径的长度可以使用构造Huffman树的方法。这种方法的核心思想是：将尽量小的数放到尽量深的层数，以使全局的路径和最短。

2.2 数据结构设计

由于本题并未要求输出具体规划出来的策略，只要求求出最小花费，那么比较高效的做法是利用霍夫曼树的原理而组织出一个小根堆，每次操作依次弹出小根堆中最小的两个数加到结果中，再将这两个数之和推入小根堆，直到堆空为止。这样仅仅用一个堆的方法就可以求得结果，但是会缺失具体的流程信息；但与之相对的，在能完成要求的情况下这种方法结构简单、代码可读性更高。

2.3 成员和操作设计

小根堆类`MinHeap`

公有操作:

```
MinHeap();                  //无参构造函数
MinHeap(int size)           //含大小的构造函数
    :m_maxSize(size) {
        m_heap = new T[size];
        auto p = m_heap;
        for (int i = 0; i < size; i++) {
            *p = 0;
            p++;
        }
    };
void insert(T val);         //向小根堆插入新元素
T pop();                    //小根堆弹出元素
T getTop();                 //获取堆顶值
bool isEmpty();             //判断堆空
bool isFull();              //判断堆满
~MinHeap() {};              //默认析构函数
```

私有操作:

```
void filterUp(int start);//自底向上调整堆
void filterDown(int start, int end);//自顶向下调整堆
```

私有成员:

```
T* m_heap;                  //用数组模拟堆
int m_maxSize;              //堆的最大大小
int m_defaultSize = 10;     //默认大小
int m_curSize = 0;          //当前堆的大小
```

修理牧场类`RepairFarm`

公有操作:

```
RepairFarm() {};                    //默认构造函数
~RepairFarm() {};                   //默认析构函数
void initIO();                      //初始化IO
void processHeap();                 //处理堆
```

私有成员：

```
int m_totalNum = 0;                 //木头总数
MinHeap<int> m_minHeap;             //小根堆成员
int m_result = 0;                   //输出结论
bool m_fLegalInput = 1;             //输入合法标记
```

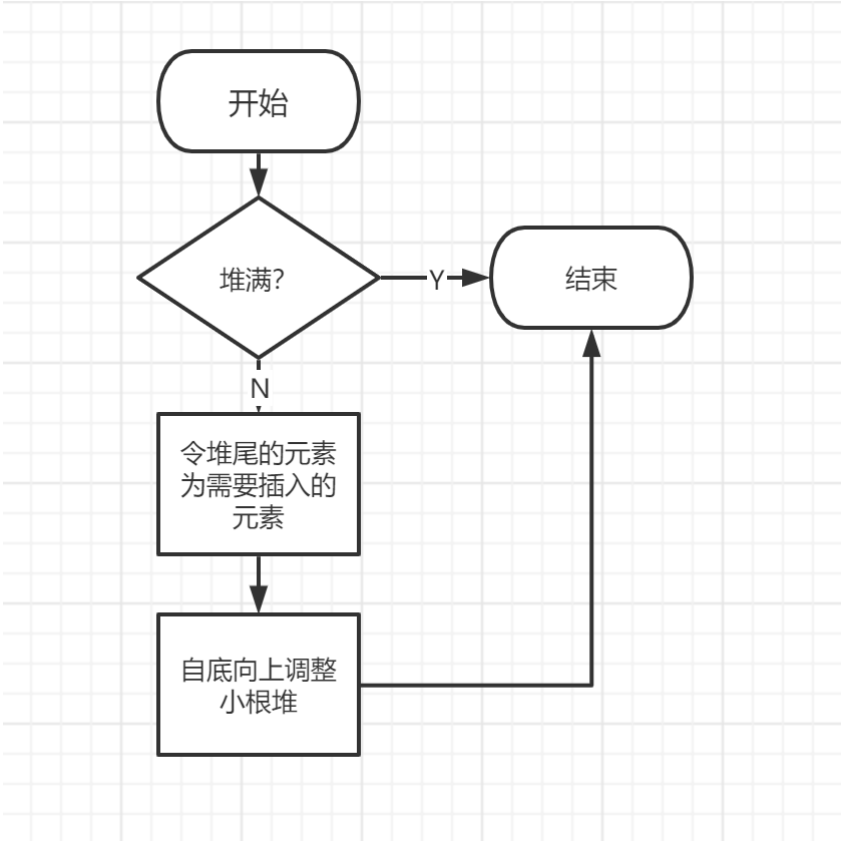
2.4 系统设计

程序运行后，系统会要求用户输入木头的总数并且据此创建小根堆，之后要求用户依次输入木头长度，此时将它们推进小根堆。随后经过对小根堆的处理，取堆顶的两个元素相加再推入堆中，直到堆空为止，随后输出结果到屏幕上。

3.实现

3.1 小根堆压入功能的实现

3.1.1 小根堆压入功能流程图

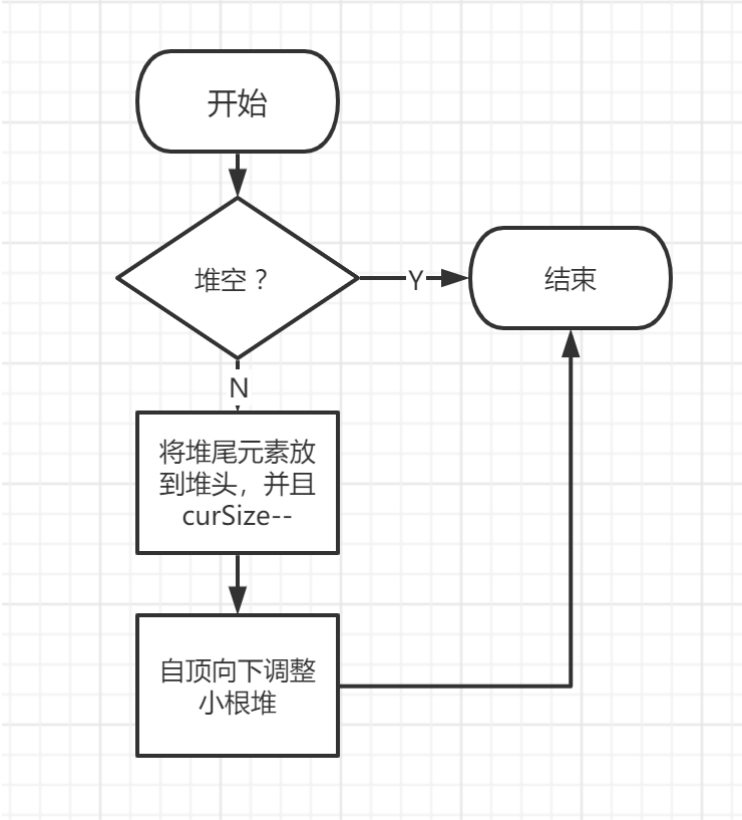


3.1.2 小根堆压入功能核心代码实现

```
void MinHeap<T>::insert(T val) {
    if (isFull()) {
        cerr << "堆满，请建更大的堆" << endl;
        return;
    }
    //插入逻辑，尾插后上浮调整
    m_heap[m_curSize] = val;
    filterUp(m_curSize++);
}
```

3.2 小根堆弹出功能的实现

3.2.1 小根堆弹出功能流程图

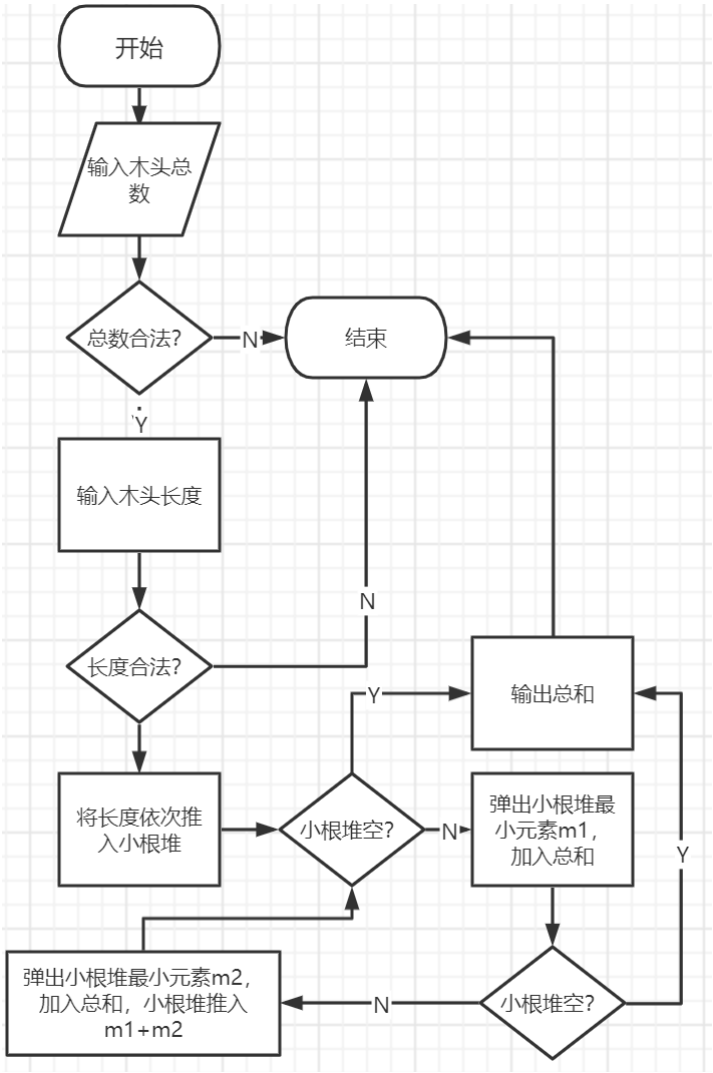


3.2.2 小根堆弹出功能核心代码实现

```
template<class T>
T MinHeap<T>::pop() {
    if (isEmpty()) {
        cerr << "空集，无法弹出堆顶元素" << endl;
        return T();
    }
    T retVal = m_heap[0];
    m_heap[0] = m_heap[m_curSize - 1];  //取堆尾元素
    m_curSize--;
    filterDown(0, m_curSize - 1);   //自上而下调整堆
    return retVal;
}
```

3.3 处理小根堆得到结果功能的实现

3.3.1 得到结果功能流程图



3.3.2 得到结果核心代码实现

```
void RepairFarm::initIO() {
    cout << "请输入要锯的木头数:";
    cin >> m_totalNum;
    m_minHeap = MinHeap<int>(m_totalNum + 1);
    while (m_totalNum <= 0) {
        cout << "木头总数错误, 请重输入:";
        cin >> m_totalNum;
    }
```

```
    cout << "请分别输入木头长度:";
    for (int i = 0; i < m_totalNum; i++) {
        int tmp;
        cin >> tmp;
        if (tmp < 0) {
            cerr << "请不要输入负数" << endl;
            m_fLegalInput = 0;
            return;
        }
        m_minHeap.insert(tmp);
    }
}

void RepairFarm::processHeap() {
    if (m_fLegalInput == 0) {
        return;
    }
    while (1) {
        int min1 = m_minHeap.pop();
        m_result += min1;
        if (m_minHeap.isEmpty()) {
            break;
        }

        int min2 = m_minHeap.pop();
        m_result += min2;
        if (m_minHeap.isEmpty()) {
            break;
        }
        m_minHeap.insert(min1 + min2);
    }
}
```

4.测试

4.1 功能测试

4.1.1 一般情况测试1

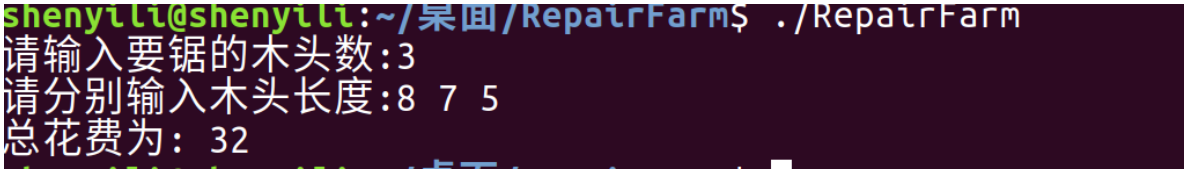
测试用例：

```
3
8 7 5
```

预期结果：

```
32
```

实验结果：



4.1.2 一般情况测试2

测试用例：

```
4
2 4 5 7
```

预期结果：

```
35
```

实验结果：

```
shenyili@shenyili:~/桌面/RepairFarm$ ./RepairFarm
请输入要锯的木头数:4
请分别输入木头长度:2 4 5 7
总花费为: 35
```

4.1.3 一般情况测试3

测试用例:

```
8
4 5 1 2 1 3 1 1
```

预期结果:

```
49
```

实验结果:

```
shenyili@shenyili:~/桌面/RepairFarm$ ./RepairFarm
请输入要锯的木头数:8
4 5 1 2 1 3 1 1请分别输入木头长度:
总花费为: 49
```

4.1.4 两位数测试

测试用例:

```
10
8 5 6 4 3 22 5 4 1 3 1
```

预期结果:

```
180
```

实验结果:

```
shenyili@shenyili:~/桌面/RepairFarm$ ./RepairFarm
请输入要锯的木头数:10
8 5 6 4 3 22 5 4 1 3 1请分别输入木头长度:
总花费为: 180
```

4.2 代码鲁棒性测试

4.2.1 错误总数输入

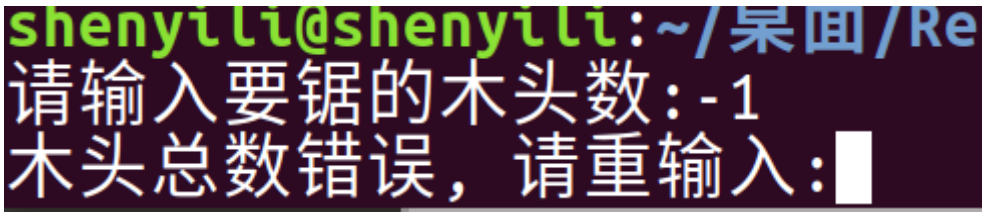
测试用例：

```
-1
```

预期结果：

```
提示总数错误
```

实验结果：



4.2.2 错误木头长度输入

测试用例：

```
1
-1
```

预期结果：

```
提示输入负数
```

实验结果：